



# analyse-R

## Introduction à l'analyse d'enquêtes avec R et RStudio

Dernière mise à jour : 21 janvier 2021

DOI [10.5281/zenodo.4067185](https://doi.org/10.5281/zenodo.4067185)

## Contributeurs

**Par ordre alphabétique :**

Julien Barnier, Julien Biaudet, François Briatte, Milan Bouchet-Valat, Ewen Gallic, Frédérique Giraud, Joël Gombin, Mayeul Kauffmann, Christophe Lalanne, Joseph Larmarange, Nicolas Robette.

**Création et Maintenance :**

Joseph Larmarange – <http://joseph.larmarange.net>

# Présentation

L'objectif premier d'**analyse-R** est de présenter comment réaliser des analyses statistiques et diverses opérations courantes (comme la manipulation de données ou la production de graphiques) avec **R**. Il ne s'agit pas d'un cours de statistiques : les différents chapitres presupposent donc que vous avez déjà une connaissance des différentes techniques présentées. Si vous souhaitez des précisions théoriques / méthodologiques à propos d'un certain type d'analyses, nous vous conseillons d'utiliser votre moteur de recherche préféré. En effet, on trouve sur internet de très nombreux supports de cours (sans compter les nombreux ouvrages spécialisés disponibles en librairie).

# Table des matières

Si vous débutez avec **R** et **RStudio**, nous vous conseillons de parcourir en premier lieu les chapitres suivants :

1. Manipuler > Prise en main
2. Analyser > Statistiques introducives
3. Manipuler > Manipulations de données
4. Analyser > Statistiques intermédiaires

puis de compléter votre lecture en fonction de vos besoins.

## Manipuler

### Prise en main

Présentation et Philosophie . . . . .	7
Installation de R et RStudio . . . . .	11
Premier contact . . . . .	13
Premier travail avec des données . . . . .	33
Extensions (installation, mise à jour) . . . . .	55
Introduction au <b>tidyverse</b> . . . . .	59
Vecteurs, indexation et assignation . . . . .	67
Listes et Tableaux de données . . . . .	87
Facteurs et vecteurs labellisés . . . . .	117
Organiser ses fichiers . . . . .	135
Import de données . . . . .	145
Où trouver de l'aide ? . . . . .	163

### Manipulation de données

Visualiser ses données . . . . .	175
Recodage de variables . . . . .	197
Manipuler les données avec <b>dplyr</b> . . . . .	233
Manipulations avancées avec <b>data.table</b> . . . . .	251
Tris . . . . .	257

Sous-ensembles . . . . .	261
Fusion de tables . . . . .	269
Gestion des dates . . . . .	285
Fonctions à fenêtre . . . . .	287
Manipuler du texte avec <b>stringr</b> . . . . .	289
Réorganiser ses données avec <b>tidyverse</b> . . . . .	299
Scraping . . . . .	317

## Exporter

Export de données . . . . .	327
Export de graphiques . . . . .	331

# Analyser

## Statistiques introducives

Statistique univariée . . . . .	339
Statistique bivariée . . . . .	365
Introduction à <b>ggplot2</b> , la grammaire des graphiques . . . . .	395
Graphiques univariés et bivariés avec <b>ggplot2</b> . . . . .	417
Données pondérées . . . . .	465

## Statistiques intermédiaires

Intervalles de confiance . . . . .	477
Comparaisons (moyennes et proportions) . . . . .	485
Définir un plan d’échantillonnage complexe . . . . .	501
Régression linéaire . . . . .	505
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Analyse des correspondances multiples (ACM) . . . . .	557
Classification ascendante hiérarchique (CAH) . . . . .	595

## Statistiques avancées

Effets d’interaction dans un modèle . . . . .	617
---	-----

Multicolinéarité dans la régression .....	645
Analyse de survie .....	651
Analyse de séquences .....	679
Trajectoires de soins : un exemple de données longitudinales .....	705
Analyse de réseaux .....	785
Analyse spatiale .....	787

## Approfondir

### Graphiques

<b>ggplot2</b> et la grammaire des graphiques .....	789
Étendre <b>ggplot2</b> .....	793
Combiner plusieurs graphiques .....	825
Exemples de graphiques avancés .....	839
Graphiques interactifs .....	875
<b>lattice</b> : graphiques et formules .....	879
Cartes .....	895
Autres extensions graphiques .....	897

### Programmation

Conditions et comparaisons .....	925
Formules .....	929
Structures conditionnelles .....	945
Vectorisation .....	953
Expressions régulières .....	977
<b>R Markdown</b> : les rapports automatisés .....	979

### Divers

Mettre en forme des nombres .....	1019
Couleurs et Palettes .....	1021
Annotations mathématiques .....	1097
Calculer un âge .....	1103

Diagramme de Lexis . . . . .	1111
------------------------------	------

## Index

Index des concepts . . . . .	1113
Index des fonctions . . . . .	1129
Index des extensions . . . . .	1153

## Licence

Le contenu de ce site est diffusé sous licence *Creative Commons Attribution - Pas d'utilisation commerciale - Partage dans les mêmes conditions* (<https://creativecommons.org/licenses/by-nc-sa/3.0/fr/>).



CC by-nc-sa

Cela signifie donc que vous êtes libre de recopier / modifier / redistribuer les contenus d'**analyse-R**, à condition que vous citiez la source et que vos modifications soient elle-mêmes distribuées sous la même licence (autorisant ainsi d’autres à pouvoir réutiliser à leur tour vos ajouts).

## Contribuer

**analyse-R** est développé avec **RStudio** et le code source est librement disponible sur **GitHub** :  
<https://github.com/larmarange/analyse-R>.

Ce projet se veut collaboratif. N’hésitez donc pas à proposer des corrections ou ajouts, voire même à rédiger des chapitres additionnels.

# Présentation et Philosophie

Présentation de R .....	7
Philosophie de R .....	8
Présentation de RStudio .....	9

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #01 (premier contact avec R & RStudio) sur [YouTube](#).

## Présentation de R

R est un langage orienté vers le traitement de données et l'analyse statistique dérivé du langage **S**. Il est développé depuis une vingtaine d'années par un groupe de volontaires de différents pays. C'est un logiciel libre<sup>1</sup>, publié sous licence GNU GPL.

L'utilisation de **R** présente plusieurs avantages :

- c'est un logiciel multiplateforme, qui fonctionne aussi bien sur des systèmes **Linux**, **Mac OS X** ou **Windows** ;
- c'est un logiciel libre, développé par ses utilisateurs et modifiable par tout un chacun ;
- c'est un logiciel gratuit ;
- c'est un logiciel très puissant, dont les fonctionnalités de base peuvent être étendues à l'aide de plusieurs milliers d'extensions ;
- c'est un logiciel dont le développement est très actif et dont la communauté d'utilisateurs ne cesse de s'élargir ;
- les possibilités de manipulation de données sous **R** sont en général largement supérieures à celles des autres logiciels usuels d'analyse statistique ;
- c'est un logiciel avec d'excellentes capacités graphiques et de nombreuses possibilités d'export ;
- avec **Rmarkdown**<sup>2</sup>, il est devenu très aisément de produire des rapports automatisés dans divers

---

1. Pour plus d'informations sur ce qu'est un logiciel libre, voir : <http://www.gnu.org/philosophy/free-sw.fr.html>.

- format (**Word, PDF, HTML, ...**) ;
- **R** est de plus utilisé dans tous les secteurs scientifiques, y compris dans le domaine des analyses d'enquêtes et, plus généralement, des sciences sociales.

Comme rien n'est parfait, on peut également trouver quelques inconvénients :

- le logiciel, la documentation de référence et les principales ressources sont en anglais. Il est toutefois parfaitement possible d'utiliser **R** sans spécialement maîtriser cette langue ;
- il n'existe pas encore d'interface graphique pour **R** équivalente à celle d'autres logiciels comme **SPSS** ou **Modalisa**. **R** fonctionne à l'aide de scripts (des petits programmes) édités et exécutés au fur et à mesure de l'analyse et se rapprocherait davantage de **SAS** dans son utilisation (mais avec une syntaxe et une philosophie très différentes). Ce point, qui peut apparaître comme un gros handicap, s'avère après un temps d'apprentissage être un mode d'utilisation d'une grande souplesse ;
- comme **R** s'apparente davantage à un langage de programmation qu'à un logiciel proprement dit, la courbe d'apprentissage peut être un peu « raide », notamment pour ceux n'ayant jamais programmé auparavant.

Il est à noter que le développement autour de **R** a été particulièrement actif ces dernières années. On trouvera dès lors aujourd'hui de nombreuses extensions permettant de se « faciliter la vie » au quotidien, ce qui n'était pas vraiment encore le cas il y a 5 ans.

## Philosophie de R

Quelques points particuliers dans le fonctionnement de **R** peuvent parfois dérouter les utilisateurs habitués à d'autres logiciels :

- Sous **R**, en général, on ne voit pas directement les données sur lesquelles on travaille ; on ne dispose pas en permanence d'une vue des données sous forme de tableau<sup>3</sup>, comme sous **Modalisa** ou **SPSS**. Ceci peut être déroutant au début, mais on se rend vite compte qu'on n'a pas besoin de voir en permanence les données pour les analyser.
- Alors qu'avec la plupart des logiciels on réfléchira avec un fichier de données ouvert à la fois, sous **R** chaque fichier de données correspondra à un objet différent chargé en mémoire, permettant de manipuler très facilement plusieurs objets à la fois (par exemple dans le cadre de fusion de tables<sup>4</sup>).
- Avec les autres logiciels, en général la production d'une analyse génère un grand nombre de résultats de toutes sortes dans lesquels l'utilisateur est censé retrouver et isoler ceux qui l'intéressent. Avec **R**, c'est l'inverse : par défaut l'affichage est réduit au minimum et c'est l'utilisateur qui demande à voir des résultats supplémentaires ou plus détaillés.

---

2. Voir <http://rmarkdown.rstudio.com/>.

3. On verra qu'il est possible avec **RStudio** de disposer d'une telle vue.

4. Voir par exemple la section dédiée à ce sujet dans le [chapitre sur la manipulation de données](#).

- Sous R, les résultats des analyses sont eux aussi stockés dans des objets et sont dès lors manipulables.

Inhabituel au début, ce fonctionnement permet en fait assez rapidement de gagner du temps dans la conduite des analyses.

## Présentation de RStudio

L'interface de base de R est assez rudimentaire (voir figure ci-après).

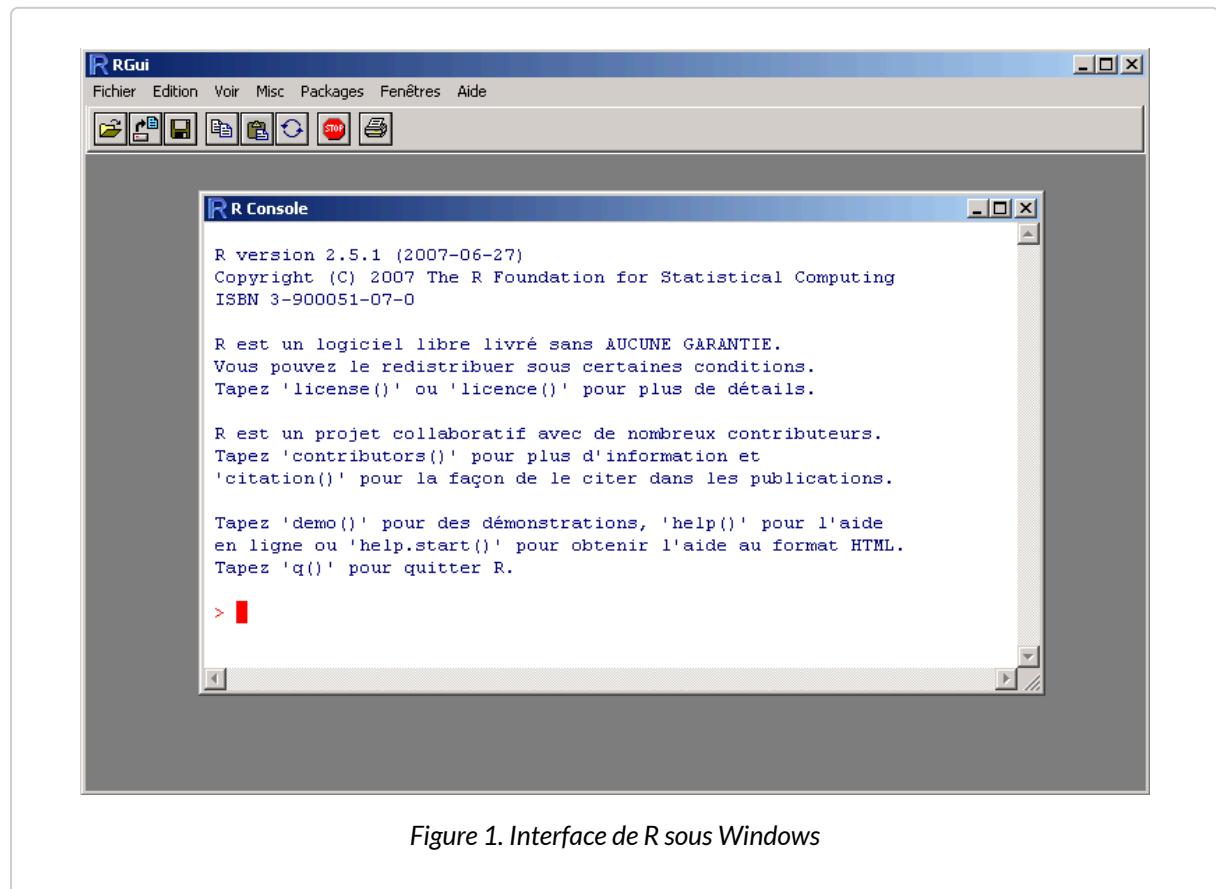


Figure 1. Interface de R sous Windows

RStudio est un environnement de développement intégré libre, gratuit, et qui fonctionne sous **Windows**, **Mac OS X** et **Linux**. Il complète R et fournit un éditeur de script avec coloration syntaxique, des fonctionnalités pratiques d'édition et d'exécution du code (comme l'autocomplétion), un affichage simultané du code, de la console R, des fichiers, graphiques et pages d'aide, une gestion des extensions, une intégration avec des systèmes de contrôle de versions comme git, etc. Il intègre de base divers outils comme par exemple la production de rapports au format **Rmarkdown**. Il est en développement actif et de nouvelles fonctionnalités sont ajoutées régulièrement. Son principal défaut est d'avoir une interface uniquement anglophone.

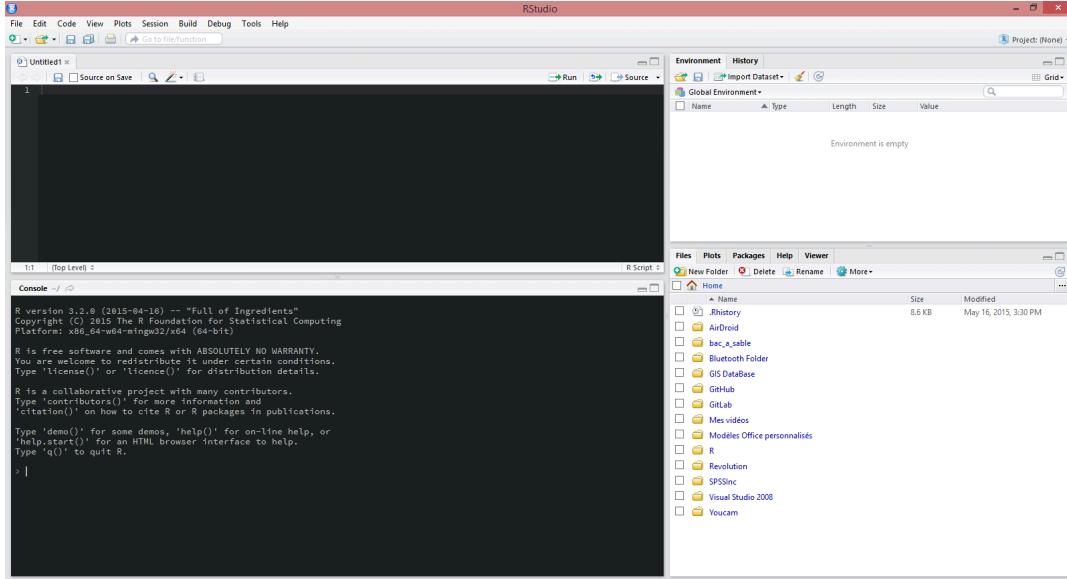


Figure 2. Interface de RStudio sous Windows

Pour une présentation plus générale de **RStudio** on pourra se référer au site du projet : <http://www.rstudio.com/>.

**RStudio** peut tout à fait être utilisé pour découvrir et démarrer avec **R**. Les différents chapitres d'**analyse-R** partent du principe que vous utilisez **R** avec **RStudio**. Cependant, à part les éléments portant sur l'interface de **RStudio**, l'ensemble du code et des fonctions **R** peuvent être utilisés directement dans **R**, même en l'absence de **RStudio**.

La documentation de **RStudio** (en anglais) est disponible en ligne à <https://support.rstudio.com>. Pour être tenu informé des dernières évolutions de **RStudio**, mais également de plusieurs extensions développées dans le cadre de ce projet, vous pouvez suivre le blog dédié <http://blog.rstudio.org/>.

# Installation de R et RStudio

Installation de R . . . . .	11
Installation de RStudio . . . . .	12
Mise à jour de R sous Windows . . . . .	12
Voir aussi . . . . .	12

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #01 (premier contact avec R & RStudio) sur [YouTube](#).

Il est préférable de commencer par installer **R** avant d'installer **RStudio**.

## Installation de R

Pour une installation sous **Windows**, on se rendra sur cette page : <http://cran.r-project.org/bin/windows/base/> et l'on suivra le premier lien pour télécharger le programme d'installation. Une fois le programme d'installation lancé, il suffira d'installer **R** avec les options par défaut<sup>1</sup>.

Pour **Mac OS X**, les fichiers d'installation sont disponibles à <http://cran.r-project.org/bin/macosx/>.

Si vous travaillez sous **Linux**, vous devriez pouvoir trouver **R** via votre gestionnaire de paquets, cela pouvant dépendre d'une distribution de **Linux** à une autre.

---

1. Dans le cas particulier où votre ordinateur est situé derrière un proxy, il est préférable de choisir *Options de démarrage personnalisées* lorsque cela vous sera demandé par le programme d'installation, puis *Internet2* lorsqu'on vous demandera le mode de connexion à Internet. Ainsi, **R** utilisera par défaut la configuration internet du navigateur **Internet Explorer** et prendra ainsi en compte les paramètres du proxy.

## Installation de RStudio

Une fois **R** correctement installé, rendez-vous sur <http://www.rstudio.com/products/rstudio/download/> pour télécharger la dernière version stable de **RStudio**. Plus précisément, il s'agit de l'édition *Open Source* de **RStudio Desktop** (en effet, il existe aussi une version serveur).

Choisissez l'installateur correspondant à votre système d'exploitation et suivez les instructions du programme d'installation.

Si vous voulez tester les dernières fonctionnalités de **RStudio**, vous pouvez télécharger la version de développement (plus riche en fonctionnalités que la version stable, mais pouvant contenir des bugs) sur <http://www.rstudio.com/products/rstudio/download/preview/>.

## Mise à jour de R sous Windows

Pour mettre à jour **R** sous **Windows**, il suffit de télécharger et d'installer la dernière version du programme d'installation.

Petite particularité, la nouvelle version sera installée à côté de l'ancienne version. Si vous souhaitez faire de la place sur votre disque dur, vous pouvez désinstaller l'ancienne version en utilisant l'utilitaire **Désinstaller un programme de Windows**.

Lorsque plusieurs versions de **R** sont disponibles, **RStudio** choisit par défaut la plus récente. Il est possible de spécifier à **RStudio** quelle version de **R** utiliser via le menu *Tools > Global Options > General*.

Petit défaut, les extensions (*packages*) sont installées par défaut sous **Windows** dans le répertoire `Documents de l'utilisateur > R > win-library > x.y` avec `x.y` correspondant au numéro de la version de **R**. Ainsi, si l'on travaillait avec la version 3.0 et que l'on passe à la version 3.2, les extensions que l'on avait sous l'ancienne version ne sont plus disponibles pour la nouvelle version. Une astuce consiste à recopier le contenu du répertoire `3.0` dans le répertoire `3.2`. Puis, on lancera **RStudio** (s'il était déjà ouvert, on le fermera puis relancera) et on mettra à jour l'ensemble des packages, soit avec la fonction, `update.packages` soit en cliquant sur *Update* dans l'onglet *Packages* du quadrant inférieur droit.

## Voir aussi

Un tutoriel détaillé en français sur le blog Quanti : <https://quanti.hypotheses.org/1813>.

# Premier contact

L'invite de commandes .....	14
Des objets .....	18
Objets simples .....	18
Vecteurs .....	21
Des fonctions .....	24
Arguments .....	25
Quelques fonctions utiles .....	26
Aide sur une fonction .....	27
Interprétation des arguments .....	28
Autocomplétion .....	30

## NOTE

Ce chapitre est inspiré de la section *Prise en main* du support de cours [Introduction à R](#) réalisé par Julien Barnier.

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #01 (premier contact avec R & RStudio) sur [YouTube](#).

Une fois **RStudio** lancé, vous devriez obtenir une fenêtre similaire à la figure ci-après.

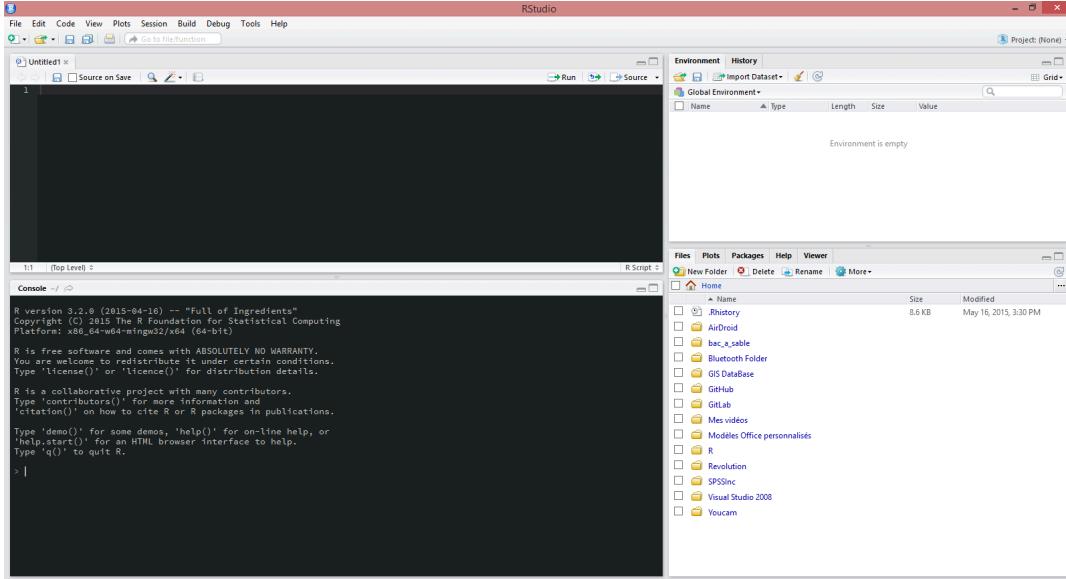


Figure 1. Interface de RStudio au démarrage

L'interface de **RStudio** est divisée en quatre quadrants :

- le quadrant supérieur gauche est dédié aux différents fichiers de travail (nous y reviendrons dans le chapitre Premier travail avec les données, page 33) ;
- le quadrant inférieur gauche correspond à ce que l'on appelle la *console*, c'est-à-dire à **R** proprement dit ;
- le quadrant supérieur droit permet de connaître
  - la liste des objets en mémoire ou environnement de travail (onglet *Environment*)
  - ainsi que l'historique des commandes saisies dans la console (onglet *History*) ;
- le quadrant inférieur droit affiche
  - la liste des fichiers du répertoire de travail (onglet *Files*),
  - les graphiques réalisés (onglet *Plots*),
  - la liste des extensions disponibles (onglet *Packages*),
  - l'aide en ligne (onglet *Help*)
  - et un *Viewer* utilisé pour visualiser certains types de graphiques au format web.

Inutile de tout retenir pour le moment. Nous aborderons chaque outil en temps utile. Pour l'heure, concentrons-nous sur la *console*, c'est-à-dire le quadrant inférieur gauche.

## L'invite de commandes

Au démarrage, la console contient un petit texte de bienvenue ressemblant à peu près à ce qui suit :

```
R version 3.2.0 (2015-04-16) -- "Full of Ingredients"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
```

>

suivi d'une ligne commençant par le caractère `>` et sur laquelle devrait se trouver votre curseur. Cette ligne est appelée l'invite de commande (ou prompt en anglais). Elle signifie que **R** est disponible et en attente de votre prochaine commande.

Nous allons tout de suite lui fournir une première commande. Tapez `2 + 3` dans la console et validez avec la touche **Entrée**.

`R> 2 + 3`

`[1] 5`

En premier lieu, vous pouvez noter la convention typographique utilisée dans ce document. Les commandes saisies dans la console sont indiquées sur un fond gris et précédé de `R>`. Le résultat renvoyé par **R** est quant à lui affiché juste en-dessous sur fond blanc.

Bien, nous savons désormais que **R** sait faire les additions à un chiffre<sup>1</sup>. Nous pouvons désormais continuer avec d'autres opérations arithmétiques de base :

`R> 8 - 12`

`[1] -4`

---

1. La présence du `[1]` en début de ligne sera expliquée par la suite dans la section sur les vecteurs, page 21.

```
R> 14 * 25
```

```
[1] 350
```

```
R> -3 / 10
```

```
[1] -0.3
```

```
R> -0.3
```

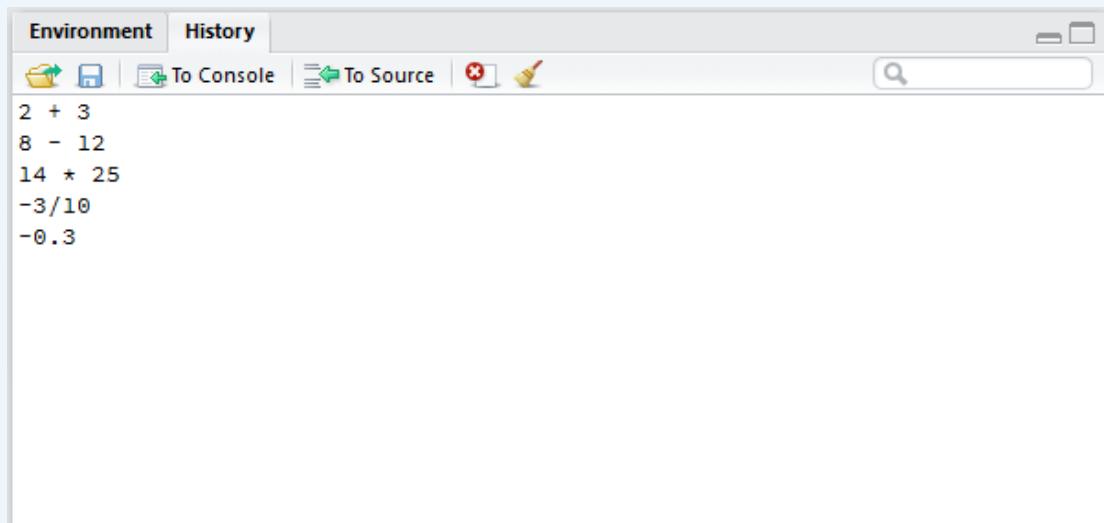
```
[1] -0.3
```

On remarquera que R est anglo-saxon. Les nombres sont donc saisies « à l'anglaise », c'est-à-dire en utilisant le point ( . ) comme séparateur pour les décimales.

**NOTE**

Une petite astuce très utile lorsque vous tapez des commandes directement dans la console : en utilisant les flèches Haut et Bas du clavier, vous pouvez naviguer dans l'historique des commandes tapées précédemment. Vous pouvez alors facilement réexécuter ou modifier une commande particulière.

Sous **RStudio**, l'onglet *History* du quadrant haut-droite vous permet de consulter l'historique des commandes que vous avez transmises à R.



Onglet History sous RStudio

Un double-clic sur une commande la recopiera automatiquement dans la console. Vous pouvez également sélectionner une ou plusieurs commandes puis cliquer sur *To Console*.

Voir également (en anglais) : <https://support.rstudio.com/hc/en-us/articles/200526217-Command-History>.

Lorsqu'on fournit à R une commande incomplète, celui-ci nous propose de la compléter en nous présentant une invite de commande spéciale utilisant les signe `+`. Imaginons par exemple que nous avons malencontreusement tapé sur `Entrée` alors que nous souhaitions calculer `4 * 3` :

```
R> 4 *
```

On peut alors compléter la commande en saisissant simplement `3` :

```
R> 4 *
```

```
  + 3
```

```
[1] 12
```

#### NOTE

Pour des commandes plus complexes, il arrive parfois qu’on se retrouve coincé avec une invite `+ sans plus savoir comment compléter la saisie correctement.` On peut alors annuler la commande en utilisant la touche `Echap` ou `Esc` sous Windows.

Sous Linux on utilise le traditionnel `Control + C`.

À noter que les espaces autour des opérateurs n’ont pas d’importance lorsque l’on saisit les commandes dans R. Les trois commandes suivantes sont donc équivalentes, mais on privilégie en général la deuxième pour des raisons de lisibilité du code.

```
R> 10+2
```

```
10 + 2
```

```
10      +      2
```

## Des objets

### Objets simples

Faire des opérations arithmétiques, c’est bien, mais sans doute pas totalement suffisant. Notamment, on aimerait pouvoir réutiliser le résultat d’une opération sans avoir à le resaisir ou à le copier/coller.

Comme tout langage de programmation, R permet de faire cela en utilisant des objets. Prenons tout de suite un exemple :

```
R> x <- 2
```

Que signifie cette commande ? L’opérateur `<-` est appelé opérateur d’assignation. Il prend une valeur quelconque à droite et la place dans l’objet indiqué à gauche. La commande pourrait donc se lire *mettre la valeur 2 dans l’objet nommé x*.

**IMPORTANT**

Il existe trois opérateurs d'assignation sous R. Ainsi les trois écritures suivantes sont équivalentes :

```
R> x <- 2  
x = 2  
2 -> x
```

Cependant, pour une meilleure lecture du code, il est conseillé de n'utiliser que `<-`. Ainsi, l'objet créé est systématiquement affiché à gauche. De plus, le symbole `=` sert également pour écrire des conditions ou à l'intérieur de fonctions. Il est donc préférable de ne pas l'utiliser pour assigner une valeur (afin d'éviter les confusions).

On va ensuite pouvoir réutiliser cet objet dans d'autres calculs ou simplement afficher son contenu :

```
R> x + 3
```

```
[1] 5
```

```
R> x
```

```
[1] 2
```

**NOTE**

Par défaut, si on donne à R seulement le nom d'un objet, il va se débrouiller pour nous présenter son contenu d'une manière plus ou moins lisible.

On peut utiliser autant d'objets qu'on veut. Ceux-ci peuvent contenir des nombres, des chaînes de caractères (indiquées par des guillemets droits doubles " ou simples ' ) et bien d'autres choses encore :

```
R> x <- 27  
y <- 10  
foo <- x + y  
foo
```

```
[1] 37
```

```
R> x <- "Hello"  
foo <- x  
foo
```

```
[1] "Hello"
```

### IMPORTANT

Les noms d'objets peuvent contenir des lettres, des chiffres, les symboles `.` et `_`. Ils doivent impérativement commencer par une lettre (jamais par un chiffre). R fait la différence entre les majuscules et les minuscules, ce qui signifie que `x` et `X` sont deux objets différents. On évitera également d'utiliser des caractères accentués dans les noms d'objets. Comme les espaces ne sont pas autorisés on pourra les remplacer par un point ou un tiret bas.

Enfin, signalons que certains noms courts sont réservés par R pour son usage interne et doivent être évités. On citera notamment `c`, `q`, `t`, `C`, `D`, `F`, `I`, `T`, `max`, `min` ...

Dans **RStudio**, l'onglet *Environment* dans le quadrant supérieur droit indique la liste des objets que vous avez précédemment créés, leur type et la taille qu'ils occupent en mémoire.

The screenshot shows the RStudio interface with the 'Environment' tab selected. The global environment contains three variables:

Name	Type	Length	Size	Value
foo	character	1	96 B	"Hello"
x	character	1	96 B	"Hello"
y	numeric	1	48 B	10

Figure 2. Onglet Environment de RStudio

## Vecteurs

Imaginons maintenant que nous avons interrogé dix personnes au hasard dans la rue et que nous avons relevé pour chacune d'elle sa taille en centimètres. Nous avons donc une série de dix nombres que nous souhaiterions pouvoir réunir de manière à pouvoir travailler sur l'ensemble de nos mesures.

Un ensemble de données de même nature constituent pour R un vecteur (en anglais vector) et se construit à l'aide d'une fonction nommée `c`<sup>2</sup>. On l'utilise en lui donnant la liste de nos données, entre parenthèses, séparées par des virgules :

```
R> tailles <- c(167, 192, 173, 174, 172, 167, 171, 185, 163, 170)
```

Ce faisant, nous avons créé un objet nommé `tailles` et comprenant l'ensemble de nos données, que nous pouvons afficher en saisissant simplement son nom :

```
R> tailles
```

```
[1] 167 192 173 174 172 167 171 185 163 170
```

Que se passe-t-il s'il on crée un vecteur plus grand ?

---

2. `c` est l'abréviation de *combine*. Le nom de cette fonction est très court car on l'utilise très souvent.

```
R> c(144, 168, 179, 175, 182, 188, 167, 152, 163, 145, 176, 155, 156, 164, 167, 1  
55, 157, 185, 155, 169, 124, 178, 182, 195, 151, 185, 159, 156, 184, 172)
```

```
[1] 144 168 179 175 182 188 167 152 163 145 176 155 156 164  
[15] 167 155 157 185 155 169 124 178 182 195 151 185 159 156  
[29] 184 172
```

On a bien notre suite de trente tailles, mais on peut remarquer la présence de nombres entre crochets au début de chaque ligne ([1], [15] et [29]). En fait ces nombres entre crochets indiquent la position du premier élément de la ligne dans notre vecteur. Ainsi, le 167 en début de deuxième ligne est le 15<sup>e</sup> élément du vecteur, tandis que le 184 de la troisième ligne est à la 29<sup>e</sup> position.

On en déduira d’ailleurs que lorsque l’on fait :

```
R> 2
```

```
[1] 2
```

R considère en fait le nombre 2 comme un vecteur à un seul élément.

On peut appliquer des opérations arithmétiques simples directement sur des vecteurs :

```
R> tailles <- c(167, 192, 173, 174, 172, 167, 171, 185, 163, 170)  
tailles + 20
```

```
[1] 187 212 193 194 192 187 191 205 183 190
```

```
R> tailles / 100
```

```
[1] 1.67 1.92 1.73 1.74 1.72 1.67 1.71 1.85 1.63 1.70
```

```
R> tailles^2
```

```
[1] 27889 36864 29929 30276 29584 27889 29241 34225 26569  
[10] 28900
```

On peut aussi combiner des vecteurs entre eux. L’exemple suivant calcule l’indice de masse corporelle à partir de la taille et du poids :

```
R> tailles <- c(167, 192, 173, 174, 172, 167, 171, 185, 163, 170)
  poids <- c(86, 74, 83, 50, 78, 66, 66, 51, 50, 55)
  tailles.m <- tailles / 100
  imc <- poids / (tailles.m^2)
  imc
```

```
[1] 30.84 20.07 27.73 16.51 26.37 23.67 22.57 14.90 18.82
[10] 19.03
```

### IMPORTANT

Quand on fait des opérations sur les vecteurs, il faut veiller à soit utiliser un vecteur et un chiffre (dans des opérations du type `v * 2` ou `v + 10`), soit à utiliser des vecteurs de même longueur (dans des opérations du type `u + v`).

Si on utilise des vecteurs de longueur différentes, on peut avoir quelques surprises. Quand R effectue une opération avec deux vecteurs de longueurs différentes, il recopie le vecteur le plus court de manière à lui donner la même taille que le plus long, ce qui s'appelle la règle de recyclage (*recycling rule*). Ainsi, `c(1,2) + c(4,5,6,7,8)` vaudra l'équivalent de `c(1,2,1,2,1) + c(4,5,6,7,8)`.

On a vu jusque-là des vecteurs composés de nombres, mais on peut tout à fait créer des vecteurs composés de chaînes de caractères, représentant par exemple les réponses à une question ouverte ou fermée :

```
R> reponse <- c("Bac+2", "Bac", "CAP", "Bac", "Bac", "CAP", "BEP")
  reponse
```

```
[1] "Bac+2" "Bac"   "CAP"   "Bac"   "Bac"   "CAP"   "BEP"
```

Enfin, notons que l'on peut accéder à un élément particulier du vecteur en faisant suivre le nom du vecteur de crochets contenant le numéro de l'élément désiré. Par exemple :

```
R> reponse <- c("Bac+2", "Bac", "CAP", "Bac", "Bac", "CAP", "BEP")
  reponse[2]
```

```
[1] "Bac"
```

Cette opération s'appelle l'indexation d'un vecteur. Il s'agit ici de sa forme la plus simple, mais il en existe d'autres beaucoup plus complexes. L'indexation des vecteurs et des tableaux dans R est l'un des éléments particulièrement souples et puissants du langage (mais aussi l'un des plus délicats à comprendre et à

maîtriser). Nous en reparlerons dans le chapitre Vecteurs, indexation et assignation, page 67.

#### NOTE

Sous **RStudio**, vous avez du remarquer que ce dernier effectue une coloration syntaxique. Lorsque vous tapez une commande, les valeurs numériques sont affichées dans une certaine couleur, les valeurs textuelles dans une autre et les noms des fonctions dans une troisième. De plus, si vous tapez une parenthèse ouvrante, **RStudio** va créer automatiquement après le curseur la parenthèse fermante correspondante (de même avec les guillements ou les crochets). Si vous placez le curseur juste après une parenthèse fermante, la parenthèse ouvrante correspondante sera surlignée, ce qui sera bien pratique lors de la rédaction de commandes complexes.

## Des fonctions

Nous savons désormais faire des opérations simples sur des nombres et des vecteurs, stocker ces données et résultats dans des objets pour les réutiliser par la suite.

Pour aller un peu plus loin nous allons aborder, après les objets, l’autre concept de base de **R**, à savoir les fonctions. Une fonction se caractérise de la manière suivante :

- elle a un nom ;
- elle accepte des arguments (qui peuvent avoir un nom ou pas) ;
- elle retourne un résultat et peut effectuer une action comme dessiner un graphique ou lire un fichier.

En fait rien de bien nouveau puisque nous avons déjà utilisé plusieurs fonctions jusqu’ici, dont la plus visible est la fonction `c`. Dans la ligne suivante :

```
R> reponse <- c("Bac+2", "Bac", "CAP", "Bac", "Bac", "CAP", "BEP")
```

on fait appel à la fonction nommée `c`, on lui passe en arguments (entre parenthèses et séparées par des virgules) une série de chaînes de caractères et elle retourne comme résultat un vecteur de chaînes de caractères, que nous stockons dans l’objet `reponse`.

Prenons tout de suite d’autres exemples de fonctions courantes :

```
R> tailles <- c(167, 192, 173, 174, 172, 167, 171, 185, 163, 170)
length(tailles)
```

```
[1] 10
```

```
R> mean(tailles)
```

```
[1] 173.4
```

```
R> var(tailles)
```

```
[1] 76.71
```

Ici, la fonction `length` nous renvoie le nombre d'éléments du vecteur, la fonction `mean` nous donne la moyenne des éléments du vecteur et fonction `var` sa variance.

## Arguments

Les arguments de la fonction lui sont indiqués entre parenthèses, juste après son nom. En général les premiers arguments passés à la fonction sont des données servant au calcul et les suivants des paramètres influant sur ce calcul. Ceux-ci sont en général transmis sous la forme d'argument nommés.

Reprendons l'exemple des tailles précédent :

```
R> tailles <- c(167, 192, 173, 174, 172, 167, 171, 185, 163, 170)
```

Imaginons que le deuxième enquêté n'ait pas voulu nous répondre. Nous avons alors dans notre vecteur une valeur manquante. Celle-ci est symbolisée dans R par le code `NA` :

```
R> tailles <- c(167, NA, 173, 174, 172, 167, 171, 185, 163, 170)
```

Recalculons notre taille moyenne :

```
R> mean(tailles)
```

```
[1] NA
```

Et oui, par défaut, R renvoie `NA` pour un grand nombre de calculs (dont la moyenne) lorsque les données comportent une valeur manquante. On peut cependant modifier ce comportement en fournissant un paramètre supplémentaire à la fonction `mean`, nommé `na.rm` :

```
R> mean(tailles, na.rm = TRUE)
```

```
[1] 171.3
```

Positionner le paramètre `na.rm` à `TRUE` (vrai) indique à la fonction `mean` de ne pas tenir compte des valeurs manquantes dans le calcul.

Lorsqu'on passe un argument à une fonction de cette manière, c'est-à-dire sous la forme `nom=valeur`, on parle d'argument nommé.

#### IMPORTANT

`NA` signifie *not available*. Cette valeur particulière peut être utilisée pour indiquer une valeur manquante pour tout type de liste (nombres, textes, valeurs logique, etc.).

## Quelques fonctions utiles

Récapitulons la liste des fonctions que nous avons déjà rencontrées :

Fonction	Description
<code>c</code>	construit un vecteur à partir d'une série de valeurs
<code>length</code>	nombre d'éléments d'un vecteur
<code>mean</code>	moyenne d'un vecteur de type numérique
<code>var</code>	variance d'un vecteur de type numérique
<code>+ , - , * , /</code>	opérateurs mathématiques de base
<code>^</code>	passage à la puissance

On peut rajouter les fonctions de base suivantes :

Fonction	Description
<code>min</code>	valeur minimale d'un vecteur numérique
<code>max</code>	valeur maximale d'un vecteur numérique
<code>sd</code>	écart-type d'un vecteur numérique
<code>:</code>	génère une séquence de nombres. <code>1:4</code> équivaut à <code>c(1,2,3,4)</code>

## Aide sur une fonction

Il est très fréquent de ne plus se rappeler quels sont les paramètres d'une fonction ou le type de résultat qu'elle retourne. Dans ce cas on peut très facilement accéder à l'aide décrivant une fonction particulière avec `?` ou `help`. Ainsi, pour obtenir de l'aide sur la fonction `mean`, on saisira l'une des deux entrées équivalentes suivantes :

```
R> ?mean
help("mean")
```

### NOTE

L'utilisation du raccourci `?` ne fonctionne pas pour certains opérateurs comme `*`. Dans ce cas on pourra utiliser `? '*'` ou bien simplement `help("*")`.

Sous **RStudio**, le fichier d'aide associé apparaîtra dans le quadrant inférieur droit sous l'onglet *Help*.

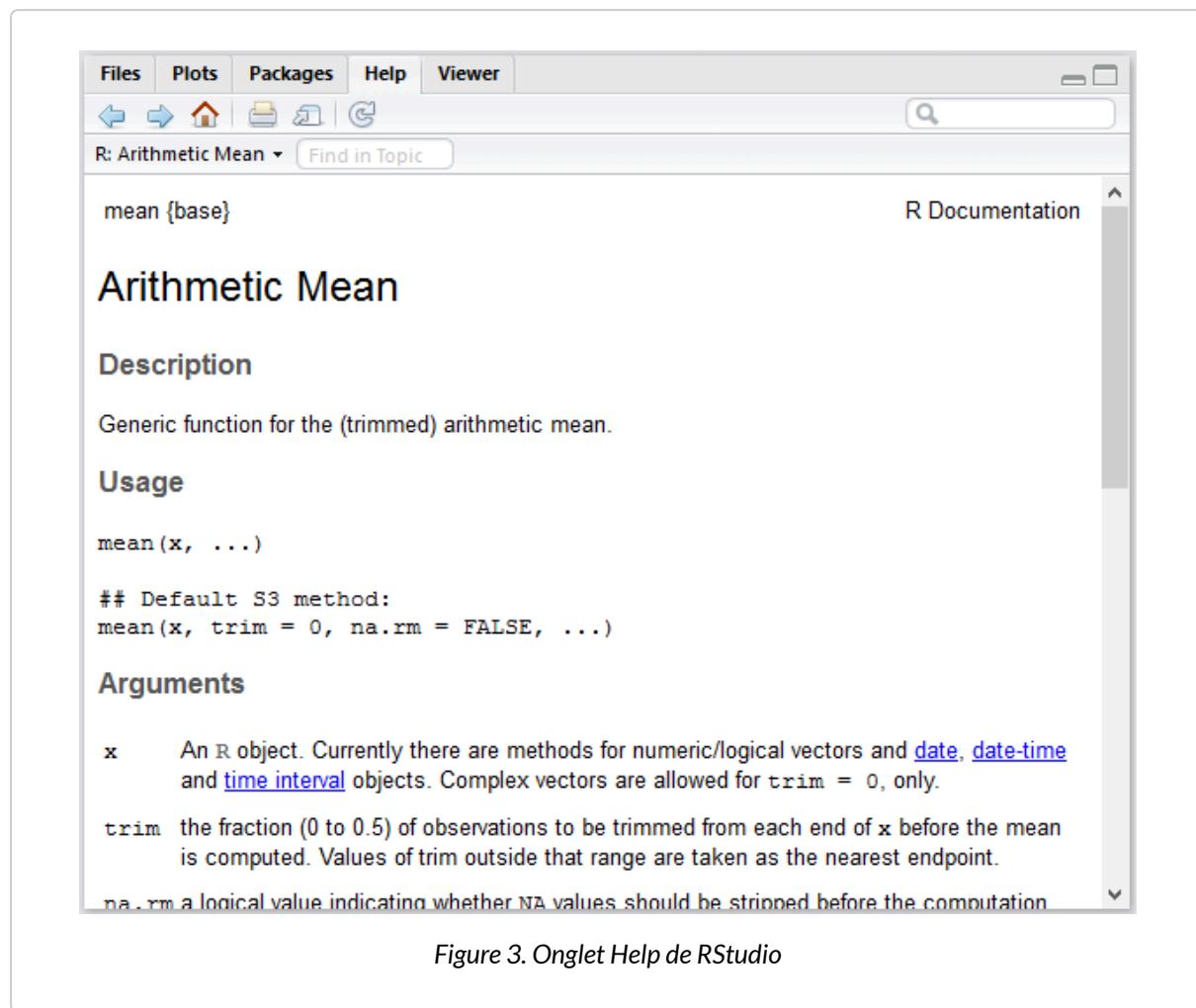


Figure 3. Onglet Help de RStudio

Cette page décrit (en anglais) la fonction, ses arguments, son résultat, le tout accompagné de diverses notes, références et exemples. Ces pages d'aide contiennent à peu près tout ce que vous pourrez chercher à savoir, mais elles ne sont pas toujours d'une lecture aisée.

Un autre cas très courant dans R est de ne pas se souvenir ou de ne pas connaître le nom de la fonction effectuant une tâche donnée. Dans ce cas on se reportera aux différentes manières de trouver de l'aide décrites dans le chapitre Où trouver de l'aide ?, page 163.

## Interprétation des arguments

Prenons l'exemple de la fonction `format` dont la version de base permet de mettre en forme un nombre. Affichons le fichier d'aide associé.

```
R> ?format
```

La section *Usage* présente les arguments de cette fonction et leur valeur par défaut :

```
format(x, trim = FALSE, digits = NULL, nsmall = 0L,
       justify = c("left", "right", "centre", "none"),
       width = NULL, na.encode = TRUE, scientific = NA,
       big.mark = "", big.interval = 3L,
       small.mark = "", small.interval = 5L,
       decimal.mark = ".", zero.print = NULL,
       drop0trailing = FALSE, ...)
```

Regardons ce que cette fonction peut faire. Passons-lui un vecteur avec deux nombres :

```
R> format(c(12.3, 5678))
```

```
[1] " 12.3" "5678.0"
```

Elle renvoie un vecteur de chaînes de caractères. Le nombre de décimales a été harmonisé et des espaces ont été ajoutés au début du premier nombre afin que l'ensemble des valeurs soient alignées vers la droite.

L'argument `trim` permet de supprimer les espaces ajoutés en début de chaîne.

```
R> format(c(12.3, 5678), TRUE)
```

```
[1] "12.3" "5678.0"
```

Dans le cas présent, nous avons saisi les arguments de la fonction sans les nommer. Dès lors, R considère l'ordre dans lesquels nous avons saisi les arguments, ordre qui correspond à celui du fichier d'aide. Il a dès lors considéré que `c(12.3, 5678)` correspond à la valeur attribuée à `x` et que `TRUE` est la valeur attribuée à `trim`.

L'argument `nsmall` permet d'indiquer le nombre minimum de décimales que l'on souhaite afficher. Il est en quatrième position. Dès lors, pour pouvoir le renseigner avec des arguments non nommés, il faut fournir également une valeur pour le troisième argument `digits`.

```
R> format(c(12.3, 5678), TRUE, NULL, 2)
```

```
[1] "12.30" "5678.00"
```

Ce n'est pas forcément ce qu'il y a de plus pratique. D'où l'intérêt des arguments nommés. En précisant `nsmall =` dans l'appel de la fonction, on pourra indiquer que l'on souhaite modifier spécifiquement

cet argument. Lorsque l'on utilise des arguments non nommés, l'ordre n'importe plus puisque **R** sera en capacité de reconnaître ses petits.

```
R> format(nsmall = 2, x = c(12.3, 5678))
```

```
[1] " 12.30" "5678.00"
```

À l'usage, on aura le plus souvent recours à une combinaison d'arguments non nommés et d'arguments nommés. On indiquera les premiers arguments (qui correspondent en général aux données de départ) sans les nommer et on précisera les options souhaitées avec des arguments nommés. Par exemple, pour un affichage à la française :

```
R> format(c(12.3, 5678), decimal.mark = ",", big.mark = " ")
```

```
[1] " 12,3" "5 678,0"
```

Lorsque l'on regarde la section *Usage* du fichier d'aide, il apparaît que certains arguments, suivi par le symbole `=`, ont une valeur par défaut. Il n'est donc pas nécessaire de les inclure dans l'appel de la fonction, auquel cas la valeur pas défaut sera prise en compte. Par contre, d'autres arguments, ici `x`, n'ont pas de valeur par défaut et il est donc nécessaire de fournir systématiquement une valeur.

```
R> format(decimal.mark = ",")
```

```
Error in format.default(decimal.mark = ","): l'argument "x" est manquant, avec aucune valeur par défaut
```

Enfin, pour certaines fonctions, on verra parfois apparaître le symbole `...`. Ce dernier correspond à un nombre indéterminé d'arguments. Il peut s'agir, comme dans le cas de `format` d'arguments additionnels qui seront utilisés dans certains cas de figure, ou bien d'arguments qui seront transmis à une fonction secondaire appelée par la fonction principale, ou encore, comme pour le cas de la fonction `c`, de la possibilité de saisir un nombre indéfini de données sources.

## Autocomplétion

**RStudio** fournit un outil bien pratique appelé autocomplete<sup>3</sup>. Saisissez les premières lettres d'une fonction, par exemple `me` puis appuyez sur la touche **Tabulation**. **RStudio** affichera la liste des fonctions dont le nom commence par `me` ainsi qu'un court descriptif de chacune. Un appui sur la touche **Entrée** provoquera la saisie du nom complet de la fonction choisie.

---

3. En bon français, il faudrait dire *complètement automatique*.

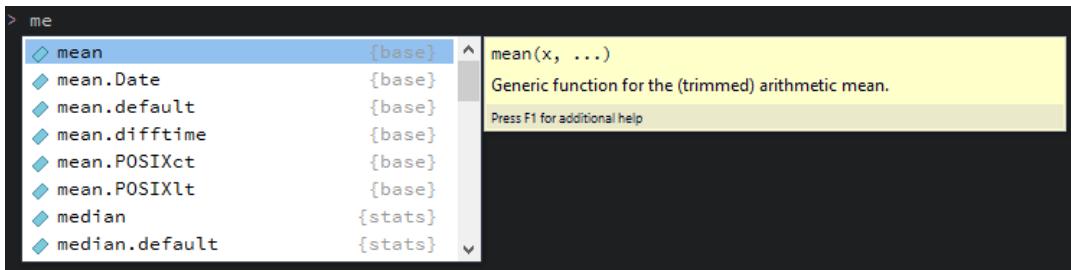


Figure 4. Auto-complétion sous RStudio

À l'intérieur des parenthèses d'une fonction, vous pouvez utiliser l'autocomplétion pour retrouver un argument de cette fonction.

Vous pouvez également utiliser l'autocomplétion pour retrouver le nom d'un objet que vous avez précédemment créé.

Pour plus de détails, voir la documentation officielle de RStudio (<https://support.rstudio.com/hc/en-us/articles/205273297-Code-Completion>).



# Premier travail avec des données

Regrouper les commandes dans des scripts . . . . .	33
Ajouter des commentaires . . . . .	36
Tableaux de données . . . . .	37
Inspection visuelle des données . . . . .	39
Structure du tableau . . . . .	40
Accéder aux variables . . . . .	41
La fonction str . . . . .	42
Quelques calculs simples . . . . .	49
Nos premiers graphiques . . . . .	52
Et ensuite ? . . . . .	53

## NOTE

Ce chapitre est inspiré de la section *Premier travail avec les données* du support de cours [Introduction à R](#) réalisé par Julien Barnier.

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #01 (premier contact avec R & RStudio) sur [YouTube](#).

## Regrouper les commandes dans des scripts

Jusqu'à maintenant nous avons utilisé uniquement la console pour communiquer avec R via l'invite de commandes. Le principal problème de ce mode d'interaction est qu'une fois qu'une commande est tapée, elle est pour ainsi dire « perdue », c'est-à-dire qu'on doit la saisir à nouveau si on veut l'exécuter une

seconde fois. L’utilisation de la console est donc restreinte aux petites commandes « jetables », le plus souvent utilisées comme test.

La plupart du temps, les commandes seront stockées dans un fichier à part, que l’on pourra facilement ouvrir, éditer et exécuter en tout ou partie si besoin. On appelle en général ce type de fichier un **script**.

Pour comprendre comment cela fonctionne, dans **RStudio** cliquez sur l’icône en haut à gauche représentant un fichier avec un signe plus vert, puis choisissez **R script**.

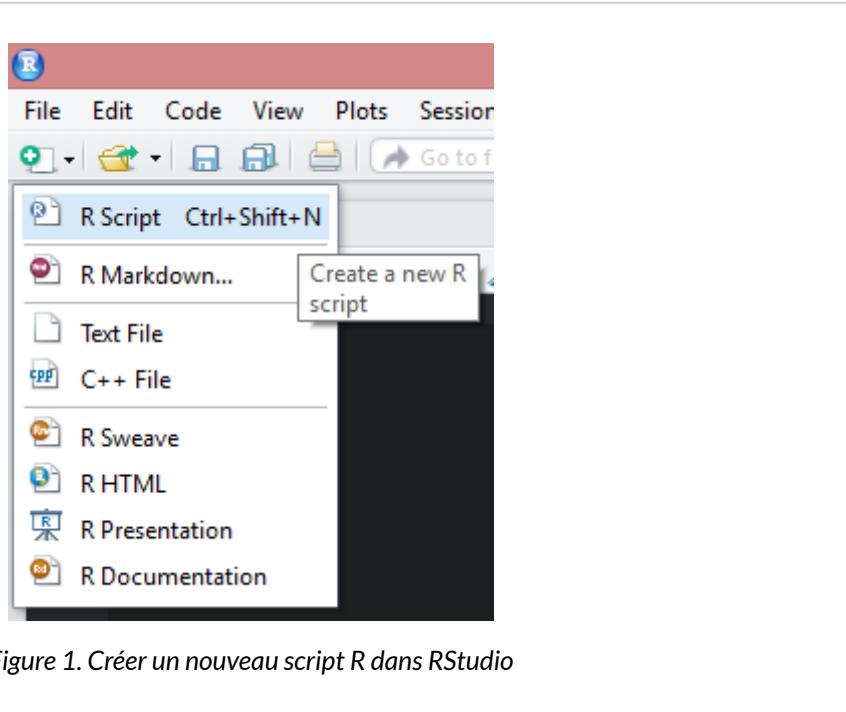


Figure 1. Créez un nouveau script R dans RStudio

Un nouvel onglet apparaît dans le quadrant supérieur gauche.

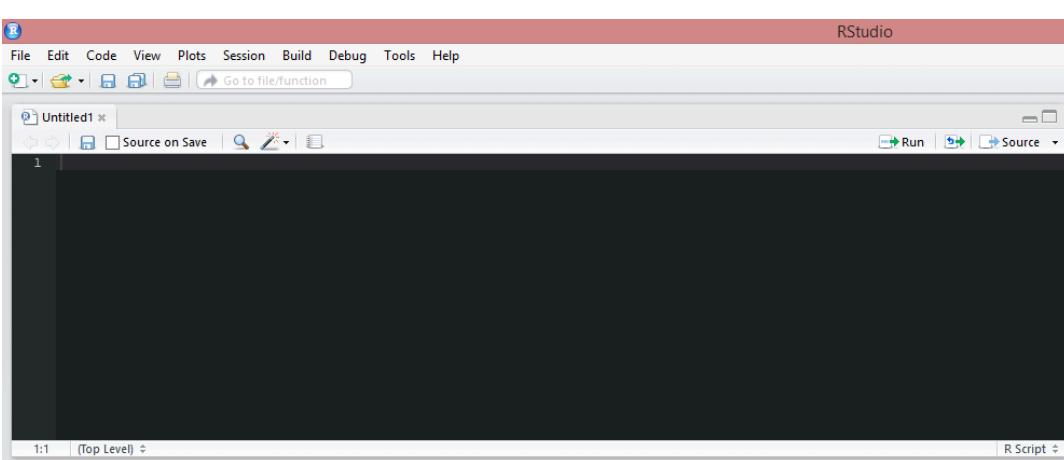


Figure 2. Onglet d'un script R dans RStudio

Nous pouvons désormais y saisir des commandes. Par exemple, tapez sur la première ligne la commande suivante : `2 + 2`. Ensuite, cliquez sur l'icône *Run* (en haut à droite de l'onglet du script) ou bien pressez simultanément les touches **CTRL** et **Entrée**<sup>1</sup>.

Les lignes suivantes ont dû faire leur apparition dans la console :

```
R> 2 + 2
```

```
[1] 4
```

Voici donc comment soumettre rapidement à R les commandes saisies dans votre fichier. Vous pouvez désormais l'enregistrer, l'ouvrir plus tard, et en exécuter tout ou partie. À noter que vous avez plusieurs possibilités pour soumettre des commandes à R :

- vous pouvez exécuter la ligne sur laquelle se trouve votre curseur en cliquant sur *Run* ou en pressant simultanément les touches **CTRL** et **Entrée** ;
- vous pouvez sélectionner plusieurs lignes contenant des commandes et les exécuter toutes en une seule fois exactement de la même manière ;
- vous pouvez exécuter d'un coup l'intégralité de votre fichier en cliquant sur l'icône *Source*.

La plupart du travail sous R consistera donc à éditer un ou plusieurs fichiers de commandes et à envoyer régulièrement les commandes saisies à R en utilisant les raccourcis clavier *ad hoc*.

Pour plus d'information sur l'utilisation des scripts R dans RStudio, voir (en anglais) : <https://support.rstudio.com/hc/en-us/articles/200484448-Editing-and-Executing-Code>.

---

1. Sous Mac OS X, on utilise les touches **Pomme** et **Entrée**.

#### NOTE

Quand vous enregistrez un script sous RStudio, il est possible qu'il vous demande de choisir un type d'encodage des caractères (*Choose Encoding*). Si tel est le cas, utilisez de préférence **UTF-8**.

## Ajouter des commentaires

Un commentaire est une ligne ou une portion de ligne qui sera ignorée par R. Ceci signifie qu'on peut y écrire ce qu'on veut et qu'on va les utiliser pour ajouter tout un tas de commentaires à notre code permettant de décrire les différentes étapes du travail, les choses à se rappeler, les questions en suspens, etc.

Un commentaire sous R commence par un ou plusieurs symboles `#` (qui s'obtient avec les touches **Alt Gr** et **3** sur les claviers de type PC). Tout ce qui suit ce symbole jusqu'à la fin de la ligne est considéré comme un commentaire. On peut créer une ligne entière de commentaire en la faisant débuter par `##`. Par exemple :

```
R> ## Tableau croisé de la CSP par le nombre de livres lus.  
## Attention au nombre de non réponses !
```

On peut aussi créer des commentaires pour une ligne en cours :

```
R> x <- 2 # On met 2 dans x, parce qu'il le vaut bien
```

#### IMPORTANT

Dans tous les cas, il est très important de documenter ses fichiers R au fur et à mesure, faute de quoi on risque de ne plus y comprendre grand chose si on les reprend ne serait-ce que quelques semaines plus tard.

Avec RStudio, vous pouvez également utiliser les commentaires pour créer des sections au sein de votre script et naviguer plus rapidement. Il suffit de faire suivre une ligne de commentaires d'au moins 4 signes moins (`----`). Par exemple, si vous saisissez ceci dans votre script :

```
R> ## Créer les objets ----

x <- 2
y <- 5

## Calculs ----

x + y
```

Vous verrez apparaître en bas à gauche de la fenêtre du script un symbole dièse orange. Si vous cliquez dessus, un menu de navigation s'affichera vous permettant de vous déplacez rapidement au sein de votre script. Pour plus d'information, voir la documentation de RStudio (en anglais) : <https://support.rstudio.com/hc/en-us/articles/200484568-Code-Folding-and-Sections>.

Les sections peuvent également être facilement créées avec le raccourci clavier **CTRL + SHIFT + R**.

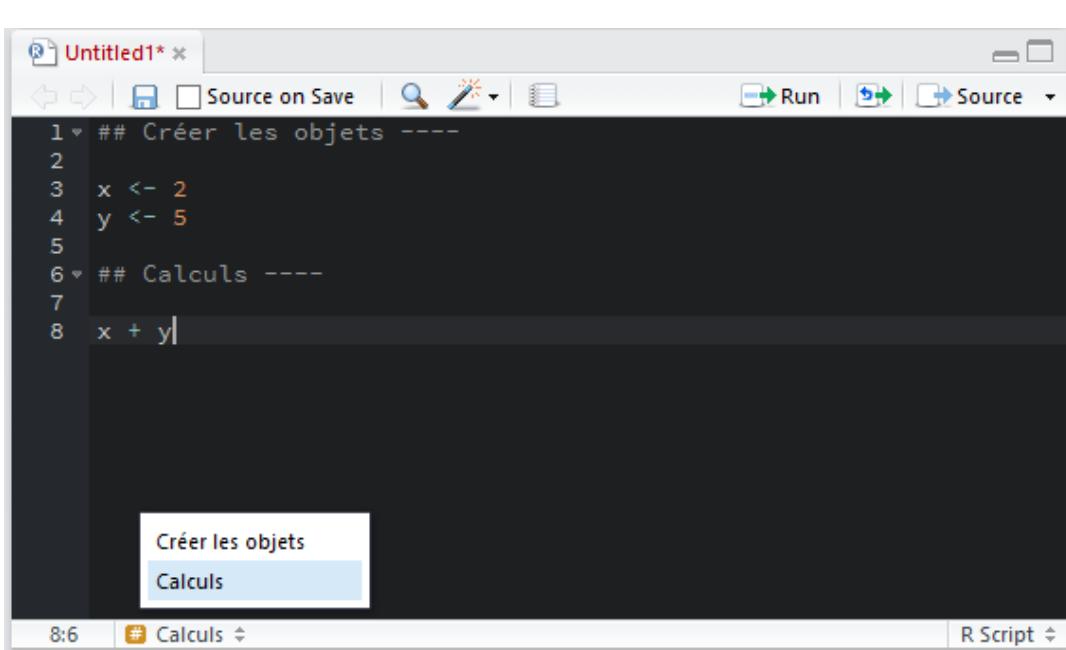


Figure 3. Navigation rapide dans les scripts sous RStudio

Note : on remarquera au passage que le titre de l'onglet est affiché en rouge et suivi d'une astérisque ( \* ), nous indiquant ainsi qu'il y a des modifications non enregistrées dans notre fichier.

## Tableaux de données

Dans cette partie nous allons utiliser un jeu de données inclus dans l'extension **questionr**. L'installation

d'extension est décrite dans le chapitre Extensions, page 55.

Le jeu de données en question est un extrait de l'enquête *Histoire de vie* réalisée par l'INSEE en 2003. Il contient 2000 individus et 20 variables. Pour pouvoir utiliser ces données, il faut d'abord charger l'extension `questionr` (après l'avoir installée, bien entendu). Le chargement d'une extension en mémoire se fait à l'aide de la fonction `library`. Sous **RStudio**, vous pouvez également charger une extension en allant dans l'onglet *Packages* du quadrant inférieur droit qui liste l'ensemble des packages disponibles et en cliquant la case à cocher située à gauche du nom du package désiré.

```
R> library(questionr)
```

Puis nous allons indiquer à **R** que nous souhaitons accéder au jeu de données `hdv2003` à l'aide de la fonction `data` :

```
R> data(hdv2003)
```

Bien. Et maintenant, elles sont où mes données ? Et bien elles se trouvent dans un objet nommé `hdv2003` désormais chargé en mémoire et accessible directement. D'ailleurs, cet objet est maintenant visible dans l'onglet *Environment* du quadrant supérieur droit.

Essayons de taper son nom à l'invite de commande :

```
R> hdv2003
```

Le résultat (non reproduit ici) ne ressemble pas forcément à grand-chose... Il faut se rappeler que par défaut, lorsqu'on lui fournit seulement un nom d'objet, **R** essaye de l'afficher de la manière la meilleure (ou la moins pire) possible. La réponse à la commande `hdv2003` n'est donc rien moins que l'affichage des données brutes contenues dans cet objet.

Ce qui signifie donc que l'intégralité de notre jeu de données est inclus dans l'objet nommé `hdv2003` ! En effet, dans **R**, un objet peut très bien contenir un simple nombre, un vecteur ou bien le résultat d'une enquête tout entier. Dans ce cas, les objets sont appelés des *data frames*, ou tableaux de données. Ils peuvent être manipulés comme tout autre objet. Par exemple :

```
R> d <- hdv2003
```

va entraîner la copie de l'ensemble de nos données dans un nouvel objet nommé `d`, ce qui peut paraître parfaitement inutile mais a en fait l'avantage de fournir un objet avec un nom beaucoup plus court, ce qui diminuera la quantité de texte à saisir par la suite.

## Résumons

Comme nous avons désormais décidé de saisir nos commandes dans un script et non plus directement dans la console, les premières lignes de notre fichier de travail sur les données de l'enquête *Histoire de vie* pourraient donc ressembler à ceci :

```
R> ## Chargement des extensions nécessaires ----
library(questionr)
## Jeu de données hdv2003 ----
data(hdv2003)
d <- hdv2003
```

## Inspection visuelle des données

La particularité de R par rapport à d'autres logiciels comme **Modalisa** ou **SPSS** est de ne pas proposer, par défaut, de vue des données sous forme de tableau. Ceci peut parfois être un peu déstabilisant dans les premiers temps d'utilisation, même si l'on perd vite l'habitude et qu'on finit par se rendre compte que « voir » les données n'est pas forcément un gage de productivité ou de rigueur dans le traitement.

Néanmoins, R propose une interface permettant de visualiser le contenu d'un tableau de données à l'aide de la fonction `View` :

```
R> View(d)
```

Sous **RStudio**, on peut aussi afficher la *visionneusee (viewer)* en cliquant sur la petite icône en forme de tableau située à droite de la ligne d'un tableau de données dans l'onglet *Environment* du quadrant supérieur droit (cf. figure ci-après).

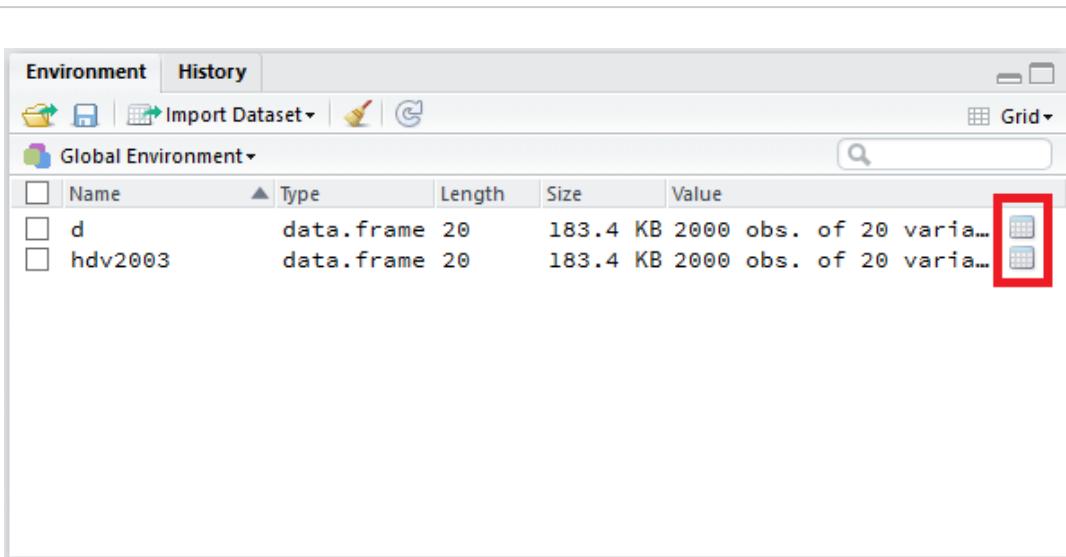


Figure 4. Icône pour afficher une vue du contenu d'un tableau

Dans tous les cas, **RStudio** lancera le viewer dans un onglet dédié dans le quadrant supérieur gauche. Le visualiseur de **RStudio** est plus avancé que celui-de base fournit par R. Il est possible de trier les données

selon une variable en cliquant sur le nom de cette dernière. Il y a également un champs de recherche et un bouton *Filter* donnant accès à des options de filtrage avancées.

	id	age	sexe	nivetud	poids	occup	qualif	freres.soeurs	ciso
1	1	28	Femme	Enseignement supérieur y compris technique supérieur	2634.3982	Exerce une profession	Employe	8	Oui
2	2	23	Femme	NA	9738.3958	Etudiant, elevé	NA	2	Oui
3	3	59	Homme	Dernière année d'études primaires	3994.1025	Exerce une profession	Technicien	2	Non
4	4	34	Homme	Enseignement supérieur y compris technique supérieur	5731.6615	Exerce une profession	Technicien	1	Non
5	5	71	Femme	Dernière année d'études primaires	4329.0940	Retraite	Employe	0	Oui
6	6	35	Femme	Enseignement technique ou professionnel court	8674.6994	Exerce une profession	Employe	5	Non
7	7	60	Femme	Dernière année d'études primaires	6165.8035	Au foyer	Ouvrier qualifié	1	Oui
8	8	47	Homme	Enseignement technique ou professionnel court	12891.6408	Exerce une profession	Ouvrier qualifié	5	Non
9	9	20	Femme	NA	7808.8721	Etudiant, elevé	NA	4	Oui
10	10	28	Homme	Enseignement technique ou professionnel long	2277.1605	Exerce une profession	Autre	2	Non
11	11	65	Femme	Enseignement supérieur y compris technique supérieur	704.3227	Retraite	Employe	3	Oui
12	12	47	Homme	2ème cycle	6697.8682	Exerce une profession	Ouvrier qualifié	4	Oui
13	13	63	Femme	Dernière année d'études primaires	7118.4659	Retraite	Employe	1	Oui
14	14	67	Femme	Enseignement technique ou professionnel court	586.7714	Exerce une profession	NA	5	Oui
15	15	76	Femme	A arrête ses études, avant la dernière année d'études ...	11042.0774	Retraite	NA	2	Oui
16	16	49	Femme	Enseignement technique ou professionnel court	9958.2287	Exerce une profession	Employe	3	Non
17	17	62	Homme	Enseignement supérieur y compris technique supérieur	4836.1393	Retraite	Cadre	4	Non

Figure 5. La visionneuse de données de RStudio

## Structure du tableau

Avant de travailler sur les données, nous allons essayer de comprendre comment elles sont structurées. Lors de l'import de données depuis un autre logiciel (que nous aborderons dans un autre chapitre, page 145), il s'agira souvent de vérifier que l'importation s'est bien déroulée.

Nous avons déjà vu qu'un tableau de données est organisé en lignes et en colonnes, les lignes correspondant aux observations et les colonnes aux variables. Les fonctions `nrow`, `ncol` et `dim` donnent respectivement le nombre de lignes, le nombre de colonnes et les dimensions de notre tableau. Nous pouvons donc d'ores et déjà vérifier que nous avons bien 2000 lignes et 20 colonnes :

```
R> nrow(d)
```

```
[1] 2000
```

```
R> ncol(d)
```

```
[1] 20
```

```
R> dim(d)
```

```
[1] 2000 20
```

La fonction `names` donne les noms des colonnes de notre tableau, c'est-à-dire les noms des variables :

```
R> names(d)
```

```
[1] "id"           "age"          "sexe"
[4] "nivetud"      "poids"        "occup"
[7] "qualif"       "freres.soeurs" "clos"
[10] "relig"        "trav.imp"     "trav.satisf"
[13] "hard.rock"    "lecture.bd"   "peche.chasse"
[16] "cuisine"      "bricol"       "cinema"
[19] "sport"        "heures.tv"
```

## Accéder aux variables

`d` représente donc l'ensemble de notre tableau de données. Nous avons vu que si l'on saisit simplement `d` à l'invite de commandes, on obtient un affichage du tableau en question. Mais comment accéder aux variables, c'est à dire aux colonnes de notre tableau ?

La réponse est simple : on utilise le nom de l'objet, suivi de l'opérateur `$`, suivi du nom de la variable, comme ceci :

```
R> d$sexe
```

Au regard du résultat (non reproduit ici), on constate alors que R a bien accédé au contenu de notre variable `sexe` du tableau `d` et a affiché son contenu, c'est-à-dire l'ensemble des valeurs prises par la variable.

Les fonctions `head` et `tail` permettent d'afficher seulement les premières (respectivement les dernières) valeurs prises par la variable. On peut leur passer en argument le nombre d'éléments à afficher :

```
R> head(d$nivetud)
```

```
[1] Enseignement superieur y compris technique superieur
[2] <NA>
[3] Derniere annee d'etudes primaires
[4] Enseignement superieur y compris technique superieur
```

```
[5] Derniere annee d'etudes primaires  
[6] Enseignement technique ou professionnel court  
8 Levels: N'a jamais fait d'etudes ...
```

```
R> tail(d$age, 10)
```

```
[1] 52 42 50 41 46 45 46 24 24 66
```

À noter que ces fonctions marchent aussi pour afficher les lignes du tableau `d` :

```
R> head(d, 2)
```

## La fonction str

La fonction `str` est plus complète que `names`. Elle liste les différentes variables, indique leur type et donne le cas échéant des informations supplémentaires ainsi qu'un échantillon des premières valeurs prises par cette variable :

```
R> str(d)
```

```
'data.frame': 2000 obs. of 20 variables:  
 $ id          : int  1 2 3 4 5 6 7 8 9 10 ...  
 $ age         : int  28 23 59 34 71 35 60 47 20 28 ...  
 $ sexe        : Factor w/ 2 levels "Homme","Femme": 2 2 1 1 2 2 2 1 2 1 ...  
 $ nivetud     : Factor w/ 8 levels "N'a jamais fait d'etudes",...: 8 NA 3 8 3  
 6 3 6 NA 7 ...  
 $ poids        : num  2634 9738 3994 5732 4329 ...  
 $ occup        : Factor w/ 7 levels "Exerce une profession",...: 1 3 1 1 4 1 6  
 1 3 1 ...  
 $ qualif       : Factor w/ 7 levels "Ouvrier specialise",...: 6 NA 3 3 6 6 2 2 N  
 A 7 ...  
 $ freres.soeurs: int  8 2 2 1 0 5 1 5 4 2 ...  
 $ cuso         : Factor w/ 3 levels "Oui","Non","Ne sait pas": 1 1 2 2 1 2 1 2  
 1 2 ...  
 $ relig         : Factor w/ 6 levels "Pratiquant regulier",...: 4 4 4 3 1 4 3 4  
 3 2 ...  
 $ trav.imp      : Factor w/ 4 levels "Le plus important",...: 4 NA 2 3 NA 1 NA 4  
 NA 3 ...  
 $ trav.satisf  : Factor w/ 3 levels "Satisfaction",...: 2 NA 3 1 NA 3 NA 2 NA 1  
 ...
```

```
$ hard.rock      : Factor w/ 2 levels "Non","Oui": 1 1 1 1 1 1 1 1 1 1 ...
$ lecture.bd    : Factor w/ 2 levels "Non","Oui": 1 1 1 1 1 1 1 1 1 1 ...
$ peche.chasse  : Factor w/ 2 levels "Non","Oui": 1 1 1 1 1 1 2 2 1 1 ...
$ cuisine       : Factor w/ 2 levels "Non","Oui": 2 1 1 2 1 1 2 2 1 1 ...
$ bricol         : Factor w/ 2 levels "Non","Oui": 1 1 1 2 1 1 1 2 1 1 ...
$ cinema         : Factor w/ 2 levels "Non","Oui": 1 2 1 2 1 2 1 1 2 2 ...
$ sport          : Factor w/ 2 levels "Non","Oui": 1 2 2 2 1 2 1 1 1 2 ...
$ heures.tv     : num  0 1 0 2 3 2 2.9 1 2 2 ...
```

La première ligne nous informe qu'il s'agit bien d'un tableau de données avec 2000 observations et 20 variables. Vient ensuite la liste des variables. La première se nomme `id` et est de type entier (`int`). La seconde se nomme `age` et est de type numérique. La troisième se nomme `sexe`, il s'agit d'un facteur (`factor`).

Un facteur est une variable pouvant prendre un nombre limité de modalités (`levels`). Ici notre variable a deux modalités possibles : « Homme » et « Femme ». Ce type de variable est décrit plus en détail dans le chapitre sur la [manipulation de données](#).

**IMPORTANT**

La fonction `str` est essentielle à connaître et peut s'appliquer à n'importe quel type d'objet. C'est un excellent moyen de connaître en détail la structure d'un objet. Cependant, les résultats peuvent être parfois trop détaillés et on lui priviliera dans certains cas la fonction `describe` que l'on abordera dans les prochains chapitres, cependant moins générique puisque ne s'appliquant qu'à des tableaux de données et à des vecteurs, tandis que `str` peut s'appliquer à absolument **tout** objet, y compris des fonctions.

```
R> describe(d)
```

d

20 Variables 2000 Observations

-----

id

n	missing	distinct	Info	Mean	Gmd
2000	0	2000	1	1000	667
.05	.10	.25	.50	.75	.90
101.0	200.9	500.8	1000.5	1500.2	1800.1
.95					
1900.0					

lowest : 1 2 3 4 5, highest: 1996 1997 1998 1999 2000

-----

age

n	missing	distinct	Info	Mean	Gmd
2000	0	78	1	48.16	19.4
.05	.10	.25	.50	.75	.90
22	26	35	48	60	72
.95					
77					

lowest : 18 19 20 21 22, highest: 91 92 93 96 97

-----

sexe

n	missing	distinct
2000	0	2

Value	Homme	Femme
Frequency	899	1101
Proportion	0.45	0.55

```

-----
nivetud
  n  missing distinct
  1888      112       8

lowest : N'a jamais fait d'etudes
        rete ses etudes, avant la dernière année d'études primaires Dernière année
        d'études primaires
        1er cycl
        e
        e
highest: 1er cycle
        e cycle
        hnique ou professionnel court
        fessionnel long
        e supérieur
        2em
        Enseignement tec
        Enseignement technique ou pro
        Enseignement supérieur y compris techniqu
        2em

-----
poids
    n  missing distinct      Info      Mean      Gmd
  2000      0      1877      1      5536      4553
  .05      .10      .25      .50      .75      .90
  799.8   1161.7   2221.8   4631.2   7626.5  10819.0
  .95
  13647.9

lowest :    78.08    92.68    92.94   127.90   153.01
highest: 27195.84 29548.79 29570.79 29657.94 31092.14

-----
occup
  n  missing distinct
  2000      0       7

lowest : Exerce une profession Chomeur
        traite Retire des affaires
highest: Etudiant, eleve Retraite
        u foyer Autre inactif
        Etudiant, eleve
        Retire des affaires
        Re
        A

Exerce une profession (1049, 0.524), Chomeur (134, 0.067),
Etudiant, eleve (94, 0.047), Retraite (392, 0.196), Retire
des affaires (77, 0.038), Au foyer (171, 0.086), Autre
inactif (83, 0.042)

-----
qualif
  n  missing distinct

```

```

1653      347      7

lowest : Ouvrier specialise      Ouvrier qualifie      Technicien
n                  Profession intermediaire Cadre
highest: Technicien            Profession intermediaire Cadr
e                  Employe                  Autre

Ouvrier specialise (203, 0.123), Ouvrier qualifie (292,
0.177), Technicien (86, 0.052), Profession intermediaire
(160, 0.097), Cadre (260, 0.157), Employe (594, 0.359),
Autre (58, 0.035)
-----
freres.soeurs
  n  missing  distinct    Info     Mean     Gmd
2000      0        19   0.977   3.283   2.87
  .05      .10       .25     .50     .75     .90
  0        1        1       2       5       7
  .95
  9

lowest :  0  1  2  3  4, highest: 14 15 16 18 22

0 (167, 0.084), 1 (407, 0.203), 2 (427, 0.214), 3 (284,
0.142), 4 (210, 0.105), 5 (151, 0.076), 6 (99, 0.050), 7
(94, 0.047), 8 (52, 0.026), 9 (37, 0.018), 10 (21, 0.011),
11 (21, 0.011), 12 (8, 0.004), 13 (10, 0.005), 14 (4,
0.002), 15 (4, 0.002), 16 (1, 0.000), 18 (2, 0.001), 22 (1,
0.000)
-----
also
  n  missing  distinct
2000      0        3

Value          Oui      Non Ne sait pas
Frequency      936      1037      27
Proportion     0.468     0.518     0.014
-----
relig
  n  missing  distinct
2000      0        6

lowest : Pratiquant regulier      Pratiquant occasionnel      Appartenanc
e sans pratique Ni croyance ni appartenance Rejet

```

```

highest: Pratiquant occasionnel      Appartenance sans pratique Ni croyance
ni appartenance Rejet                  NSP ou NVPR

Pratiquant regulier (266, 0.133), Pratiquant occasionnel
(442, 0.221), Appartenance sans pratique (760, 0.380), Ni
croyance ni appartenance (399, 0.200), Rejet (93, 0.046),
NSP ou NVPR (40, 0.020)
-----
trav.imp
    n   missing  distinct
    1048       952        4

Le plus important (29, 0.028), Aussi important que le reste
(259, 0.247), Moins important que le reste (708, 0.676),
Peu important (52, 0.050)
-----
trav.satisf
    n   missing  distinct
    1048       952        3

Value      Satisfaction  Insatisfaction      Equilibre
Frequency          480           117            451
Proportion         0.458          0.112          0.430
-----
hard.rock
    n   missing  distinct
    2000       0        2

Value      Non     Oui
Frequency  1986     14
Proportion 0.993  0.007
-----
lecture.bd
    n   missing  distinct
    2000       0        2

Value      Non     Oui
Frequency  1953     47
Proportion 0.977  0.024
-----
peche.chasse
    n   missing  distinct
    2000       0        2

```

```
Value      Non    Oui
Frequency   1776   224
Proportion  0.888  0.112
-----
cuisine
  n  missing distinct
  2000       0        2

Value      Non    Oui
Frequency   1119   881
Proportion  0.559  0.440
-----
bricol
  n  missing distinct
  2000       0        2

Value      Non    Oui
Frequency   1147   853
Proportion  0.574  0.426
-----
cinema
  n  missing distinct
  2000       0        2

Value      Non    Oui
Frequency   1174   826
Proportion  0.587  0.413
-----
sport
  n  missing distinct
  2000       0        2

Value      Non    Oui
Frequency   1277   723
Proportion  0.638  0.362
-----
heures.tv
  n  missing distinct      Info      Mean      Gmd
  1995       5       29     0.972    2.247    1.877
  .05       .10      .25      .50      .75      .90
  0         0       1        2        3        4
  .95
```

```
5
lowest : 0.0 0.1 0.2 0.3 0.4, highest: 8.0 9.0 10.0 11.0 12.0
-----
```

## Quelques calculs simples

Maintenant que nous savons accéder aux variables, effectuons quelques calculs simples comme la moyenne, la médiane, le minimum et le maximum, à l'aide des fonctions `mean`, `median`, `min` et `max`.

```
R> mean(d$age)
```

```
[1] 48.16
```

```
R> median(d$age)
```

```
[1] 48
```

```
R> min(d$age)
```

```
[1] 18
```

```
R> max(d$age)
```

```
[1] 97
```

### NOTE

Au sens strict, il ne s'agit pas d'un véritable âge moyen puisqu'il faudrait ajouter 0,5 à cette valeur calculée, un âge moyen se calculant à partir d'âges exacts et non à partir d'âges révolus. Voir le chapitre Calculer un âge, page 1103.

On peut aussi très facilement obtenir un tri à plat à l'aide la fonction `table` :

```
R> table(d$qualif)
```

Ouvrier specialise	Ouvrier qualifie
203	292
Technicien Profession intermediaire	
86	160
Cadre	Employe
260	594
Autre	
58	

La fonction `summary`, bien pratique, permet d'avoir une vue résumée d'une variable. Elle s'applique à tout type d'objets (y compris un tableau de données entier) et s'adapte à celui-ci.

```
R> summary(d$age)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
18.0	35.0	48.0	48.2	60.0	97.0

```
R> summary(d$qualif)
```

Ouvrier specialise	Ouvrier qualifie
203	292
Technicien Profession intermediaire	
86	160
Cadre	Employe
260	594
Autre	NA's
58	347

```
R> summary(d)
```

id	age	sexe
Min. : 1	Min. :18.0	Homme: 899
1st Qu.: 501	1st Qu.:35.0	Femme:1101
Median :1000	Median :48.0	
Mean :1000	Mean :48.2	
3rd Qu.:1500	3rd Qu.:60.0	
Max. :2000	Max. :97.0	

		nivetud		
Enseignement technique ou professionnel court		:463		
Enseignement superieur y compris technique superieur:		441		
Derniere annee d'etudes primaires		:341		
1er cycle		:204		
2eme cycle		:183		
(Other)		:256		
NA's		:112		
	poids	occup		
Min. : 78	Exerce une profession:	1049		
1st Qu.: 2222	Chomeur	: 134		
Median : 4631	Etudiant, eleve	: 94		
Mean : 5536	Retraite	: 392		
3rd Qu.: 7627	Retire des affaires	: 77		
Max. :31092	Au foyer	: 171		
	Autre inactif	: 83		
	qualif	freres.soeurs		
Employe	:594	Min. : 0.00		
Ouvrier qualifie	:292	1st Qu.: 1.00		
Cadre	:260	Median : 2.00		
Ouvrier specialise	:203	Mean : 3.28		
Profession intermediaire:	160	3rd Qu.: 5.00		
(Other)	:144	Max. :22.00		
NA's	:347			
	cuso	relig		
Oui : 936	Pratiquant regulier	:266		
Non :1037	Pratiquant occasionnel	:442		
Ne sait pas: 27	Appartenance sans pratique	:760		
	Ni croyance ni appartenance	:399		
	Rejet	: 93		
	NSP ou NVPR	: 40		
	trav.imp	trav.satisf		
Le plus important : 29	Satisfaction	:480		
Aussi important que le reste:259	Insatisfaction	:117		
Moins important que le reste:708	Equilibre	:451		
Peu important : 52	NA's	:952		
NA's :952				
hard.rock	lecture.bd	peche.chasse	cuisine	bricol
Non:1986	Non:1953	Non:1776	Non:1119	Non:1147
Oui: 14	Oui: 47	Oui: 224	Oui: 881	Oui: 853

```
cinema      sport      heures.tv
Non:1174    Non:1277   Min.   : 0.00
Oui: 826    Oui: 723   1st Qu.: 1.00
              Median : 2.00
              Mean   : 2.25
              3rd Qu.: 3.00
              Max.   :12.00
              NA's   :5
```

## Nos premiers graphiques

R est très puissant en termes de représentations graphiques, notamment grâce à des extensions dédiées. Pour l'heure contentons-nous d'un premier essai à l'aide de la fonction générique `plot`.

```
R> plot(d$sex)
```

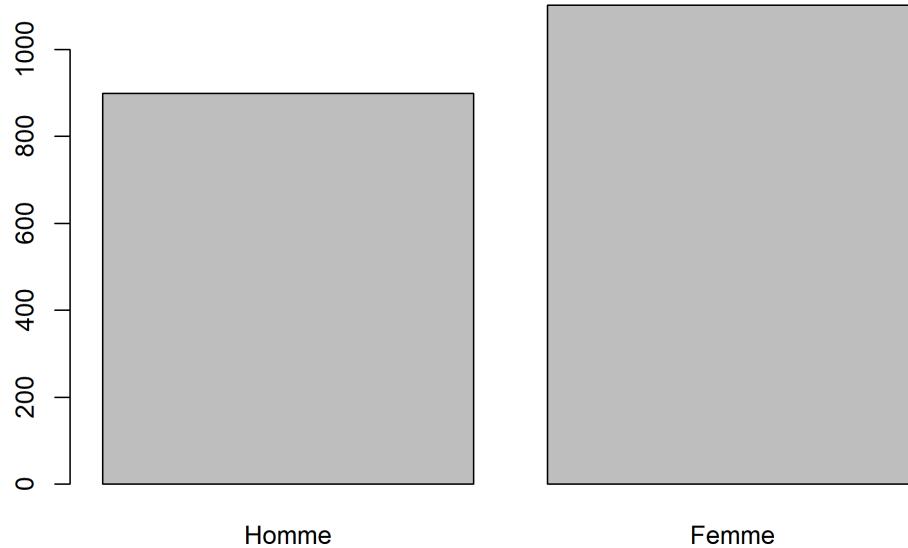


Figure 6. Nombre d'observations par sexe

Essayons avec deux variables :

```
R> plot(d$hard.rock, d$age)
```

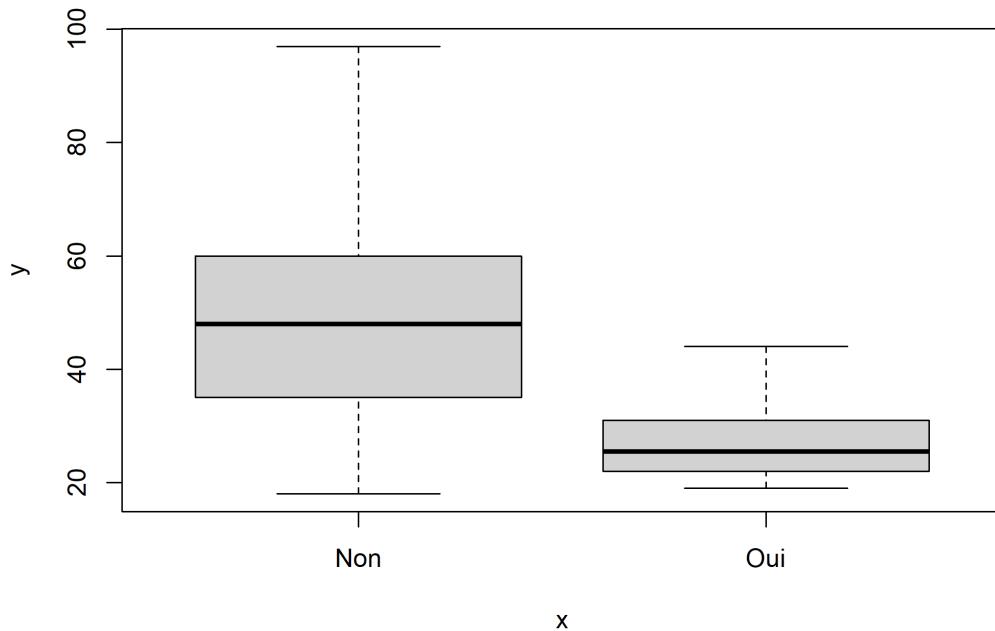


Figure 7. Âge des enquêtés selon qu'ils écoutent ou non du hard rock

Il semblerait bien que les amateurs de hard rock soient plus jeunes.

## Et ensuite ?

Nous n'avons qu'entr'aperçu les possibilités de **R**. Avant de pouvoir nous lancer dans des analyses statistiques, il est préférable de revenir un peu aux fondamentaux de **R** (les types d'objets, la syntaxe, le recodage de variables...) mais aussi comment installer des extensions, importer des données, etc. Nous vous conseillons donc de poursuivre la lecture de la section *Prise en main* puis de vous lancer à l'assaut de la section *Statistique introductive*.



# Extensions (installation, mise à jour)

Présentation .....	55
Le «tidyverse» .....	56
Installation depuis CRAN .....	56
Installation depuis GitHub .....	57
Mise à jour des extensions .....	58

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #01 (premier contact avec R & RStudio) sur [YouTube](#).

## Présentation

L'installation par défaut du logiciel **R** contient le cœur du programme ainsi qu'un ensemble de fonctions de base fournissant un grand nombre d'outils de traitement de données et d'analyse statistiques.

R étant un logiciel libre, il bénéficie d'une forte communauté d'utilisateurs qui peuvent librement contribuer au développement du logiciel en lui ajoutant des fonctionnalités supplémentaires. Ces contributions prennent la forme d'extensions (*packages* en anglais) pouvant être installées par l'utilisateur et fournissant alors diverses fonctionnalités supplémentaires.

Il existe un très grand nombre d'extensions (plus de 6500 à ce jour), qui sont diffusées par un réseau baptisé **CRAN** (*Comprehensive R Archive Network*).

La liste de toutes les extensions disponibles sur **CRAN** est disponible ici : <http://cran.r-project.org/web/packages/>.

Pour faciliter un peu le repérage des extensions, il existe un ensemble de regroupements thématiques (économétrie, finance, génétique, données spatiales...) baptisés *Task views* : <http://cran.r-project.org/>

[web/views/](#).

On y trouve notamment une *Task view* dédiée aux sciences sociales, listant de nombreuses extensions potentiellement utiles pour les analyses statistiques dans ce champ disciplinaire : <http://cran.r-project.org/web/views/SocialSciences.html>.

On peut aussi citer le site *Awesome R* (<https://awesome-r.com/>) qui fournit une liste d’extensions choisies et triées par thématique.

## Le «tidyverse»

Hadley Wickham est professeur associé à l’université de Rice et scientifique en chef à **Rstudio**. Il a développé de nombreux extensions pour **R** (plus d’une cinquantaine à ce jour) qui, pour la plupart, fonctionne de manière harmonisée entre elles. Par ailleurs, la plupart s’intègre parfaitement avec **RStudio**. Cet ensemble d’extensions est appelé **tidyverse** et est développé sur GitHub : <https://github.com/tidyverse/>. Une présentation plus générale du **tidyverse** est disponible sur le site de **RStudio** (<https://www.rstudio.com/products/rpackages/>) et sur un site dédié (<http://tidyverse.org/>).

Pour certaines tâches, il peut exister plusieurs solutions / extensions différentes pour les réaliser. Dans la mesure où il n’est pas possible d’être exhaustif, nous avons fait le choix dans le cadre d’**analyse-R** de choisir en priorité, lorsque cela est possible, les extensions du **tidyverse**, en particulier **haven**, **readr** et **readxl** pour l’import de données, **dplyr**, **tidyr** ou **reshape2** pour la manipulation de données, **ggplot2** pour les graphiques, **lubridate** pour la gestion des dates, **forcats** pour la manipulation des facteurs ou encore **stringr** pour la manipulation de chaînes de caractères.

Il existe par ailleurs une extension homonyme **tidyverse**. L’installation (voir ci-dessous) de cette extension permet l’installation automatique de l’ensemble des autres extensions du **tidyverse**. Le chargement de cette extension avec la fonction **library** (voir ci-après) permet de charger en mémoire en une seule opération les principales extensions du **tidyverse**, à savoir **ggplot2**, **tibble**, **tidyr**, **readr**, **purrr** et **dplyr**.

Pour une présentation plus poussée, voir le chapitre consacré au **tidyverse**, page 59.

## Installation depuis CRAN

L’installation d’une extension se fait par la fonction **install.packages**, à qui on fournit le nom de l’extension. Par exemple, si on souhaite installer l’extension **ade4** :

```
R> install.packages("ade4", dep = TRUE)
```

L’option **dep=TRUE** indique à **R** de télécharger et d’installer également toutes les extensions dont l’extension choisie dépend pour son fonctionnement.

Sous **RStudio**, on pourra également cliquer sur *Install* dans l’onglet *Packages* du quadrant inférieur droit.

Une fois l'extension installée, elle peut être appelée depuis la console ou un fichier script avec la fonction `library` ou la fonction `require` :

```
R> library(ade4)
```

À partir de là, on peut utiliser les fonctions de l'extension, consulter leur page d'aide en ligne, accéder aux jeux de données qu'elle contient, etc.

Pour mettre à jour l'ensemble des extensions installées, <dfn data-index="mise à jour, extensions"> la fonction `update.packages` suffit :

```
R> update.packages()
```

Sous **RStudio**, on pourra alternativement cliquer sur *Update* dans l'onglet *Packages* du quadrant inférieur droit.

Si on souhaite désinstaller une extension précédemment installée, on peut utiliser la fonction `remove.packages` :

```
R> remove.packages("ade4")
```

#### IMPORTANT

Il est important de bien comprendre la différence entre `install.packages` et `library`. La première va chercher les extensions sur internet et les installe en local sur le disque dur de l'ordinateur. On n'a besoin d'effectuer cette opération qu'une seule fois. La seconde lit les informations de l'extension sur le disque dur et les met à disposition de **R**. On a besoin de l'exécuter à chaque début de session ou de script.

## Installation depuis GitHub

Certains packages sont développés sur **GitHub**. Dès lors, la version de développement sur **GitHub** peut contenir des fonctions qui ne sont pas encore disponibles dans la version stable disponible sur **CRAN**. Ils arrivent aussi parfois que certains packages ne soient disponibles que sur **GitHub**.

L'installation d'un package depuis **GitHub** est très facile grâce à la fonction `install_github` de l'extension `devtools` (que l'on aura préalablement installée depuis **CRAN** ;-)).

## Mise à jour des extensions

Il est facile de mettre à jour l’ensemble des extensions installées, soit avec la fonction, `update.packages` soit en cliquant sur *Update* dans l’onglet *Packages* du quadrant inférieur droit.

# Introduction au tidyverse

Extensions . . . . .	59
Installation . . . . .	60
tidy data . . . . .	60
tibbles . . . . .	61

## NOTE

La version originale de ce chapitre a été écrite par Julien Barnier dans le cadre de son [Introduction à R et au tidyverse](#).

## Extensions

Le terme **tidyverse** est une contraction de *tidy* (qu'on pourrait traduire par “bien rangé”) et de *universe*. Il s’agit en fait d’une collection d’extensions conçues pour travailler ensemble et basées sur une philosophie commune.

Elles abordent un très grand nombre d’opérations courantes dans **R** (la liste n’est pas exhaustive) :

- visualisation
- manipulation des tableaux de données
- import/export de données
- manipulation de variables
- extraction de données du Web
- programmation

Un des objectifs de ces extensions est de fournir des fonctions avec une syntaxe cohérente, qui fonctionnent bien ensemble, et qui retournent des résultats prévisibles. Elles sont en grande partie issues du travail d'[Hadley Wickham](#), qui travaille désormais pour [RStudio](#).

# Installation

**tidyverse** est également le nom d’une extension qu’on peut installer de manière classique, soit via le bouton *Install* de l’onglet *Packages* de **RStudio**, soit en utilisant la commande :

```
R> install.packages("tidyverse")
```

Cette commande va en fait installer plusieurs extensions qui constituent le «coeur» du *tidyverse*, à savoir :

- **ggplot2** (visualisation)
- **dplyr** (manipulation des données)
- **tidyr** (remise en forme des données)
- **purrr** (programmation)
- **readr** (importation de données)
- **tibble** (tableaux de données)
- **forcats** (variables qualitatives)
- **stringr** (chaînes de caractères)

De la même manière, charger l’extension avec :

```
R> library(tidyverse)
```

Chargera l’ensemble des extensions précédentes.

Il existe d’autres extensions qui font partie du **tidyverse** mais qui doivent être chargées explicitement, comme par exemple **readxl** (pour l’importation de données depuis des fichiers Excel).

La liste complète des extensions se trouve sur [le site officiel du tidyverse](#).

## tidy data

Le **tidyverse** est en partie fondé sur le concept de *tidy data*, développé à l’origine par Hadley Wickham dans un [article de 2014](#) du *Journal of Statistical Software*.

Il s’agit d’un modèle d’organisation des données qui vise à faciliter le travail souvent long et fastidieux de nettoyage et de préparation préalable à la mise en oeuvre de méthodes d’analyse.

Les principes d’un jeu de données *tidy* sont les suivants :

1. chaque variable est une colonne
2. chaque observation est une ligne
3. chaque type d’observation est dans une table différente

Un chapitre dédié à **tidyR**, page 299 présente comment définir et rendre des données *tidy* avec l'extension **tidyR**.

Les extensions du **tidyverse**, notamment **ggplot2** et **dplyr**, sont prévues pour fonctionner avec des données *tidy*.

## tibbles

Une autre particularité du **tidyverse** est que ces extensions travaillent avec des tableaux de données au format **tibble**, qui est une évolution plus moderne du classique *data frame* du R de base. Ce format est fourni et géré par l'extension du même nom (**tibble**), qui fait partie du cœur du **tidyverse**. La plupart des fonctions des extensions du **tidyverse** acceptent des *data frames* en entrée, mais retournent un objet de classe **tibble**.

Contrairement aux *data frames*, les *tibbles* :

- n'ont pas de noms de lignes (*rownames*)
- autorisent des noms de colonnes invalides pour les *data frames* (espaces, caractères spéciaux, nombres...) <sup>1</sup>
- s'affichent plus intelligemment que les *data frames* : seules les premières lignes sont affichées, ainsi que quelques informations supplémentaires utiles (dimensions, types des colonnes...)
- ne font pas de *partial matching* sur les noms de colonnes <sup>2</sup>
- affichent un avertissement si on essaie d'accéder à une colonne qui n'existe pas

Pour autant, les *tibbles* restent compatibles avec les *data frames*. On peut ainsi facilement convertir un *data frame* en *tibble* avec **as\_tibble** :

```
R> library(tidyverse)
as_tibble(mtcars)
```

```
# A tibble: 32 x 12
  mpg   cyl  disp    hp  drat    wt  qsec    vs    am
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 21     6   160   110  3.9   2.62  16.5     0     1
2 21     6   160   110  3.9   2.88  17.0     0     1
3 22.8   4   108   93   3.85  2.32  18.6     1     1
4 21.4   6   258   110  3.08  3.22  19.4     1     0
5 18.7   8   360   175  3.15  3.44  17.0     0     0
6 18.1   6   225   105  2.76  3.46  20.2     1     0
7 14.3   8   360   245  3.21  3.57  15.8     0     0
```

1. Quand on veut utiliser des noms de ce type, on doit les entourer avec des *backticks* (`)

2. Dans R de base, si une table **d** contient une colonne **qualif**, **d\$qual** retournera cette colonne.

```

8 24.4      4 147.     62 3.69  3.19  20      1      0
9 22.8      4 141.     95 3.92  3.15  22.9     1      0
10 19.2     6 168.    123 3.92  3.44  18.3     1      0
# ... with 22 more rows, and 3 more variables: gear <dbl>,
#   carb <dbl>, name <chr>

```

Si le *data frame* d'origine a des *rownames*, on peut d'abord les convertir en colonnes avec `rownames_to_columns` :

```
R> d <- as_tibble(rownames_to_column(mtcars))
d
```

```

# A tibble: 32 x 13
  rowname     mpg cyl disp  hp drat    wt  qsec    vs
  <chr>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4  21     6   160   110  3.9   2.62  16.5   0
2 Mazda RX~  21     6   160   110  3.9   2.88  17.0   0
3 Datsun 7~  22.8   4   108   93   3.85  2.32  18.6   1
4 Hornet 4~  21.4   6   258   110  3.08  3.22  19.4   1
5 Hornet S~  18.7   8   360   175  3.15  3.44  17.0   0
6 Valiant    18.1   6   225   105  2.76  3.46  20.2   1
7 Duster 3~  14.3   8   360   245  3.21  3.57  15.8   0
8 Merc 240D  24.4   4   147.   62   3.69  3.19  20      1
9 Merc 230   22.8   4   141.   95   3.92  3.15  22.9   1
10 Merc 280   19.2   6   168.  123   3.92  3.44  18.3   1
# ... with 22 more rows, and 4 more variables: am <dbl>,
#   gear <dbl>, carb <dbl>, name <chr>

```

À l'inverse, on peut à tout moment convertir un tibble en *data frame* avec `as.data.frame` :

```
R> as.data.frame(d)
```

	rowname	mpg	cyl	disp	hp	drat	wt	qsec	vs
1	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	
2	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	
3	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	
4	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	
5	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	
6	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	
7	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	
8	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	
9	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	
10	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	
11	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	

12	Merc	450SE	16.4	8	275.8	180	3.07	4.070	17.40
13	Merc	450SL	17.3	8	275.8	180	3.07	3.730	17.60
14	Merc	450SLC	15.2	8	275.8	180	3.07	3.780	18.00
15	Cadillac	Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98
16	Lincoln	Continental	10.4	8	460.0	215	3.00	5.424	17.82
17	Chrysler	Imperial	14.7	8	440.0	230	3.23	5.345	17.42
18	Fiat	128	32.4	4	78.7	66	4.08	2.200	19.47
19	Honda	Civic	30.4	4	75.7	52	4.93	1.615	18.52
20	Toyota	Corolla	33.9	4	71.1	65	4.22	1.835	19.90
21	Toyota	Corona	21.5	4	120.1	97	3.70	2.465	20.01
22	Dodge	Challenger	15.5	8	318.0	150	2.76	3.520	16.87
23	AMC	Javelin	15.2	8	304.0	150	3.15	3.435	17.30
24		Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41
25	Pontiac	Firebird	19.2	8	400.0	175	3.08	3.845	17.05
26	Fiat	X1-9	27.3	4	79.0	66	4.08	1.935	18.90
27	Porsche	914-2	26.0	4	120.3	91	4.43	2.140	16.70
28	Lotus	Europa	30.4	4	95.1	113	3.77	1.513	16.90
29	Ford	Pantera L	15.8	8	351.0	264	4.22	3.170	14.50
30	Ferrari	Dino	19.7	6	145.0	175	3.62	2.770	15.50
31	Maserati	Bora	15.0	8	301.0	335	3.54	3.570	14.60
32	Volvo	142E	21.4	4	121.0	109	4.11	2.780	18.60
	vs	am	gear	carb					name
1	0	1	4	4					Mazda RX4
2	0	1	4	4					Mazda RX4 Wag
3	1	1	4	1					Datsun 710
4	1	0	3	1					Hornet 4 Drive
5	0	0	3	2					Hornet Sportabout
6	1	0	3	1					Valiant
7	0	0	3	4					Duster 360
8	1	0	4	2					Merc 240D
9	1	0	4	2					Merc 230
10	1	0	4	4					Merc 280
11	1	0	4	4					Merc 280C
12	0	0	3	3					Merc 450SE
13	0	0	3	3					Merc 450SL
14	0	0	3	3					Merc 450SLC
15	0	0	3	4					Cadillac Fleetwood
16	0	0	3	4					Lincoln Continental
17	0	0	3	4					Chrysler Imperial
18	1	1	4	1					Fiat 128
19	1	1	4	2					Honda Civic
20	1	1	4	1					Toyota Corolla
21	1	0	3	1					Toyota Corona
22	0	0	3	2					Dodge Challenger
23	0	0	3	2					AMC Javelin
24	0	0	3	4					Camaro Z28
25	0	0	3	2					Pontiac Firebird

26	1	1	4	1	Fiat	X1-9
27	0	1	5	2	Porsche	914-2
28	1	1	5	2	Lotus	Europa
29	0	1	5	4	Ford	Pantera L
30	0	1	5	6	Ferrari	Dino
31	0	1	5	8	Maserati	Bora
32	1	1	4	2	Volvo	142E

Là encore, on peut convertir la colonne `rowname` en “vrais” `rownames` avec `column_to_rownames` :

```
R> column_to_rownames(as.data.frame(d))
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1

	am	gear	carb	name
Mazda RX4	1	4	4	Mazda RX4
Mazda RX4 Wag	1	4	4	Mazda RX4 Wag
Datsun 710	1	4	1	Datsun 710
Hornet 4 Drive	0	3	1	Hornet 4 Drive
Hornet Sportabout	0	3	2	Hornet Sportabout
Valiant	0	3	1	Valiant
Duster 360	0	3	4	Duster 360
Merc 240D	0	4	2	Merc 240D
Merc 230	0	4	2	Merc 230
Merc 280	0	4	4	Merc 280
Merc 280C	0	4	4	Merc 280C
Merc 450SE	0	3	3	Merc 450SE
Merc 450SL	0	3	3	Merc 450SL
Merc 450SLC	0	3	3	Merc 450SLC
Cadillac Fleetwood	0	3	4	Cadillac Fleetwood
Lincoln Continental	0	3	4	Lincoln Continental
Chrysler Imperial	0	3	4	Chrysler Imperial
Fiat 128	1	4	1	Fiat 128
Honda Civic	1	4	2	Honda Civic
Toyota Corolla	1	4	1	Toyota Corolla
Toyota Corona	0	3	1	Toyota Corona
Dodge Challenger	0	3	2	Dodge Challenger
AMC Javelin	0	3	2	AMC Javelin
Camaro Z28	0	3	4	Camaro Z28
Pontiac Firebird	0	3	2	Pontiac Firebird
Fiat X1-9	1	4	1	Fiat X1-9
Porsche 914-2	1	5	2	Porsche 914-2
Lotus Europa	1	5	2	Lotus Europa
Ford Pantera L	1	5	4	Ford Pantera L
Ferrari Dino	1	5	6	Ferrari Dino
Maserati Bora	1	5	8	Maserati Bora
Volvo 142E	1	4	2	Volvo 142E

**NOTE**

Les deux fonctions `column_to_rrownames` et `rownames_to_column` acceptent un argument supplémentaire `var` qui permet d'indiquer un nom de colonne autre que le nom `rowname` utilisé par défaut pour créer ou identifier la colonne contenant les noms de lignes.



# Vecteurs, indexation et assignation

Présentation des vecteurs . . . . .	68
Les principaux types de vecteurs . . . . .	68
Création . . . . .	70
La fonction c . . . . .	70
La fonction rep . . . . .	71
La fonction seq . . . . .	72
L'opérateur : . . . . .	73
Longueur d'un vecteur . . . . .	73
Quelques vecteurs remarquables . . . . .	74
Combiner des vecteurs . . . . .	75
Valeurs manquantes . . . . .	76
Indexation par position . . . . .	77
Des vecteurs nommés . . . . .	79
Indexation par nom . . . . .	80
Indexation par condition . . . . .	81
Assignation par indexation . . . . .	84
En résumé . . . . .	86

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #02 (les bases du langage R) sur [YouTube](#).

Nous allons reprendre plusieurs éléments de base du langage R que nous avons déjà abordés mais de manière plus formelle. Une bonne compréhension des bases du langage, bien qu'un peu ardue de prime abord, permet de comprendre le sens des commandes qu'on utilise et de pleinement exploiter la puissance que R offre en matière de manipulation de données.

Dans ce chapitre, nous reviendrons sur les vecteurs, tandis que les listes et les tableaux de données seront

abordés dans un chapitre dédié, page 87.

## Présentation des vecteurs

Les vecteurs sont l’un des objets de base de **R** et correspondent à une «liste de valeurs». Leurs propriétés fondamentales sont :

- les vecteurs sont unidimensionnels (i.e. ce sont des objets à une seule dimension, à la différence d’une matrice par exemple) ;
- toutes les valeurs d’un vecteur sont d’un seul et même type ;
- les vecteurs ont une longueur qui correspond au nombre de valeurs contenues dans le vecteur.

## Les principaux types de vecteurs

Dans **R**, il existe quatre types fondamentaux de vecteurs :

- les nombres réels (c’est-à-dire les nombres décimaux que nous utilisons au quotidien),
- les nombres entiers,
- les chaînes de caractères (qui correspondent à du texte) et
- les valeurs logiques ou valeurs booléennes, à savoir «vrai» ou «faux».

Pour connaître la nature d’un objet, le plus simple est d’utiliser la fonction `class`. Par exemple :

```
R> class(12.5)
```

```
[1] "numeric"
```

La réponse `"numeric"` nous indique qu’il s’agit d’un nombre réel. Parfois, vous pourrez rencontrer le terme `"double"` qui désigne également les nombres réels. Notez que **R** étant anglophone, la décimale est indiquée avec un point (.) et non avec une virgule comme c’est l’usage en français.

Essayons avec un nombre entier :

```
R> class(3)
```

```
[1] "numeric"
```

Sous **R**, lorsqu’on tape un nombre sans autre précision, il est considéré par défaut comme un nombre réel. Pour indiquer spécifiquement qu’on veut un nombre entier, il faut rajouter le suffixe `L` :

```
R> class(3L)
```

```
[1] "integer"
```

Au quotidien, il arrive rarement d'avoir à utiliser ce suffixe, mais il est toujours bon de le connaître au cas où vous le rencontriez dans des manuels ou des exemples de code.

Pour saisir une chaîne de caractères, on aura recours aux doubles guillemets droits ( " ) :

```
R> class("abc")
```

```
[1] "character"
```

Il est également possible d'utiliser des guillemets simples ( ' ), dès lors qu'on utilise bien le même type de guillemets pour indiquer le début et la fin de la chaîne de caractères (par exemple 'abc' ).

Enfin, les valeurs logiques s'indiquent avec `TRUE` pour vrai et `FALSE` pour faux. Il est aussi possible d'utiliser les raccourcis `T` et `F`. Attention à bien utiliser les majuscules, `R` étant sensible à la casse.

```
R> class(TRUE)
```

```
[1] "logical"
```

En résumé, les classes R des quatre types fondamentaux de vecteur sont :

Exemple	Classe R	Type
<code>5L</code>	<code>integer</code>	nombre entier
<code>3.14</code>	<code>numeric</code>	nombre réel
<code>"abcd"</code>	<code>character</code>	chaîne de caractères
<code>TRUE</code>	<code>logical</code>	booléenne

En plus des types de base, il existe de nombreuses autres classes de vecteurs dans R que nous aborderons ultérieurement dans d'autres chapitres. Les plus courantes sont :

Classe R	Type
factor	facteur, page 118
labelled	vecteur labellisé, page 123
Date	date, page 285
POSIXct	date et heure, page 285

# Création

## La fonction c

Pour créer un vecteur, on utilisera la fonction `c`, la lettre «c» étant un raccourci du mot anglais *combine* puisque cette fonction permet de combiner des valeurs individuelles dans un vecteur unique. Il suffit de lui passer la liste des valeurs à combiner :

```
R> taille <- c(1.88, 1.65, 1.92, 1.76)
taille
```

```
[1] 1.88 1.65 1.92 1.76
```

```
R> class(taille)
```

```
[1] "numeric"
```

```
R> sexe <- c("h", "f", "h", "f")
sexe
```

```
[1] "h" "f" "h" "f"
```

```
R> class(sexe)
```

```
[1] "character"
```

```
R> urbain <- c(TRUE, TRUE, FALSE, FALSE)
urbain
```

```
[1] TRUE TRUE FALSE FALSE
```

```
R> class(urbain)
```

```
[1] "logical"
```

Nous l'avons vu, toutes les valeurs d'un vecteur doivent obligatoirement être du même type. Dès lors, si on essaie de combiner des valeurs de différents types, R essaiera de les convertir au mieux. Par exemple :

```
R> x <- c(2L, 3.14, "a")
x
```

```
[1] "2"     "3.14"  "a"
```

```
R> class(x)
```

```
[1] "character"
```

Dans le cas présent, toutes les valeurs ont été converties en chaînes de caractères.

## La fonction rep

Dans certaines situations, on peut avoir besoin de créer un vecteur d'une certaine longueur mais dont toutes les valeurs sont identiques. Cela se réalise facilement avec `rep` à qui on indiquera la valeur à répéter puis le nombre de répétitions :

```
R> rep(2, 10)
```

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

On peut aussi lui indiquer plusieurs valeurs qui seront alors répétées en boucle :

```
R> rep(c("a", "b"), 3)
```

```
[1] "a" "b" "a" "b" "a" "b"
```

## La fonction seq

Dans d’autres situations, on peut avoir besoin de créer un vecteur contenant une suite de valeurs, ce qui se réalise aisément avec `seq` à qui on précisera les arguments `from` (point de départ), `to` (point d’arrivée) et `by` (pas). Quelques exemples valent mieux qu’un long discours :

```
R> seq(1, 10)
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
R> seq(5, 17, by = 2)
```

```
[1] 5 7 9 11 13 15 17
```

```
R> seq(10, 0)
```

```
[1] 10 9 8 7 6 5 4 3 2 1 0
```

```
R> seq(100, 10, by = -10)
```

```
[1] 100 90 80 70 60 50 40 30 20 10
```

```
R> seq(1.23, 5.67, by = 0.33)
```

```
[1] 1.23 1.56 1.89 2.22 2.55 2.88 3.21 3.54 3.87 4.20 4.53
```

```
[12] 4.86 5.19 5.52
```

## L'opérateur :

L'opérateur `:` est un raccourci de la fonction `seq` pour créer une suite de nombres entiers. Il s'utilise ainsi :

```
R> 1:5
```

```
[1] 1 2 3 4 5
```

```
R> 24:32
```

```
[1] 24 25 26 27 28 29 30 31 32
```

```
R> 55:43
```

```
[1] 55 54 53 52 51 50 49 48 47 46 45 44 43
```

Nous verrons un peu plus loin que ce raccourci est fort pratique.

## Longueur d'un vecteur

Un vecteur dispose donc d'une longueur qui correspond au nombre de valeurs qui le composent. Elle s'obtient avec `length` :

```
R> length(taille)
```

```
[1] 4
```

```
R> length(c("a", "b"))
```

```
[1] 2
```

Il est possible de faire un vecteur de longueur nulle avec `c()`. Bien évidemment sa longueur est zéro.

```
R> length(c())
```

```
[1] 0
```

## Quelques vecteurs remarquables

R fournit quelques vecteurs particuliers qui sont directement accessibles :

- `LETTERS` : les 26 lettres de l'alphabet en majuscules
- `letters` : les 26 lettres de l'alphabet en minuscules
- `month.name` : les noms des 12 mois de l'année en anglais
- `month.abb` : la version abrégée des 12 mois en anglais
- `pi` : la constante mathématique  $\pi$

```
R> LETTERS
```

```
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"  
[15] "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

```
R> letters
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"  
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
R> length(letters)
```

```
[1] 26
```

```
R> month.name
```

```
[1] "January"    "February"   "March"       "April"  
[5] "May"         "June"        "July"        "August"  
[9] "September"  "October"    "November"   "December"
```

```
R> month.abb
```

```
[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep"  
[10] "Oct" "Nov" "Dec"
```

```
R> length(month.abb)
```

```
[1] 12
```

```
R> pi
```

```
[1] 3.141593
```

```
R> length(pi)
```

```
[1] 1
```

## Combiner des vecteurs

Pour combiner des vecteurs, rien de plus simple. Il suffit d'utiliser `c` ! Les valeurs des différents vecteurs seront mises bout à bout pour créer un unique vecteur.

```
R> x <- c(2, 1, 3, 4)  
length(x)
```

```
[1] 4
```

```
R> y <- c(9, 1, 2, 6, 3, 0)  
length(y)
```

```
[1] 6
```

```
R> z <- c(x, y)
z
```

```
[1] 2 1 3 4 9 1 2 6 3 0
```

```
R> length(z)
```

```
[1] 10
```

```
R> min_maj <- c(letters, LETTERS)
min_maj
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z" "A" "B"
[29] "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P"
[43] "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

```
R> length(min_maj)
```

```
[1] 52
```

## Valeurs manquantes

Lorsqu’on travaille avec des données d’enquête, il est fréquent que certaines données soient manquantes, en raison d’un refus du participant de répondre à une question donnée ou d’un oubli ou d’un dysfonctionnement du matériel de mesure, etc.

Une valeur manquante s’indique sous R avec `NA` (pour *not available*). Cette valeur peut s’appliquer à n’importe quel type de vecteur, qu’il soit numérique, textuel ou logique.

```
R> taille <- c(1.88, NA, 1.65, 1.92, 1.76, NA)
sexes <- c("h", "f", NA, "h", NA, "f")
```

Les valeurs manquantes sont prises en compte dans le calcul de la longueur du vecteur.

```
R> length(taille)
```

```
[1] 6
```

Il ne faut pas confondre `NA` avec un autre objet qu'on rencontre sous R qui s'appelle `NULL` et qui représente l'«objet vide». `NULL` ne contient absolument rien. La différence se comprend mieux lorsqu'on essaie de combiner ces objets :

```
R> c(NULL, NULL, NULL)
```

```
NULL
```

```
R> length(c(NULL, NULL, NULL))
```

```
[1] 0
```

On peut combiner `NULL` avec `NULL`, du vide plus du vide renverra toujours du vide dont la dimension est égale à zéro.

```
R> c(NA, NA, NA)
```

```
[1] NA NA NA
```

```
R> length(c(NA, NA, NA))
```

```
[1] 3
```

Par contre, un vecteur composé de trois valeurs manquantes a une longueur de 3, même si toutes ses valeurs sont manquantes.

## Indexation par position

L'indexation est l'une des fonctionnalités les plus puissantes mais aussi les plus difficiles à maîtriser de R. Il s'agit d'opérations permettant de sélectionner des sous-ensembles de valeurs en fonction de différents critères. Il existe trois types d'indexation : (i) l'indexation par position, (ii) l'indexation par nom et (iii) l'indexation par condition. Le principe est toujours le même : on indique entre crochets (`[]`) ce qu'on souhaite garder ou non.

Pour rappel, les crochets s’obtiennent sur un clavier français de type PC en appuyant sur la touche **Alt Gr** et la touche **(** ou **)**.

Commençons par l’indexation par position encore appelée indexation directe. Ce mode le plus simple d’indexation consiste à indiquer la position des éléments à conserver.

Reprendons notre vecteur `taille` :

```
R> taille
```

```
[1] 1.88 NA 1.65 1.92 1.76 NA
```

Si on souhaite le premier élément du vecteur, on peut faire :

```
R> taille[1]
```

```
[1] 1.88
```

Si on souhaite les trois premiers éléments ou les éléments 2, 5 et 6 :

```
R> taille[1:3]
```

```
[1] 1.88 NA 1.65
```

```
R> taille[c(2, 5, 6)]
```

```
[1] NA 1.76 NA
```

Si on veut le dernier élément :

```
R> taille[length(taille)]
```

```
[1] NA
```

Il est tout à fait possible de sélectionner les valeurs dans le désordre :

```
R> taille[c(5, 1, 4, 3)]
```

```
[1] 1.76 1.88 1.92 1.65
```

Dans le cadre de l'indexation par position, il est également possible de spécifier des nombres négatifs, auquel cas cela signifiera «toutes les valeurs sauf celles-là». Par exemple :

```
R> taille[c(-1, -5)]
```

```
[1] NA 1.65 1.92 NA
```

À noter, si on indique une position au-delà de la longueur du vecteur, R renverra NA . Par exemple :

```
R> taille[23:25]
```

```
[1] NA NA NA
```

## Des vecteurs nommés

Les différentes valeurs d'un vecteur peuvent être nommées. Une première manière de nommer les éléments d'un vecteur est de le faire à sa création :

```
R> sexe <- c(Michel = "h", Anne = "f", Dominique = NA, Jean = "h", Claude = NA, Marie = "f")
```

Lorsqu'on affiche le vecteur, la présentation change quelque peu.

```
R> sexe
```

Michel	Anne	Dominique	Jean	Claude	Marie
"h"	"f"	NA	"h"	NA	"f"

La liste des noms s'obtient avec names .

```
R> names(sexe)
```

```
[1] "Michel"    "Anne"       "Dominique" "Jean"  
[5] "Claude"   "Marie"
```

Pour ajouter ou modifier les noms d'un vecteur, on doit attribuer un nouveau vecteur de noms :

```
R> names(sexe) <- c("Michael", "Anna", "Dom", "John", "Alex", "Mary")
sexes
```

Michael	Anna	Dom	John	Alex	Mary
"h"	"f"	NA	"h"	NA	"f"

Pour supprimer tous les noms, il y a la fonction `uname` :

```
R> anonyme <- uname(sexe)
anonyme
```

```
[1] "h" "f" NA "h" NA "f"
```

## Indexation par nom

Lorsqu'un vecteur est nommé, il est dès lors possible d'accéder à ses valeurs à partir de leur nom. Il s'agit de l'indexation par nom.

```
R> sexe["Anna"]
```

```
Anna
"f"
```

```
R> sexe[c("Mary", "Michael", "John")]
```

```
Mary Michael John
"f"     "h"    "h"
```

Par contre il n'est pas possible d'utiliser l'opérateur `-` comme pour l'indexation directe. Pour exclure un élément en fonction de son nom, on doit utiliser une autre forme d'indexation, l'indexation par condition, expliquée dans la section suivante. On peut ainsi faire...

```
R> sexe[names(sexe) != "Dom"]
```

... pour sélectionner tous les éléments sauf celui qui s'appelle «Dom».

## Indexation par condition

L'indexation par condition consiste à fournir un vecteur logique indiquant si chaque élément doit être inclus (si `TRUE`) ou exclu (si `FALSE`). Par exemple :

```
R> sexe
```

Michael	Anna	Dom	John	Alex	Mary
"h"	"f"	NA	"h"	NA	"f"

```
R> sexe[c(TRUE, FALSE, FALSE, TRUE, FALSE, FALSE)]
```

Michael	John
"h"	"h"

Écrire manuellement une telle condition n'est pas très pratique à l'usage. Mais supposons que nous ayons également à notre disposition les deux vecteurs suivants, également de longueur 6.

```
R> urbain <- c(TRUE, FALSE, FALSE, FALSE, TRUE, TRUE)
poids <- c(80, 63, 75, 87, 82, 67)
```

Le vecteur `urbain` est un vecteur logique. On peut directement l'utiliser pour avoir le sexe des enquêtés habitant en milieu urbain :

```
R> sexe[urbain]
```

Michael	Alex	Mary
"h"	NA	"f"

Supposons qu'on souhaite maintenant avoir la taille des individus pesant 80 kilogrammes ou plus. Nous pouvons effectuer une comparaison à l'aide des opérateurs de comparaison suivants :

Opérateur de comparaison	Signification
<code>==</code>	égal à
<code>!=</code>	différent de
<code>&gt;</code>	strictement supérieur à
<code>&lt;</code>	strictement inférieur à
<code>&gt;=</code>	supérieur ou égal à
<code>&lt;=</code>	inférieur ou égal à

Voyons tout de suite un exemple :

```
R> poids >= 80
```

```
[1] TRUE FALSE FALSE TRUE TRUE FALSE
```

Que s'est-il passé ? Nous avons fourni à R une condition et il nous a renvoyé un vecteur logique avec autant d'éléments qu'il y a d'observations et dont la valeur est `TRUE` si la condition est remplie et `FALSE` dans les autres cas. Nous pouvons alors utiliser ce vecteur logique pour obtenir la taille des participants pesant 80 kilogrammes ou plus :

```
R> taille[poids >= 80]
```

```
[1] 1.88 1.92 1.76
```

On peut combiner ou modifier des conditions à l'aide des opérateurs logiques habituels :

Opérateur logique	Signification
<code>&amp;</code>	et logique
<code> </code>	ou logique
<code>!</code>	négation logique

Comment les utilise-t-on ? Voyons tout de suite un exemple. Supposons que je veuille identifier les personnes pesant 80 kilogrammes ou plus et vivant en milieu urbain :

```
R> poids >= 80 & urbain
```

```
[1] TRUE FALSE FALSE FALSE TRUE FALSE
```

Les résultats sont différents si je souhaite isoler les personnes pesant 80 kilogrammes ou plus **ou** vivant milieu urbain :

```
R> poids >= 80 | urbain
```

```
[1] TRUE FALSE FALSE TRUE TRUE TRUE
```

Une remarque importante : quand l'un des termes d'une condition comporte une valeur manquante (`NA`), le résultat de cette condition n'est pas toujours `TRUE` ou `FALSE`, il peut aussi être à son tour une valeur manquante.

```
R> taille
```

```
[1] 1.88 NA 1.65 1.92 1.76 NA
```

```
R> taille > 1.8
```

```
[1] TRUE NA FALSE TRUE FALSE NA
```

On voit que le test `NA > 1.8` ne renvoie ni vrai ni faux, mais `NA`.

Une autre conséquence importante de ce comportement est qu'on ne peut pas utiliser l'opérateur l'expression `== NA` pour tester la présence de valeurs manquantes. On utilisera à la place la fonction *ad hoc* `is.na` :

```
R> is.na(taille > 1.8)
```

```
[1] FALSE TRUE FALSE FALSE FALSE TRUE
```

Pour compliquer encore un peu le tout, lorsqu'on utilise une condition pour l'indexation, si la condition renvoie `NA`, R ne sélectionne pas l'élément mais retourne quand même la valeur `NA`. Ceci a donc des conséquences sur le résultat d'une indexation par comparaison.

Par exemple si je cherche à connaître le poids des personnes mesurant 1,80 mètre ou plus :

```
R> taille
```

```
[1] 1.88 NA 1.65 1.92 1.76 NA
```

```
R> poids
```

```
[1] 80 63 75 87 82 67
```

```
R> poids[taille > 1.8]
```

```
[1] 80 NA 87 NA
```

Les éléments pour lesquels la taille n'est pas connue ont été transformés en `NA`, ce qui n'influera pas le calcul d'une moyenne. Par contre, lorsqu'on utilisera assignation et indexation ensemble, cela peut créer des problèmes. Il est donc préférable lorsqu'on a des valeurs manquantes de les exclure ainsi :

```
R> poids[taille > 1.8 & !is.na(taille)]
```

```
[1] 80 87
```

Pour plus de détails sur les conditions et le calcul logique dans R, on pourra se référer au chapitre dédié, page 925.

## Assignment par indexation

Dans tous les exemples précédents, on a utilisé l'indexation pour extraire une partie d'un vecteur, en plaçant l'opération d'indexation à droite de l'opérateur `<-`.

Mais l'indexation peut également être placée à gauche de cet opérateur d'assignation. Dans ce cas, les éléments sélectionnés par l'indexation sont alors remplacés par les valeurs indiquées à droite de l'opérateur `<-`.

Prenons donc un exemple simple :

```
R> v <- 1:5
v
```

```
[1] 1 2 3 4 5
```

```
R> v[1] <- 3
v
```

```
[1] 3 2 3 4 5
```

Cette fois, au lieu d'utiliser quelque chose comme `x <- v[1]`, qui aurait placé la valeur du premier élément de `v` dans `x`, on a utilisé `v[1] <- 3`, ce qui a mis à jour le premier élément de `v` avec la valeur 3. Ceci fonctionne également pour les différents types d'indexation évoqués précédemment :

```
R> sexe["Alex"] <- "f"
```

Enfin on peut modifier plusieurs éléments d'un seul coup soit en fournissant un vecteur, soit en profitant du mécanisme de recyclage. Les deux commandes suivantes sont ainsi rigoureusement équivalentes :

```
R> sexe[c(1, 3, 4)] <- c("Homme", "Homme", "Homme")
sexe[c(1, 3, 4)] <- "Homme"
```

L'assignation par indexation peut aussi être utilisée pour ajouter une ou plusieurs valeurs à un vecteur :

```
R> length(sexe)
```

```
[1] 6
```

```
R> sexe[7] <- "f"
sexe
```

Michael	Anna	Dom	John	Alex	Mary
"Homme"	"f"	"Homme"	"Homme"	"f"	"f"

```
R> length(sexe)
```

```
[1] 7
```

On commence à voir comment l’utilisation de l’indexation par conditions et de l’assignation va nous permettre de faire des recodages (que nous aborderons plus en détail dans un chapitre dédié, page 197).

## En résumé

- Un vecteur est un objet unidimensionnel contenant une liste de valeurs qui sont toutes du même type (entières, numériques, textuelles ou logiques).
- La fonction `class` permet de connaître le type du vecteur et la fonction `length` sa longueur, c'est-à-dire son nombre d'éléments.
- La fonction `c` sert à créer et à combiner des vecteurs.
- Les valeurs manquantes sont représentées avec `NA`.
- Un vecteur peut être nommé, c'est-à-dire qu'un nom textuel a été associé à chaque élément. Cela peut se faire lors de sa création ou avec la fonction `names`.
- L’indexation consiste à extraire certains éléments d’un vecteur. Pour cela, on indique ce qu’on souhaite extraire entre crochets (`[]`) juste après le nom du vecteur. Le type d’indexation dépend du type d’information transmise.
- S’il s’agit de nombres entiers, c’est l’indexation par position : les nombres représentent la position dans le vecteur des éléments qu’on souhaite extraire. Un nombre négatif s’interprète comme «tous les éléments sauf celui-là».
- Si on indique des chaînes de caractères, c’est l’indexation par nom : on indique le nom des éléments qu’on souhaite extraire. Cette forme d’indexation ne fonctionne que si le vecteur est nommé.
- Si on transmet des valeurs logiques, le plus souvent sous la forme d’une condition, c’est l’indexation par condition : `TRUE` indique les éléments à extraire et `FALSE` les éléments à exclure. Il faut être vigilant aux valeurs manquantes (`NA`) dans ce cas précis.
- Enfin, il est possible de modifier que certains éléments d’un vecteur en ayant recours à la fois à l’indexation (`[]`) et à l’assignation (`<-`).

# Listes et Tableaux de données

Listes .....	87
Propriétés et création .....	88
Indexation .....	90
Tableaux de données .....	94
Propriétés et création .....	94
Indexation .....	96
Afficher les données .....	99
En résumé .....	115

## NOTE

Il est préférable d'avoir déjà lu le chapitre Vecteurs, indexation et assignation, page 67 avant d'aborder celui-ci.

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #02 (les bases du langage R) sur [YouTube](#).

## Listes

Par nature, les vecteurs ne peuvent contenir que des valeurs de même type (numérique, textuel ou logique). Or, on peut avoir besoin de représenter des objets plus complexes composés d'éléments disparates. C'est ce que permettent les listes.

## Propriétés et création

Une liste se crée tout simplement avec la fonction `list` :

```
R> l1 <- list(1:5, "abc")  
l1
```

```
[[1]]  
[1] 1 2 3 4 5
```

```
[[2]]  
[1] "abc"
```

Une liste est un ensemble d’objets, quels qu’ils soient, chaque élément d’une liste pouvant avoir ses propres dimensions. Dans notre exemple précédent, nous avons créé une liste `l1` composée de deux éléments : un vecteur d’entiers de longueur 5 et un vecteur textuel de longueur 1. La longueur d’une liste correspond aux nombres d’éléments qu’elle contient et s’obtient avec `length` :

```
R> length(l1)
```

```
[1] 2
```

Comme les vecteurs, une liste peut être nommée et les noms des éléments d’une liste sont accessibles avec `names` :

```
R> l2 <- list(minuscules = letters, majuscules = LETTERS, mois = month.name)  
l2
```

```
$minuscules  
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"  
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
$majuscules  
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"  
[15] "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

```
$mois  
[1] "January"    "February"   "March"       "April"  
[5] "May"         "June"        "July"        "August"  
[9] "September"   "October"    "November"   "December"
```

```
R> length(l2)
```

```
[1] 3
```

```
R> names(l2)
```

```
[1] "minuscules" "majuscules" "mois"
```

Que se passe-t-il maintenant si on effectue la commande suivante ?

```
R> l <- list(l1, l2)
```

À votre avis, quelle est la longueur de cette nouvelle liste `l` ? 5 ?

```
R> length(l)
```

```
[1] 2
```

Eh bien non ! Elle est de longueur 2 car nous avons créé une liste composée de deux éléments qui sont eux-mêmes des listes. Cela est plus lisible si on fait appel à la fonction `str` qui permet de visualiser la structure d'un objet.

```
R> str(l)
```

```
List of 2
$ :List of 2
..$ : int [1:5] 1 2 3 4 5
..$ : chr "abc"
$ :List of 3
..$ minuscules: chr [1:26] "a" "b" "c" "d" ...
..$ majuscules: chr [1:26] "A" "B" "C" "D" ...
..$ mois       : chr [1:12] "January" "February" "March" "April" ...
```

Une liste peut contenir tous types d'objets, y compris d'autres listes. Pour combiner les éléments d'une liste, il faut utiliser la fonction `append` :

```
R> l <- append(l1, l2)
length(l)
```

```
[1] 5
```

```
R> str(l)
```

```
List of 5
$ : int [1:5] 1 2 3 4 5
$ : chr "abc"
$ minuscules: chr [1:26] "a" "b" "c" "d" ...
$ majuscules: chr [1:26] "A" "B" "C" "D" ...
$ mois : chr [1:12] "January" "February" "March" "April" ...
```

On peut noter en passant qu’une liste peut tout à fait n’être que partiellement nommée.

## Indexation

Les crochets simples ( `[]` ) fonctionnent comme pour les vecteurs. On peut utiliser à la fois l’indexation par position, l’indexation par nom et l’indexation par condition.

```
R> l
```

```
[[1]]
[1] 1 2 3 4 5

[[2]]
[1] "abc"

$minuscules
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"

$majuscules
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"
[15] "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"

$mois
[1] "January"   "February"   "March"      "April"
[5] "May"        "June"        "July"       "August"
[9] "September"  "October"    "November"   "December"
```

```
R> l[c(1, 3, 4)]
```

```
[[1]]
[1] 1 2 3 4 5

$minuscules
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"

$majuscules
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"
[15] "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

```
R> l[c("majuscules", "minuscules")]
```

```
$majuscules
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"
[15] "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"

$minuscules
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
R> l[c(TRUE, TRUE, FALSE, FALSE, TRUE)]
```

```
[[1]]
[1] 1 2 3 4 5

[[2]]
[1] "abc"

$mois
[1] "January"   "February"   "March"      "April"
[5] "May"        "June"       "July"       "August"
[9] "September" "October"    "November"   "December"
```

Même si on extrait un seul élément, l'extraction obtenue avec les crochets simples renvoie toujours une liste, ici composée d'un seul élément :

```
R> str(l[1])
```

```
List of 1  
$ : int [1:5] 1 2 3 4 5
```

Supposons que je souhaite calculer la moyenne des valeurs du premier élément de ma liste. Essayons la commande suivante :

```
R> mean(l[1])
```

```
Warning in mean.default(l[1]): argument is not numeric or  
logical: returning NA
```

```
[1] NA
```

Nous obtenons un message d’erreur. En effet, R ne sait pas calculer une moyenne à partir d’une liste. Ce qu’il lui faut, c’est un vecteur de valeurs numériques. Autrement dit, ce que nous cherchons à obtenir c’est le contenu même du premier élément de notre liste et non une liste à un seul élément.

C’est ici que les doubles crochets ( [ [] ] ) vont rentrer en jeu. Pour ces derniers, nous pourrons utiliser l’indexation par position ou l’indexation par nom, mais pas l’indexation par condition. De plus, le critère qu’on indiquera doit indiquer *un et un seul* élément de notre liste. Au lieu de renvoyer une liste à un élément, les doubles crochets vont renvoyer l’élément désigné. Vite, un exemple :

```
R> str(l[1])
```

```
List of 1  
$ : int [1:5] 1 2 3 4 5
```

```
R> str(l[[1]])
```

```
int [1:5] 1 2 3 4 5
```

Maintenant, nous pouvons calculer notre moyenne :

```
R> mean(l[[1]])
```

```
[1] 3
```

Nous pouvons aussi tester l'indexation par nom.

```
R> l[["mois"]]
```

```
[1] "January"   "February"  "March"      "April"
[5] "May"        "June"       "July"       "August"
[9] "September" "October"   "November"  "December"
```

Mais il faut avouer que cette écriture avec doubles crochets et guillemets est un peu lourde. Heureusement, un nouvel acteur entre en scène : le symbole dollar ( \$ ). C'est un raccourci des doubles crochets pour l'indexation par nom qu'on utilise ainsi :

```
R> l$mois
```

```
[1] "January"   "February"  "March"      "April"
[5] "May"        "June"       "July"       "August"
[9] "September" "October"   "November"  "December"
```

Les écritures `l$mois` et `l[["mois"]]` sont équivalentes. Attention ! Cela ne fonctionne que pour l'indexation par nom.

```
R> l$1
```

```
Error: unexpected numeric constant in "l$1"
```

L'assignation par indexation fonctionne également avec les doubles crochets ou le signe dollar :

```
R> l[[2]] <- list(c("un", "vecteur", "textuel"))
l$mois <- c("Janvier", "Février", "Mars")
l
```

```
[[1]]
[1] 1 2 3 4 5

[[2]]
[[2]][[1]]
[1] "un"      "vecteur" "textuel"

$minuscules
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
```

```
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"  
  
$majuscules  
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"  
[15] "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"  
  
$mois  
[1] "Janvier" "Février" "Mars"
```

## Tableaux de données

Il y a un type d'objets que nous avons déjà abordé dans le chapitre Premier travail avec les données, page 33, il s'agit du tableau de données ou *data frame* en anglais.

### Propriétés et création

Dans R, les tableaux de données sont tout simplement des listes avec quelques propriétés spécifiques :

- les tableaux de données ne peuvent contenir que des vecteurs ;
- tous les vecteurs d'un tableau de données ont la même longueur ;
- tous les éléments d'un tableau de données sont nommés et ont chacun un nom unique.

Dès lors, un tableau de données correspond aux fichiers de données qu'on a l'habitude de manipuler dans d'autres logiciels de statistiques comme **SPSS** ou **Stata**. Les variables sont organisées en colonnes et les observations en lignes.

On peut créer un tableau de données avec la fonction `data.frame` :

```
R> df <- data.frame(  
  sexe = c("f", "f", "h", "h"),  
  age = c(52, 31, 29, 35),  
  blond = c(FALSE, TRUE, TRUE, FALSE)  
)  
df
```

```
R> str(df)
```

```
'data.frame': 4 obs. of 3 variables:  
 $ sexe : chr  "f" "f" "h" "h"  
 $ age  : num  52 31 29 35
```

```
$ blond: logi FALSE TRUE TRUE FALSE
```

La fonction `data.frame` a un gros défaut : si on ne désactive pas l'option `stringsAsFactors` elle transforme les chaînes de caractères, ici la variable `sex` en facteurs (un type de vecteur que nous aborderons plus en détail dans un prochain chapitre, page 117).

```
R> df <- data.frame(
  sexe = c("f", "f", "h", "h"),
  age = c(52, 31, 29, 35),
  blond = c(FALSE, TRUE, TRUE, FALSE),
  stringsAsFactors = FALSE
)
df
```

```
R> str(df)
```

```
'data.frame': 4 obs. of 3 variables:
 $ sexe : chr "f" "f" "h" "h"
 $ age  : num 52 31 29 35
 $ blond: logi FALSE TRUE TRUE FALSE
```

Un tableau de données étant une liste, la fonction `length` renverra le nombre d'éléments de la liste, donc dans le cas présent le nombre de variables, et `names` leurs noms :

```
R> length(df)
```

```
[1] 3
```

```
R> names(df)
```

```
[1] "sexe"  "age"   "blond"
```

Comme tous les éléments d'un tableau de données ont la même longueur, cet objet peut être vu comme bidimensionnel. Les fonctions `nrow`, `ncol` et `dim` donnent respectivement le nombre de lignes, le nombre de colonnes et les dimensions de notre tableau.

```
R> nrow(df)
```

```
[1] 4
```

```
R> ncol(df)
```

```
[1] 3
```

```
R> dim(df)
```

```
[1] 4 3
```

De plus, tout comme les colonnes ont un nom, il est aussi possible de nommer les lignes avec `row.names` :

```
R> row.names(df) <- c("Anna", "Mary-Ann", "Michael", "John")  
df
```

## Indexation

Les tableaux de données étant des listes, nous pouvons donc utiliser les crochets simples ( `[]` ), les crochets doubles ( `[][]` ) et le symbole dollar ( `$` ) pour extraire des parties de notre tableau, de la même manière que pour n'importe quelle liste.

```
R> df[1]
```

```
R> df[[1]]
```

```
[1] "f" "f" "h" "h"
```

```
R> df$sexe
```

```
[1] "f" "f" "h" "h"
```

Cependant, un tableau de données étant un objet bidimensionnel, il est également possible d'extraire des données sur deux dimensions, à savoir un premier critère portant sur les lignes et un second portant sur les colonnes. Pour cela, nous utiliserons les crochets simples ( `[]` ) en séparant nos deux critères par une virgule ( `,` ).

Un premier exemple :

```
R> df
```

```
R> df[3, 2]
```

```
[1] 29
```

Cette première commande indique que nous souhaitons la troisième ligne de la seconde colonne, autrement dit l'âge de Michael. Le même résultat peut être obtenu avec l'indexation par nom, l'indexation par condition, ou un mélange de tout ça.

```
R> df["Michael", "age"]
```

```
[1] 29
```

```
R> df[c(F, F, T, F), c(c(F, T, F))]
```

```
[1] 29
```

```
R> df[3, "age"]
```

```
[1] 29
```

```
R> df["Michael", 2]
```

```
[1] 29
```

Il est également possible de préciser un seul critère. Par exemple, si je souhaite les deux premières observations, ou les variables *sex*e et *blond*:

```
R> df[1:2, ]
```

```
R> df[, c("sex", "blond")]
```

Il a suffi de laisser un espace vide avant ou après la virgule. ATTENTION ! Il est cependant impératif de laisser la virgule pour indiquer à R qu'on souhaite effectuer une indexation à deux dimensions. Si on oublie la virgule, cela nous ramène au mode de fonctionnement des listes. Et le résultat n'est pas forcément le

même :

```
R> df[2, ]
```

```
R> df[, 2]
```

```
[1] 52 31 29 35
```

```
R> df[2]
```

**NOTE**

Au passage, on pourra noter quelques subtilités sur le résultat renvoyé.

```
R> str(df[2, ])
```

```
'data.frame': 1 obs. of 3 variables:
 $ sexe : chr "f"
 $ age  : num 31
 $ blond: logi TRUE
```

```
R> str(df[, 2])
```

```
num [1:4] 52 31 29 35
```

```
R> str(df[2])
```

```
'data.frame': 4 obs. of 1 variable:
 $ age: num 52 31 29 35
```

```
R> str(df[[2]])
```

```
num [1:4] 52 31 29 35
```

`df[2, ]` signifie qu'on veut toutes les variables pour le second individu. Le résultat est un tableau de données à une ligne et trois colonnes. `df[2]` correspond au mode d'extraction des listes et renvoie donc une liste à un élément, en l'occurrence un tableau de données à quatre observations et une variable. `df[[2]]` quant à lui renvoie le contenu de cette variable, soit un vecteur numérique de longueur quatre. Reste `df[, 2]` qui renvoie toutes les observations pour la seconde colonne. Or l'indexation bidimensionnelle a un fonctionnement un peu particulier : par défaut elle renvoie un tableau de données mais s'il y a une seule variable dans l'extraction, c'est un vecteur qui est renvoyé. Pour plus de détails, on pourra consulter l'entrée d'aide de `[.data.frame]`.

## Afficher les données

Prenons un tableau de données un peu plus conséquent, en l'occurrence un jeu de données disponible

dans l’extension **questionr** et correspondant à un extrait de l’enquête *Histoire de vie* réalisée par l’INSEE en 2003. Il contient 2000 individus et 20 variables.

```
R> library(questionr)
  data(hdv2003)
  d <- hdv2003
```

Si on demande d’afficher l’objet `d` dans la console (résultat non reproduit ici), R va afficher l’ensemble du contenu de `d` à l’écran ce qui, sur un tableau de cette taille, ne sera pas très lisible. Pour une exploration visuelle, le plus simple est souvent d’utiliser la visionneuse intégrée à **RStudio** et qu’on peut appeler avec la fonction **View**.

```
R> View(d)
```

Les fonctions `head` et `tail`, qui marchent également sur les vecteurs, permettent d’afficher seulement les premières (respectivement les dernières) lignes d’un tableau de données :

```
R> head(d)
```

```
R> tail(d, 2)
```

L’extension **dplyr**, que nous n’aborderons en détails que plus tard, page 233, propose une fonction `glimpse` (ce qui signifie «aperçu» en anglais) qui permet de visualiser rapidement et de manière condensée le contenu d’un tableau de données.

```
R> library(dplyr)
  glimpse(d)
```

```
Rows: 2,000
Columns: 20
$ id              <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11...
$ age             <int> 28, 23, 59, 34, 71, 35, 60, 47, 2...
$ sexe            <fct> Femme, Femme, Homme, Homme, Femme...
$ nivetud         <fct> "Enseignement superieur y compris...
$ poids            <dbl> 2634.3982, 9738.3958, 3994.1025, ...
$ occup            <fct> "Exerce une profession", "Etudian...
$ qualif           <fct> Employe, NA, Technicien, Technici...
$ freres.soeurs   <int> 8, 2, 2, 1, 0, 5, 1, 5, 4, 2, 3, ...
$ clso             <fct> Oui, Oui, Non, Non, Oui, Non, Oui...
$ relig            <fct> Ni croyance ni appartenance, Ni c...
$ trav.imp          <fct> Peu important, NA, Aussi importan...
$ trav.satisf      <fct> Insatisfaction, NA, Equilibre, Sa...
$ hard.rock        <fct> Non, Non, Non, Non, Non, Non...
```

```
$ lecture.bd      <fct> Non, Non, Non, Non, Non, Non, Non...
$ peche.chasse   <fct> Non, Non, Non, Non, Non, Oui...
$ cuisine         <fct> Oui, Non, Non, Oui, Non, Non, Oui...
$ bricol          <fct> Non, Non, Non, Oui, Non, Non, Non...
$ cinema          <fct> Non, Oui, Non, Oui, Non, Oui, Non...
$ sport           <fct> Non, Oui, Oui, Oui, Non, Oui, Non...
$ heures.tv       <dbl> 0.0, 1.0, 0.0, 2.0, 3.0, 2.0, 2.9...
```

L'extension **labelled** propose une fonction `look_for` qui permet de lister les différentes variables d'un fichier de données :

```
R> library(labelled)
look_for(d)
```

pos	variable	label	col_type	value
s				
<chr>	<chr>	<chr>	<chr>	<ch
r>				
1	id	–	int	range: 1 – 200
0				
2	age	–	int	range: 18 – 9
7				
3	sexe	–	fct	Homm
e				
e				Femm
4	nivetud	–	fct	N'a jamais fait d'etude
s				
d'etudes primaires				A arrete ses etudes, avant la dernière année
				Dernière année d'études primaire
s				
1er cycl				
e				
2eme cycl				
e				
t				Enseignement technique ou professionnel cour
te				
ng				Enseignement technique ou professionnel lon
perieur				
poids	–	dbl	range: 78.0783403 – 31092.1413	Enseignement supérieur y compris technique su
2				
occup	–	fct	Exerce une professio	

```
n                               Chomeu
r                               Etudiant, elev
e                               Retrait
e                               Retire des affaire
s                               Au foye
r                               Autre inacti
f                               qualif      -      fct      Ouvrier specialis
7                               Ouvrier qualifi
e                               Technicie
n                               Profession intermediair
e                               Cadr
e                               Employ
e                               Autr
e
8                               freres.soeurs -      int      range: 0 - 2
2
9                               calso      -      fct      Ou
i                               No
n                               Ne sait pa
s
10                              relig      -      fct      Pratiquant regulie
r                               Pratiquant occasionne
l                               Appartenance sans pratiqu
e                               Ni croyance ni appartenanc
e
t                               Reje
NSP ou NVP
R
```

11	trav.imp	-	fct	Le plus importan
t				Aussi important que le rest
e				Moins important que le rest
e				Peu importan
t				
12	trav.satisf	-	fct	Satisfactio
n				Insatisfactio
n				Equilibr
e				
13	hard.rock	-	fct	No
n				Ou
i				
14	lecture.bd	-	fct	No
n				Ou
i				
15	peche.chasse	-	fct	No
n				Ou
i				
16	cuisine	-	fct	No
n				Ou
i				
17	bricol	-	fct	No
n				Ou
i				
18	cinema	-	fct	No
n				Ou
i				
19	sport	-	fct	No
n				Ou
i				
20	heures.tv	-	dbl	range: 0 - 12

Lorsqu'on a un gros tableau de données avec de nombreuses variables, il peut être difficile de retrouver la ou les variables d'intérêt. Il est possible d'indiquer à `look_for` un mot-clé pour limiter la recherche. Par exemple :

```
R> look_for(d, "trav")
```

pos	variable	label	col_type	values
11	trav.imp	-	fct	Le plus important Aussi important que le reste Moins important que le reste Peu important
12	trav.satisf	-	fct	Satisfaction Insatisfaction Equilibre

Il est à noter que si la recherche n'est pas sensible à la casse (i.e. aux majuscules et aux minuscules), elle est sensible aux accents.

La méthode `summary` qui fonctionne sur tout type d'objet permet d'avoir quelques statistiques de base sur les différentes variables de notre tableau, les statistiques affichées dépendant du type de variable.

```
R> summary(d)
```

<i>id</i>	<i>age</i>	<i>sexe</i>
Min. : 1.0	Min. :18.00	Homme: 899
1st Qu.: 500.8	1st Qu.:35.00	Femme:1101
Median :1000.5	Median :48.00	
Mean :1000.5	Mean :48.16	
3rd Qu.:1500.2	3rd Qu.:60.00	
Max. :2000.0	Max. :97.00	
		<i>nivetud</i>
Enseignement technique ou professionnel court		:463
Enseignement superieur y compris technique superieur		:441
Derniere annee d'etudes primaires		:341
1er cycle		:204
2eme cycle		:183
(Other)		:256
NA's		:112
<i>poids</i>		<i>occup</i>
Min. : 78.08	Exerce une profession:	1049
1st Qu.: 2221.82	Chomeur	: 134
Median : 4631.19	Etudiant, eleve	: 94
Mean : 5535.61	Retraite	: 392
3rd Qu.: 7626.53	Retire des affaires	: 77
Max. :31092.14	Au foyer	: 171
	Autre inactif	: 83

```

                    qualif    freres.soeurs
Employe           :594     Min.   : 0.000
Ouvrier qualifie :292     1st Qu.: 1.000
Cadre             :260     Median : 2.000
Ouvrier specialise :203     Mean   : 3.283
Profession intermediaire:160     3rd Qu.: 5.000
(Other)           :144     Max.   :22.000
NA's              :347

                    cuso                  relig
Oui      : 936  Pratiquant regulier   :266
Non     :1037  Pratiquant occasionnel :442
Ne sait pas: 27  Appartenance sans pratique :760
                  Ni croyance ni appartenance:399
                  Rejet                 : 93
                  NSP ou NVPR            : 40

                    trav.imp          trav.satisf
Le plus important      : 29  Satisfaction :480
Aussi important que le reste:259  Insatisfaction:117
Moins important que le reste:708  Equilibre    :451
Peu important          : 52  NA's       :952
NA's                  :952

hard.rock  lecture.bd peche.chasse cuisine    bricol
Non:1986   Non:1953   Non:1776    Non:1119   Non:1147
Oui: 14    Oui: 47    Oui: 224   Oui: 881   Oui: 853

                    cinema    sport      heures.tv
Non:1174   Non:1277   Min.   : 0.000
Oui: 826   Oui: 723   1st Qu.: 1.000
                  Median : 2.000
                  Mean   : 2.247
                  3rd Qu.: 3.000
                  Max.   :12.000
                  NA's   :5

```

On peut également appliquer `summary` à une variable particulière.

```
R> summary(d$sex)
```

```
Homme Femme  
899 1101
```

```
R> summary(d$age)
```

```
Min. 1st Qu. Median Mean 3rd Qu. Max.  
18.00 35.00 48.00 48.16 60.00 97.00
```

L'extension **questionr** fournit également une fonction bien pratique pour décrire les différentes variables d'un tableau de données. Il s'agit de **describe**. Faisons de suite un essai :

```
R> describe(d)
```

```
d
```

```
20 Variables 2000 Observations
```

```
-----  
id
```

	n	missing	distinct	Info	Mean	Gmd
2000	0	2000	1	1000	667	
.05	.10	.25	.50	.75	.90	
101.0	200.9	500.8	1000.5	1500.2	1800.1	
.95						
1900.0						

```
lowest : 1 2 3 4 5, highest: 1996 1997 1998 1999 2000
```

```
-----  
age
```

	n	missing	distinct	Info	Mean	Gmd
2000	0	78	1	48.16	19.4	
.05	.10	.25	.50	.75	.90	
22	26	35	48	60	72	
.95						
77						

```
lowest : 18 19 20 21 22, highest: 91 92 93 96 97
```

```
-----  
sex
```

	n	missing	distinct
2000	0	2	

Value	Homme	Femme
Frequency	899	1101
Proportion	0.45	0.55

---

nivetud

n	missing	distinct
1888	112	8

lowest : N'a jamais fait d'etudes A arret  
e ses etudes, avant la dernière année d'études primaires Dernière année d'étude  
s primaires 1er cycl

e 2eme cycl

e

highest: 1er cycle 2eme cy  
cle Enseignement technique  
ou professionnel court Enseignement technique ou professionne  
l long Enseignement supérieur y compris technique superieu  
r

---

poids

n	missing	distinct	Info	Mean	Gmd
2000	0	1877	1	5536	4553
.05	.10	.25	.50	.75	.90
799.8	1161.7	2221.8	4631.2	7626.5	10819.0
.95					
13647.9					

lowest : 78.07834 92.68033 92.93637 127.90122 153.01026  
highest: 27195.83782 29548.78954 29570.78696 29657.94035 31092.14132

---

occup

n	missing	distinct
2000	0	7

lowest : Exerce une profession Chomeur Etudiant, élève Retraite  
ite Retire des affaires

highest: Etudiant, élève Retraite Retire des affaires Au fo  
yer Autre inactif

Exerce une profession (1049, 0.524), Chomeur (134, 0.067),  
Etudiant, élève (94, 0.047), Retraite (392, 0.196), Retire  
des affaires (77, 0.038), Au foyer (171, 0.086), Autre  
inactif (83, 0.042)

---

qualif

n	missing	distinct
---	---------	----------

1653 347 7

```
lowest : Ouvrier specialise      Ouvrier qualifie      Technicie
n                  Profession intermediaire Cadre
highest: Technicien            Profession intermediaire Cadr
e                   Employe           Autre
```

Ouvrier specialise (203, 0.123), Ouvrier qualifie (292, 0.177), Technicien (86, 0.052), Profession intermediaire (160, 0.097), Cadre (260, 0.157), Employe (594, 0.359), Autre (58, 0.035)

---

freres.soeurs

n	missing	distinct	Info	Mean	Gmd
2000	0	19	0.977	3.283	2.87
.05	.10	.25	.50	.75	.90
0	1	1	2	5	7
.95					
9					

lowest : 0 1 2 3 4, highest: 14 15 16 18 22

0 (167, 0.084), 1 (407, 0.203), 2 (427, 0.214), 3 (284, 0.142), 4 (210, 0.105), 5 (151, 0.076), 6 (99, 0.050), 7 (94, 0.047), 8 (52, 0.026), 9 (37, 0.018), 10 (21, 0.011), 11 (21, 0.011), 12 (8, 0.004), 13 (10, 0.005), 14 (4, 0.002), 15 (4, 0.002), 16 (1, 0.000), 18 (2, 0.001), 22 (1, 0.000)

---

also

n	missing	distinct
2000	0	3

Value	Oui	Non	Ne sait pas
Frequency	936	1037	27
Proportion	0.468	0.518	0.014

---

relig

n	missing	distinct
2000	0	6

```
lowest : Pratiquant regulier      Pratiquant occasionnel      Appartenance sa
ns pratique Ni croyance ni appartenance Rejet
highest: Pratiquant occasionnel    Appartenance sans pratique Ni croyance ni
appartenance Rejet                 NSP ou NVPR
```

Pratiquant regulier (266, 0.133), Pratiquant occasionnel

(442, 0.221), Appartenance sans pratique (760, 0.380), Ni croyance ni appartenance (399, 0.200), Rejet (93, 0.046), NSP ou NVPR (40, 0.020)

## trav.imp

n	missing	distinct
1048	952	4

Le plus important (29, 0.028), Aussi important que le reste (259, 0.247), Moins important que le reste (708, 0.676), Peu important (52, 0.050)

## trav.satisf

n	missing	distinct
1048	952	3

Value	Satisfaction	Insatisfaction	Equilibre
Frequency	480	117	451
Proportion	0.458	0.112	0.430

## hard.rock

n	missing	distinct
2000	0	2

Value	Non	Oui
Frequency	1986	14
Proportion	0.993	0.007

## lecture.bd

n	missing	distinct
2000	0	2

Value	Non	Oui
Frequency	1953	47
Proportion	0.977	0.024

## peche.chasse

n	missing	distinct
2000	0	2

Value	Non	Oui
Frequency	1776	224
Proportion	0.888	0.112

## cuisine

n	missing	distinct
2000	0	2

```

Value      Non   Oui
Frequency  1119   881
Proportion 0.559  0.440
-----
bricol
    n  missing distinct
    2000       0        2

Value      Non   Oui
Frequency  1147   853
Proportion 0.574  0.426
-----
cinema
    n  missing distinct
    2000       0        2

Value      Non   Oui
Frequency  1174   826
Proportion 0.587  0.413
-----
sport
    n  missing distinct
    2000       0        2

Value      Non   Oui
Frequency  1277   723
Proportion 0.638  0.362
-----
heures.tv
    n  missing distinct      Info      Mean      Gmd
    1995       5       29    0.972    2.247    1.877
    .05       .10      .25     .50      .75      .90
    0         0       1       2       3       4
    .95
    5

lowest :  0.0  0.1  0.2  0.3  0.4, highest:  8.0  9.0 10.0 11.0 12.0
-----
```

Comme on le voit sur cet exemple, `describe` nous affiche le type des variables, les premières valeurs de chacune, le nombre de valeurs manquantes, le nombre de valeurs différentes (uniques) ainsi que quelques autres informations suivant le type de variables.

Il est possible de restreindre l'affichage à seulement quelques variables en indiquant le nom de ces dernières.

```
R> describe(d, c("age", "trav.satisf"))
```

age trav.satisf

20 Variables 2000 Observations

id

	n	missing	distinct	Info	Mean	Gmd
2000	0	2000	1	1000	667	
.05	.10	.25	.50	.75	.90	
101.0	200.9	500.8	1000.5	1500.2	1800.1	
.95						
1900.0						

lowest : 1 2 3 4 5, highest: 1996 1997 1998 1999 2000

age

	n	missing	distinct	Info	Mean	Gmd
2000	0	78	1	48.16	19.4	
.05	.10	.25	.50	.75	.90	
22	26	35	48	60	72	
.95						
77						

lowest : 18 19 20 21 22, highest: 91 92 93 96 97

sexe

	n	missing	distinct
2000	0	2	

Value Homme Femme

Frequency 899 1101

Proportion 0.45 0.55

nivetud

	n	missing	distinct
1888	112	8	

lowest : N'a jamais fait d'etudes

A arret

e ses etudes, avant la derniere annee d'etudes primaires Derniere annee d'etude  
s primaires 1er cycl

2eme cycl

e

e

highest: 1er cycle

2eme cy

cle

Enseignement technique

ou professionnel court

Enseignement technique ou professionne

```

l long                                Enseignement superieur y compris technique superieu
r
-----
poids
    n   missing  distinct      Info      Mean      Gmd
  2000        0     1877        1     5536     4553
    .05       .10     .25       .50     .75     .90
  799.8    1161.7   2221.8    4631.2   7626.5  10819.0
    .95
13647.9

lowest :  78.07834   92.68033   92.93637  127.90122  153.01026
highest: 27195.83782 29548.78954 29570.78696 29657.94035 31092.14132
-----
occup
    n   missing  distinct
  2000        0     7

lowest : Exerce une profession Chomeur          Etudiant, eleve      Retra
ite           Retire des affaires
highest: Etudiant, eleve          Retraite          Retire des affaires  Au fo
yer           Autre inactif

Exerce une profession (1049, 0.524), Chomeur (134, 0.067),
Etudiant, eleve (94, 0.047), Retraite (392, 0.196), Retire
des affaires (77, 0.038), Au foyer (171, 0.086), Autre
inactif (83, 0.042)
-----
qualif
    n   missing  distinct
  1653      347     7

lowest : Ouvrier specialise      Ouvrier qualifie      Technicie
n           Profession intermediaire Cadre
highest: Technicien            Profession intermediaire Cadr
e           Employe             Autre

Ouvrier specialise (203, 0.123), Ouvrier qualifie (292,
0.177), Technicien (86, 0.052), Profession intermediaire
(160, 0.097), Cadre (260, 0.157), Employe (594, 0.359),
Autre (58, 0.035)
-----
freres.soeurs
    n   missing  distinct      Info      Mean      Gmd
  2000        0     19     0.977     3.283     2.87
    .05       .10     .25       .50     .75     .90
    0         1       1       2       5       7

```

```

.95
9

lowest : 0 1 2 3 4, highest: 14 15 16 18 22

0 (167, 0.084), 1 (407, 0.203), 2 (427, 0.214), 3 (284,
0.142), 4 (210, 0.105), 5 (151, 0.076), 6 (99, 0.050), 7
(94, 0.047), 8 (52, 0.026), 9 (37, 0.018), 10 (21, 0.011),
11 (21, 0.011), 12 (8, 0.004), 13 (10, 0.005), 14 (4,
0.002), 15 (4, 0.002), 16 (1, 0.000), 18 (2, 0.001), 22 (1,
0.000)
-----
calso
    n missing distinct
    2000      0      3

Value          Oui        Non Ne sait pas
Frequency      936       1037      27
Proportion     0.468     0.518     0.014
-----
relig
    n missing distinct
    2000      0      6

lowest : Pratiquant regulier      Pratiquant occasionnel      Appartenance sa
ns pratique Ni croyance ni appartenance Rejet
highest: Pratiquant occasionnel   Appartenance sans pratique Ni croyance ni
appartenance Rejet                  NSP ou NVPR

Pratiquant regulier (266, 0.133), Pratiquant occasionnel
(442, 0.221), Appartenance sans pratique (760, 0.380), Ni
croyance ni appartenance (399, 0.200), Rejet (93, 0.046),
NSP ou NVPR (40, 0.020)
-----
trav.imp
    n missing distinct
    1048     952      4

Le plus important (29, 0.028), Aussi important que le reste
(259, 0.247), Moins important que le reste (708, 0.676),
Peu important (52, 0.050)
-----
trav.satisf
    n missing distinct
    1048     952      3

Value      Satisfaction Insatisfaction      Equilibre

```

```
Frequency          480           117           451
Proportion        0.458         0.112         0.430
-----
hard.rock
  n  missing distinct
  2000      0       2

Value      Non   Oui
Frequency  1986   14
Proportion 0.993  0.007
-----
lecture.bd
  n  missing distinct
  2000      0       2

Value      Non   Oui
Frequency  1953   47
Proportion 0.977  0.024
-----
peche.chasse
  n  missing distinct
  2000      0       2

Value      Non   Oui
Frequency  1776  224
Proportion 0.888  0.112
-----
cuisine
  n  missing distinct
  2000      0       2

Value      Non   Oui
Frequency  1119  881
Proportion 0.559  0.440
-----
bricol
  n  missing distinct
  2000      0       2

Value      Non   Oui
Frequency  1147  853
Proportion 0.574  0.426
-----
cinema
  n  missing distinct
  2000      0       2
```

```

Value      Non   Oui
Frequency    1174   826
Proportion  0.587  0.413
-----
sport
  n  missing distinct
  2000       0        2

Value      Non   Oui
Frequency    1277   723
Proportion  0.638  0.362
-----
heures.tv
      n  missing distinct      Info      Mean      Gmd
  1995       5        29     0.972    2.247    1.877
  .05       .10       .25      .50      .75      .90
  0         0        1        2        3        4
  .95
  5

lowest :  0.0  0.1  0.2  0.3  0.4, highest:  8.0  9.0 10.0 11.0 12.0
-----
```

On peut également transmettre juste une variable :

```
R> describe(d$sex)
```

```

d$sex
  n  missing distinct
  2000       0        2

Value      Homme Femme
Frequency    899   1101
Proportion  0.45  0.55
```

## En résumé

### Les Listes

- Les listes sont des objets unidimensionnels pouvant contenir tout type d'objet, y compris d'autres listes.
- Elles ont une longueur qu'on obtient avec `length`.
- On crée une liste avec `list` et on peut fusionner des listes avec `append`.

- Tout comme les vecteurs, les listes peuvent être nommées et les noms des éléments s'obtiennent avec `names`.
- Les crochets simples ( `[]` ) permettent de sélectionner les éléments d'une liste, en utilisant l'indexation par position, l'indexation par nom ou l'indexation par condition. Cela renvoie toujours une autre liste.
- Les doubles crochets ( `[[]]` ) renvoient directement le contenu d'un élément de la liste qu'on aura sélectionné par position ou par nom.
- Le symbole `$` est un raccourci pour facilement sélectionner un élément par son nom, `liste$nom` étant équivalent à `liste[["nom"]]`.

## Les Tableaux de données

- Les tableaux de données sont des listes avec des propriétés particulières :
  - i. tous les éléments sont des vecteurs ;
  - ii. tous les vecteurs ont la même longueur ;
  - iii. tous les vecteurs ont un nom et ce nom est unique.
- On peut créer un tableau de données avec `data.frame`.
- Les tableaux de données correspondent aux fichiers de données qu'on utilise usuellement dans d'autres logiciels de statistiques : les variables sont représentées en colonnes et les observations en lignes.
- Ce sont des objets bidimensionnels : `ncol` renvoie le nombre de colonnes et `nrow` le nombre de lignes.
- Les doubles crochets ( `[[]]` ) et le symbole dollar ( `$` ) fonctionnent comme pour les listes et permettent d'accéder aux variables.
- Il est possible d'utiliser des coordonnées bidimensionnelles avec les crochets simples ( `[]` ) en indiquant un critère sur les lignes puis un critère sur les colonnes, séparés par une virgule ( `,` ).

# Facteurs et vecteurs labellisés

Facteurs . . . . .	118
Vecteurs labellisés . . . . .	123
Les étiquettes de variable . . . . .	124
Les étiquettes de valeur . . . . .	126
Assignation et condition . . . . .	131
Quelques fonctions supplémentaires . . . . .	133

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #02 (les bases du langage R) sur [YouTube](#).

Dans le chapitre sur les vecteurs, page 67, nous avons abordé les types fondamentaux de vecteurs (numériques, textuels, logiques). Mais il existe de nombreux autres classes de vecteurs afin de représenter des données diverses (comme les dates). Dans ce chapitre, nous nous intéressons plus particulièrement aux variables catégorielles.

Les facteurs (ou *factors* en anglais) sont un type de vecteur géré nativement par **R** et utilisés dans de nombreux domaines (modèles statistiques, représentations graphiques, ...).

Les facteurs sont souvent mis en regard des données labellisées telles qu'elles sont utilisées dans d'autres logiciels comme **SPSS** ou **Stata**. Or, les limites propres aux facteurs font qu'ils ne sont pas adaptés pour rendre compte des différents usages qui sont fait des données labellisées. Plusieurs extensions (telles que **memisc** ou **Hmisc**) ont proposé leur propre solution qui, bien qu'elles apportaient un plus pour la gestion des données labellisées, ne permettaient pas que celles-ci soient utilisées en dehors de ces extensions ou des extensions compatibles. Nous aborderons ici une nouvelle classe de vecteurs, la classe **labelled**, introduite par l'extension **haven** (que nous aborderons dans le cadre de l'import de données, page 145) et qui peut être manipulée avec l'extension homonyme **labelled**.

# Facteurs

Dans ce qui suit on travaillera sur le jeu de données tiré de l'enquête *Histoire de vie*, fourni avec l'extension [questionr](#).

```
R> library(questionr)
  data(hdv2003)
  d <- hdv2003
```

Jetons un œil à la liste des variables de `d` :

```
R> str(d)
```

```
'data.frame': 2000 obs. of 20 variables:
 $ id          : int 1 2 3 4 5 6 7 8 9 10 ...
 $ age         : int 28 23 59 34 71 35 60 47 20 28 ...
 $ sexe        : Factor w/ 2 levels "Homme","Femme": 2 2 1 1 2 2 2 1 2 1 ...
 $ nivetud     : Factor w/ 8 levels "N'a jamais fait d'etudes",...: 8 NA 3 8 3
 6 3 6 NA 7 ...
 $ poids        : num 2634 9738 3994 5732 4329 ...
 $ occup        : Factor w/ 7 levels "Exerce une profession",...: 1 3 1 1 4 1 6
 1 3 1 ...
 $ qualif       : Factor w/ 7 levels "Ouvrier specialise",...: 6 NA 3 3 6 6 2 2 N
 A 7 ...
 $ freres.soeurs: int 8 2 2 1 0 5 1 5 4 2 ...
 $ cuso         : Factor w/ 3 levels "Oui","Non","Ne sait pas": 1 1 2 2 1 2 1 2
 1 2 ...
 $ relig         : Factor w/ 6 levels "Pratiquant regulier",...: 4 4 4 3 1 4 3 4
 3 2 ...
 $ trav.imp      : Factor w/ 4 levels "Le plus important",...: 4 NA 2 3 NA 1 NA 4
 NA 3 ...
 $ trav.satisf   : Factor w/ 3 levels "Satisfaction",...: 2 NA 3 1 NA 3 NA 2 NA 1
 ...
 $ hard.rock     : Factor w/ 2 levels "Non","Oui": 1 1 1 1 1 1 1 1 1 1 ...
 $ lecture.bd    : Factor w/ 2 levels "Non","Oui": 1 1 1 1 1 1 1 1 1 1 ...
 $ peche.chasse  : Factor w/ 2 levels "Non","Oui": 1 1 1 1 1 1 2 2 1 1 ...
 $ cuisine       : Factor w/ 2 levels "Non","Oui": 2 1 1 2 1 1 2 2 1 1 ...
 $ bricol        : Factor w/ 2 levels "Non","Oui": 1 1 1 2 1 1 1 2 1 1 ...
 $ cinema         : Factor w/ 2 levels "Non","Oui": 1 2 1 2 1 2 1 1 2 2 ...
 $ sport          : Factor w/ 2 levels "Non","Oui": 1 2 2 2 1 2 1 1 1 2 ...
 $ heures.tv     : num 0 1 0 2 3 2 2.9 1 2 2 ...
```

Nous voyons que de nombreuses variables de ce tableau de données, telles que `sex` ou `nivetud`, sont

du type facteur.

Les facteurs prennent leurs valeurs dans un ensemble de modalités prédéfinies et ne peuvent en prendre d'autres. La liste des valeurs possibles est donnée par la fonction `levels` :

```
R> levels(d$sex)
```

```
[1] "Homme" "Femme"
```

Si on veut modifier la valeur du sexe du premier individu de notre tableau de données avec une valeur non autorisée, on obtient un message d'erreur et une valeur manquante est utilisée à la place :

```
R> d$sex[1] <- "Chihuahua"
```

```
Warning in `[<-factor`(`*tmp*`, 1, value = structure(c(NA,
2L, 1L, 1L, : invalid factor level, NA generated
```

```
R> d$sex[1]
```

```
[1] <NA>
Levels: Homme Femme
```

```
R> d$sex[1] <- "Homme"
d$sex[1]
```

```
[1] Homme
Levels: Homme Femme
```

On peut très facilement créer un facteur à partir d'une variable textuelle avec la fonction `factor` :

```
R> v <- factor(c("H", "H", "F", "H"))
v
```

```
[1] H H F H
Levels: F H
```

Par défaut, les niveaux d'un facteur nouvellement créés sont l'ensemble des valeurs de la variable textuelle, ordonnées par ordre alphabétique. Cette ordre des niveaux est utilisé à chaque fois qu'on utilise des fonctions comme `table`, par exemple :

```
R> table(v)
```

```
v  
F H  
1 3
```

On peut modifier cet ordre au moment de la création du facteur en utilisant l'option `levels` :

```
R> v <- factor(c("H", "H", "F", "H"), levels = c("H", "F"))  
table(v)
```

```
v  
H F  
3 1
```

On peut aussi modifier l'ordre des niveaux d'une variable déjà existante :

```
R> d$qualif <- factor(d$qualif, levels = c(  
    "Ouvrier specialise", "Ouvrier qualifie",  
    "Employe", "Technicien", "Profession intermediaire", "Cadre", "Autre"  
)  
table(d$qualif)
```

Ouvrier specialise	Ouvrier qualifie
203	292
Employe	Technicien
594	86
Profession intermediaire	Cadre
160	260
Autre	
58	

**NOTE**

L'extension `questionr` propose une *interface interactive* pour le réordonnancement des niveaux d'un facteur. Cette fonction, nommée `iorder`, vous permet de réordonner les modalités de manière graphique et de générer le code R correspondant.

Dans l'exemple précédent, si vous exécutez :

```
R> iorder(d, "qualif")
```

RStudio devrait ouvrir une fenêtre semblable à celle de la figure ci-dessous.

### Interface de la commande iorder

Vous pouvez alors déplacer les modalités par glisser-déposer, vérifier le résultat dans l'onglet *Vérification* et, une fois le résultat satisfaisant, récupérer le code généré pour l'inclure dans votre script.

On peut également modifier les niveaux eux-mêmes. Imaginons que l'on souhaite créer une nouvelle variable `qualif.abr` contenant les noms abrégés des catégories socioprofessionnelles de `qualif`. On peut

alors procéder comme suit :

```
R> d$qualif.abr <- factor(d$qualif,
  levels = c(
    "Ouvrier specialise", "Ouvrier qualifie",
    "Employe", "Technicien", "Profession intermediaire", "Cadre", "Autre"
  ),
  labels = c("OS", "OQ", "Empl", "Tech", "Interm", "Cadre", "Autre")
)
table(d$qualif.abr)
```

	OS	OQ	Empl	Tech	Interm	Cadre	Autre
	203	292	594	86	160	260	58

Dans ce qui précède, le paramètre `levels` de `factor` permet de spécifier quels sont les niveaux retenus dans le facteur résultat, ainsi que leur ordre. Le paramètre `labels`, lui, permet de modifier les noms de ces niveaux dans le facteur résultat. Il est donc capital d'indiquer les noms de `labels` exactement dans le même ordre que les niveaux de `levels`. Pour s'assurer de ne pas avoir commis d'erreur, il est recommandé d'effectuer un tableau croisé entre l'ancien et le nouveau facteur :

```
R> table(d$qualif, d$qualif.abr)
```

	OS	OQ	Empl	Tech	Interm	Cadre
Ouvrier specialise	203	0	0	0	0	0
Ouvrier qualifie	0	292	0	0	0	0
Employe	0	0	594	0	0	0
Technicien	0	0	0	86	0	0
Profession intermediaire	0	0	0	0	160	0
Cadre	0	0	0	0	0	260
Autre	0	0	0	0	0	0

	Autre
Ouvrier specialise	0
Ouvrier qualifie	0
Employe	0
Technicien	0
Profession intermediaire	0
Cadre	0
Autre	58

On a donc ici un premier moyen d'effectuer un recodage des modalités d'une variable de type facteur. D'autres méthodes existent, que nous aborderons dans le chapitre Recodage, page 197.

À noter que par défaut, les valeurs manquantes ne sont pas considérées comme un niveau de facteur. On

peut cependant les transformer en niveau en utilisant la fonction `addNA`. Ceci signifie cependant qu'elles ne seront plus considérées comme manquantes par R mais comme une modalité à part entière :

```
R> summary(d$trav.satisf)
```

Satisfaction	Insatisfaction	Equilibre	NA's
480	117	451	952

```
R> summary(addNA(d$trav.satisf))
```

Satisfaction	Insatisfaction	Equilibre	<NA>
480	117	451	952

La fonction `addNAstr` de l'extension `questionr` fait la même chose mais permet de spécifier l'étiquette de la modalité des valeurs manquantes.

```
R> library(questionr)
summary(addNAstr(d$trav.satisf, "Manquant"))
```

Satisfaction	Insatisfaction	Equilibre	Manquant
480	117	451	952

## Vecteurs labellisés

Nous abordons ici une nouvelle classe de vecteurs, la classe `haven_labelled`, introduite récemment par l'extension `haven` (que nous aborderons dans le cadre de l'import de données, page 145) et qui peut être manipulée avec l'extension homonyme `labelled`.

Pour cette section, nous allons utiliser d'autres données d'exemple, également disponibles dans l'extension `questionr`. Il s'agit d'un ensemble de trois tableaux de données (`menages`, `femmes` et `enfants`) contenant les données d'une enquête de fécondité. Commençons par les charger en mémoire :

```
R> library(questionr)
data(feconde)
```

Pour ailleurs, nous allons avoir besoin de l'extension `labelled` qui permet de manipuler ces données labellisées.

```
R> library(labelled)
```

## Les étiquettes de variable

Les étiquettes de variable permettent de donner un nom long, plus explicite, aux différentes colonnes d'un tableau de données (ou encore directement à un vecteur autonome).

La visonneuse de données de **RStudio** sait reconnaître et afficher ces étiquettes de variable lorsqu'elles existent. Essayez par exemple la commande suivante :

```
R> View(femmes)
```

Les fonctions `look_for` de l'extension `labelled` et `describe` de l'extension `questionr` affichent également les étiquettes de variables lorsqu'elles existent.

```
R> look_for(femmes, "rés")
```

pos	variable	label	col_type	values
<chr>	<chr>	<chr>	<chr>	<chr>
7	milieu	Milieu de résidence	dbl+lbl	[1] urbain [2] rural
8	region	Région de résidence	dbl+lbl	[1] Nord [2] Est [3] Sud [4] Ouest

```
R> describe(femmes$id_femme)
```

[2000 obs.] Identifiant de l'enquêtée
numeric: 391 1643 85 881 1981 1072 1978 1607 738 1656 ...
min: 1 - max: 2000 - NAs: 0 (0%) - 2000 unique values

Pour manipuler les étiquettes de variable, il suffit d'utiliser la fonction `var_label` de l'extension `labelled`.

```
R> var_label(femmes$id_menage)
```

[1] "Identifiant du ménage"
-----------------------------

```
R> var_label(femmes$id_menage) <- "ID du ménage auquel elle appartient"
var_label(femmes$id_menage)
```

```
[1] "ID du ménage auquel elle appartient"
```

On utilisera la valeur `NULL` pour supprimer une étiquette :

```
R> v <- c(1, 5, 2, 4, 1)
var_label(v)
```

```
NULL
```

```
R> var_label(v) <- "Ma variable"
var_label(v)
```

```
[1] "Ma variable"
```

```
R> var_label(v) <- NULL
var_label(v)
```

```
NULL
```

```
R> var_label(v) <- "Une autre étiquette"
var_label(v)
```

```
[1] "Une autre étiquette"
```

Le fait d'ajouter une étiquette à un vecteur ne modifie en rien son type. Regardons la structure de notre objet `v` :

```
R> str(v)
```

```
num [1:5] 1 5 2 4 1
- attr(*, "label")= chr "Une autre étiquette"
```

Que voit-on ? Notre vecteur possède maintenant ce qu'on appelle un **attribut**, c'est-à-dire une information supplémentaire qui lui est attachée. Un objet peut avoir plusieurs attributs. Ici, notre étiquette de variable

est stocké dans un attribut nommé `"label"`. Cela ne modifie en rien sa nature. Il ne s'agit que d'information en plus. Toutes les fonctions ne tiennent pas compte des étiquettes de variable. Peu importe ! La présence d'un attribut ne les empêchera de fonctionner. De même, même si l'extension `labelled` n'est pas installée sur votre machine, vous pourrez toujours manipuler vos données comme si de rien n'était.

On peut associer une étiquette de variable à n'importe quel type de variable, qu'elle soit numérique, textuelle, un facteur ou encore des dates.

## Les étiquettes de valeur

Les étiquettes de valeur consistent à attribuer une étiquette textuelle à certaines valeurs d'un vecteur. Elles ne peuvent s'appliquer qu'aux vecteurs numériques ou textuels.

Lorsqu'un vecteur possède des étiquettes de valeur, sa classe change et devient `labelled`. Regardons déjà quelques exemples. Tout d'abord, jetons un aperçu au contenu de l'objet `femmes` grâce à la fonction `glimpse` de l'extension `dplyr`.

```
R> library(dplyr)
    glimpse(femmes)
```

```
Rows: 2,000
Columns: 17
$ id_femme      <dbl> 391, 1643, 85, 881, 1981, 1072, ...
$ id_menage     <dbl> 381, 1515, 85, 844, 1797, 1015, ...
$ poids         <dbl> 1.803150, 1.803150, 1.803150, 1....
$ date_entretien <date> 2012-05-05, 2012-01-23, 2012-01...
$ date_naissance <date> 1997-03-07, 1982-01-06, 1979-01...
$ age           <dbl> 15, 30, 33, 43, 25, 18, 45, 23, ...
$ milieu        <dbl+lbl> 2, 2, 2, 2, 2, 2, 2, 2, 2...
$ region        <dbl+lbl> 4, 4, 4, 4, 4, 3, 3, 3, 3...
$ educ          <dbl+lbl> 0, 0, 0, 0, 1, 0, 0, 0, 0...
$ travail       <dbl+lbl> 1, 1, 0, 1, 1, 0, 1, 0, 1...
$ matri         <dbl+lbl> 0, 2, 2, 2, 1, 0, 1, 1, 2, 5...
$ religion      <dbl+lbl> 1, 3, 2, 3, 2, 2, 3, 1, 3, 3...
$ journal       <dbl+lbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
$ radio          <dbl+lbl> 0, 1, 1, 0, 0, 1, 1, 0, 0, 0...
$ tv             <dbl+lbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0...
$ nb_enf_ideal   <dbl+lbl> 4, 4, 4, 4, 4, 5, 10, ...
$ test           <dbl+lbl> 0, 9, 0, 0, 1, 0, 0, 0, 1...
```

Il apparaît que la variable `region` est de type `haven_labelled`. On peut le confirmer avec `class`.

```
R> class(femmes$region)
```

```
[1] "haven_labelled" "vctrs_vctr"     "double"
```

Regardons les premières valeurs prises par cette variable.

```
R> head(femmes$region)
```

```
<labelled<double>[6]>: Région de résidence
[1] 4 4 4 4 4 3

Labels:
  value label
    1   Nord
    2   Est
    3   Sud
    4 Ouest
```

Nous voyons que quatre étiquettes de valeurs ont été associées à notre variable. Le code 1 correspond ainsi à la région «Nord», le code 2 à la région «Est», etc. Laissons de côté pour le moment la colonne `is_na` que nous aborderons dans une prochaine section.

La liste des étiquettes est également renvoyée par la fonction `describe` de `questionr`.

```
R> describe(femmes$region)
```

```
[2000 obs.] Région de résidence
labelled double: 4 4 4 4 4 3 3 3 3 ...
min: 1 - max: 4 - NAs: 0 (0%) - 4 unique values
4 value labels: [1] Nord [2] Est [3] Sud [4] Ouest

      n      %
[1] Nord   707  35.4
[2] Est    324  16.2
[3] Sud    407  20.3
[4] Ouest  562  28.1
Total    2000 100.0
```

L'extension `labelled` fournit la fonction `val_labels` qui renvoie la liste des étiquettes de valeurs d'une variable sous la forme d'un vecteur nommé et la fonction `val_label` (notez l'absence de 's') qui renvoie l'étiquette associée à une valeur particulière. S'il n'y a pas d'étiquette de valeur, ces fonctions renvoient `NULL`.

```
R> val_labels(femmes$region)
```

Nord	Est	Sud	Ouest
1	2	3	4

```
R> val_label(femmes$region, 2)
```

```
[1] "Est"
```

```
R> val_label(femmes$region, 6)
```

```
NULL
```

```
R> val_labels(femmes$age)
```

```
NULL
```

Re-regardons d'un peu plus près les premières valeurs de notre variable *region*.

```
R> head(femmes$region)
```

```
<labelled<double>[6]>: Région de résidence
```

```
[1] 4 4 4 4 4 3
```

Labels:

value	label
1	Nord
2	Est
3	Sud
4	Ouest

On s’aperçoit qu’il s’agit de valeurs numériques. Et l’affichage indique que notre variable est plus précisément du type `labelled double`. Pour rappel, `double` est synonyme de `numeric`. Autrement dit, la classe `haven_labelled` ne modifie pas le type sous-jacent d’un vecteur, que l’on peut toujours obtenir avec la fonction `typeof`. Nous pouvons également tester si notre variable est numérique avec la fonction `is.numeric`.

```
R> typeof(femmes$region)
```

```
[1] "double"
```

```
R> is.numeric(femmes$region)
```

```
[1] TRUE
```

À la différence des facteurs, le type original d'une variable labellisée n'est pas modifié par la présence d'étiquettes de valeur. Ainsi, il reste possible de calculer une moyenne à partir de notre variable *region* (même si cela n'est pas pertinent ici d'un point de vue sémantique).

```
R> mean(femmes$region)
```

```
[1] 2.412
```

Avec un facteur, nous aurions eu un bon message d'erreur.

```
R> mean(d$nivetud)
```

```
Warning in mean.default(d$nivetud): argument is not numeric  
or logical: returning NA
```

```
[1] NA
```

Nous allons voir qu'il est aussi possible d'associer des étiquettes de valeurs à des vecteurs textuels. Créons tout d'abord un vecteur textuel qui nous servira d'exemple.

```
R> v <- c("f", "f", "h", "f", "h")  
v
```

```
[1] "f" "f" "h" "f" "h"
```

Le plus facile pour lui associer des étiquettes de valeur est d'utiliser `val_label`.

```
R> val_label(v, "f") <- "femmes"  
val_label(v, "h") <- "hommes"  
v
```

```
<labelled<character>[5]>  
[1] f f h f h
```

```
Labels:  
value label  
f femmes  
h hommes
```

```
R> typeof(v)
```

```
[1] "character"
```

Notre vecteur `v` a automatiquement été transformé en un vecteur de la classe `labelled`. Mais son type sous-jacent est resté `"character"`. Par ailleurs, les données elle-même n'ont pas été modifiées et ont conservé leurs valeurs originales.

Il est également possible de définir/modifier/supprimer l'ensemble des étiquettes de valeur d'une variable avec `val_labels` en lui assignant un vecteur nommé.

```
R> val_labels(v) <- c(Homme = "h", Femme = "f", "Valeur indéterminée" = "i")  
v
```

```
<labelled<character>[5]>  
[1] f f h f h
```

value	label
h	Homme
f	Femme
i	Valeur indéterminée

Comme précédemment, on utilisera `NULL` pour supprimer une ou toutes les étiquettes.

```
R> val_label(v, "i") <- NULL  
v
```

```
<labelled<character>[5]>
```

```
[1] f f h f h
```

Labels:

value	label
h	Homme
f	Femme

```
R> val_labels(v) <- NULL
v
```

```
[1] "f" "f" "h" "f" "h"
```

```
R> class(v)
```

```
[1] "character"
```

Si l'on supprime toutes les étiquettes de valeur, alors notre vecteur retrouve sa classe initiale.

## Attribution et condition

Les étiquettes de valeur sont plus souples que les facteurs, en ce sens qu'il n'est pas obligatoire d'indiquer une étiquette pour chaque valeur prise par une variable. Alors qu'il n'est pas possible avec un facteur d'assigner une valeur qui n'a pas été préalablement définie comme une des modalités possibles du facteur, nous n'avons pas cette limite avec les vecteurs labellisés.

```
R> femmes$region[3] <- 5
```

Important : quand on assigne une valeur à un facteur, on doit transmettre le texte correspondant à la modalité, alors que pour un vecteur labellisé on transmettra le code sous-jacent (pour rappel, les étiquettes de valeur ne sont qu'une information additionnelle).

De plus, nous avons vu que les données initiales n'étaient pas modifiées par l'ajout ou la suppression d'étiquettes de valeur, alors que pour les facteurs ce n'est pas vrai. Pour mieux comprendre, essayons la commande suivante :

```
R> unclass(factor(v))
```

```
[1] 1 1 2 1 2
attr(,"levels")
```

```
[1] "f" "h"
```

Un facteur stocke de manière interne les valeurs sous la forme d'une suite d'entiers, démarrant toujours par 1, forcément consécutifs, et dont les valeurs dépendent de l'ordre des facteurs. Pour s'en rendre compte :

```
R> unclass(factor(v, levels = c("h", "f")))
```

```
[1] 2 2 1 2 1  
attr(,"levels")  
[1] "h" "f"
```

```
R> unclass(factor(v, levels = c("f", "h")))
```

```
[1] 1 1 2 1 2  
attr(,"levels")  
[1] "f" "h"
```

Ce qui importe pour un facteur ce sont les modalités de ce dernier tandis que pour un vecteur labellisé ce sont les valeurs du vecteur elles-mêmes. Cela reste vrai pour l'écriture de conditions.

Prenons un premier exemple avec un facteur :

```
R> describe(d$sex)
```

```
[2000 obs.]  
nominal factor: "Homme" "Femme" "Homme" "Homme" "Femme" "Femme" "Femme" "Homme"  
"Femme" "Homme" ...  
2 levels: Homme | Femme  
NAs: 0 (0%)  
  
n %  
Homme 900 45  
Femme 1100 55  
Total 2000 100
```

```
R> table(d$sex == "Homme")
```

```
FALSE TRUE  
1100 900
```

```
R> table(d$sexe == 1)
```

```
FALSE  
2000
```

La condition valide est celle utilisant "Homme" qui est la valeur de la modalité du facteur.

Et avec un vecteur labellisé ?

```
R> describe(femmes$milieu)
```

```
[2000 obs.] Milieu de résidence  
labelled double: 2 2 2 2 2 2 2 2 2 ...  
min: 1 - max: 2 - NAs: 0 (0%) - 2 unique values  
2 value labels: [1] urbain [2] rural  
  
n      %  
[1] urbain  912  45.6  
[2] rural   1088 54.4  
Total       2000 100.0
```

```
R> table(femmes$milieu == "urbain")
```

```
Erreur : Can't combine `...1` <character> and `...2` <double>.
```

```
R> table(femmes$milieu == 1)
```

```
FALSE  TRUE  
1088   912
```

Ici, pour être valide, la condition doit porter sur les valeurs de la variable elle-même et non sur les étiquette.

## Quelques fonctions supplémentaires

L'extension **labelled** fournit quelques fonctions supplémentaires qui peuvent s'avérer utiles :

- `labelled` pour créer directement des vecteurs labellisés ;
- `nolabel_to_na` pour convertir les valeurs n'ayant pas d'étiquette en `NA` ;
- `val_labels_to_na` qui, à l'inverse, converti les valeurs avec étiquette en `NA` ;

- `sort_val_labels` pour trier l’ordre des étiquettes de valeurs.

On pourra se référer à l’aide de chacune de ces fonctions.

L’import de données labellisées, page 145 et le recodage de variables, page 202 (dont la conversion d’un vecteur labellisé en facteur) seront quant à eux abordés dans les prochains chapitres.

# Organiser ses fichiers

Le répertoire de travail . . . . .	135
Les projets dans RStudio . . . . .	136
Créer un nouveau projet . . . . .	137
Fonctionnement par défaut des projets . . . . .	140
Options des projets . . . . .	141
Naviguer d'un projet à un autre . . . . .	141
Voir aussi . . . . .	142
Appeler un script depuis un autre script . . . . .	142

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #02 (les bases du langage R) sur [YouTube](#).

Ce chapitre est évoqué dans le webin-R #03 (statistiques descriptives avec gtsummary et esquisse) sur [YouTube](#).

## Le répertoire de travail

À chaque fois que l'on demandera à R de charger ou d'enregistrer un fichier (en particulier lorsque l'on cherchera à importer des données, voir le chapitre dédié, page 145), R évaluera le nom du fichier qu'on lui a transmis par rapport au répertoire de travail actuellement défini, qui correspond au répertoire dans lequel R est actuellement en train de s'exécuter.

Pour connaître le répertoire de travail actuel, on pourra utiliser la fonction `getwd` :

```
R> getwd()
```

Lorsque l'on travaille sous **RStudio**, le répertoire de travail est également affiché dans le quadrant inférieur droit, en gris, à la droite du mot *Console* (voir la capture d'écran ci-après).

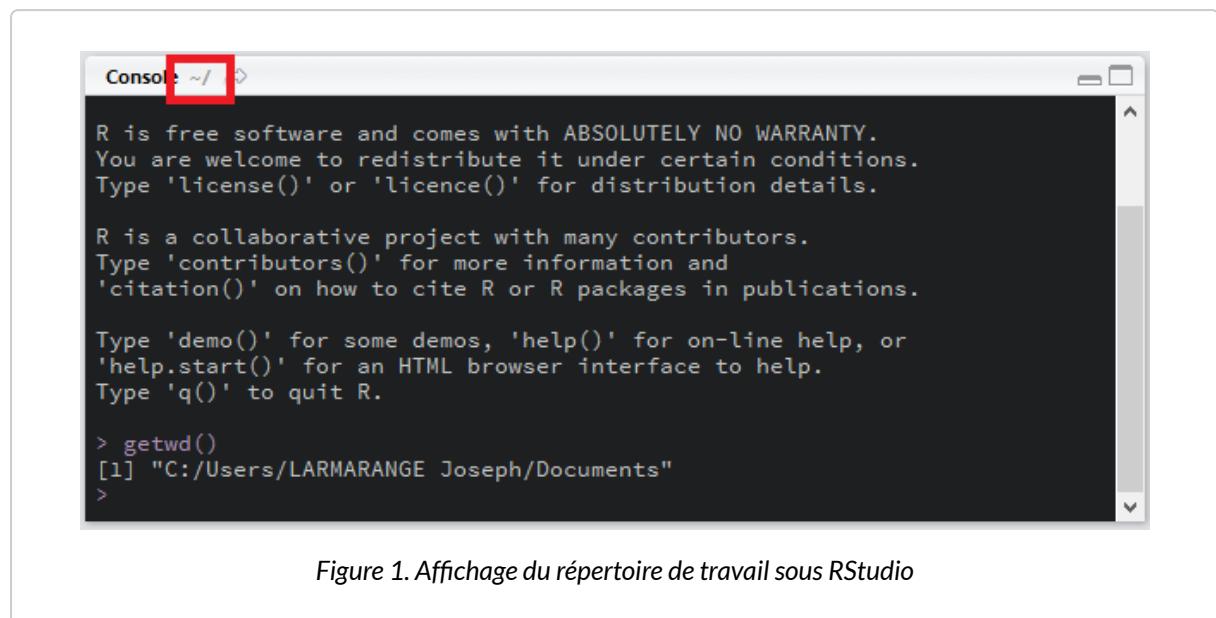


Figure 1. Affichage du répertoire de travail sous RStudio

Le symbole `~` correspond dans ce cas-là au répertoire utilisateur système, dont l'emplacement dépend du système d'exploitation. Sous **Windows**, il s'agit du répertoire *Mes documents* ou *Documents* (le nom varie suivant la version de **Windows**).

Le répertoire de travail peut être modifié avec la fonction `setwd` ou, sous **RStudio**, via le menu *Session > Set Working Directory*. Cependant, nous allons voir que nous n'aurons en pratique presque jamais besoin de le faire si l'on travaille avec **RStudio**.

## Les projets dans RStudio

**RStudio** dispose d'une fonctionnalité très pratique pour organiser son travail en différents projets.

L'idée principale est de réunir tous les fichiers / documents relatifs à un même projet (que ce soit les données, les scripts, les rapports automatisés...) dans un répertoire dédié<sup>1</sup>.

Le menu *Projects* est accessible via une icône dédiée située tout en haut à droite (voir la capture d'écran ci-après).

---

1. Dans lequel il sera possible de créer des sous-répertoires.

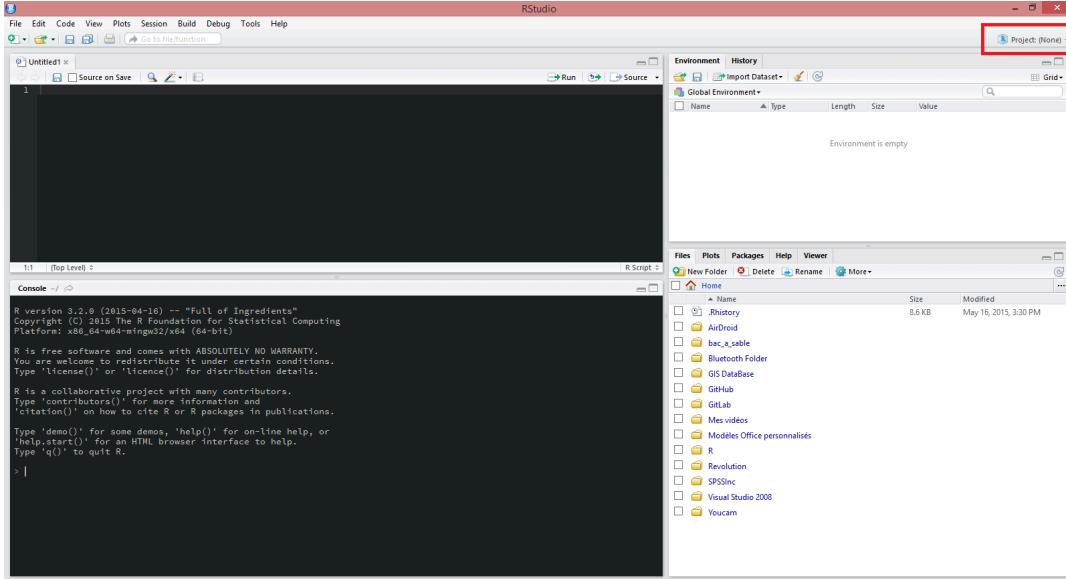


Figure 2. Accès au menu Projects sous RStudio

## Créer un nouveau projet

Dans le menu **Projects** on sélectionnera l'option **New project**. **RStudio** nous demandera dans un premier temps si l'on souhaite créer un projet (i) dans un nouveau répertoire, (ii) dans un répertoire déjà existant ou bien (iii) à partir d'un gestionnaire de versions (**Git** ou **SVN**).

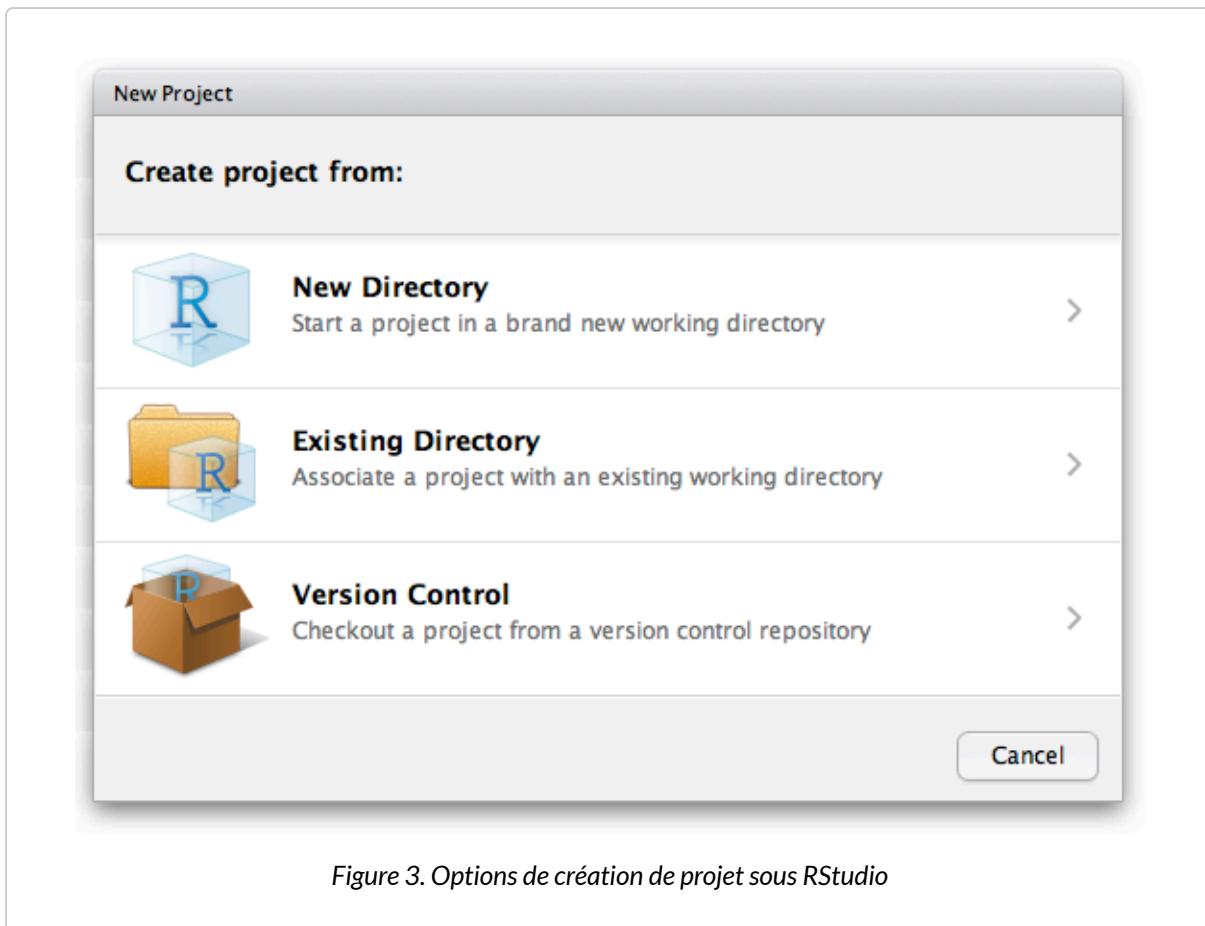


Figure 3. Options de création de projet sous RStudio

Si vous débutez avec R, laissez de côté pour le moment les gestionnaires de versions qui sont destinés aux utilisateurs avancés. Dans le cadre d'un usage courant, on aura recours à *New Directory*.

RStudio nous demande alors le type de projet que l'on souhaite créer : (i) un projet vide, (ii) une extension R ou (iii) une application Shiny.

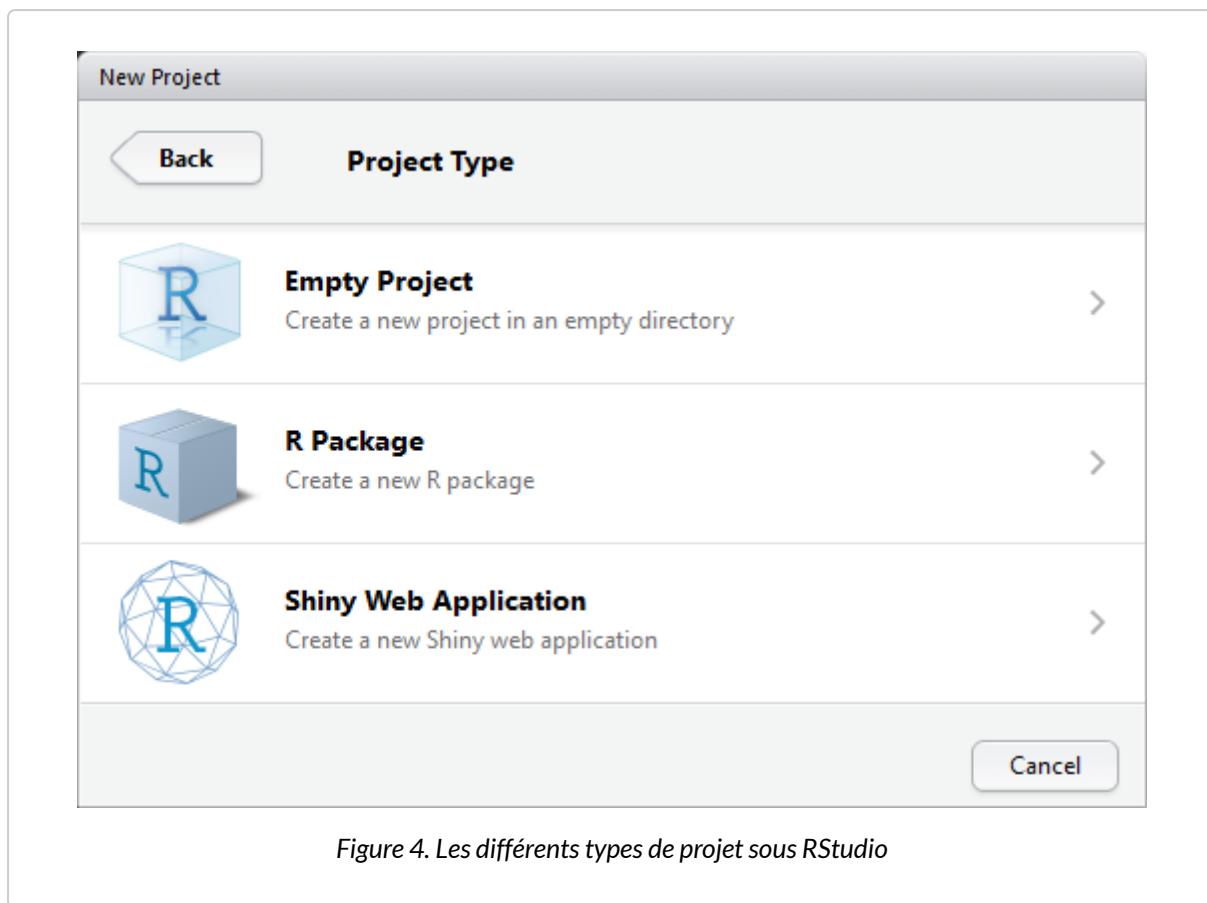


Figure 4. Les différents types de projet sous RStudio

Il est encore un peu tôt pour se lancer dans la création de sa propre extension pour R (voir le chapitre [Développer un package](#)). Les applications **Shiny** (voir le [chapitre dédié](#)) sont des applications webs interactives. Là encore, on attendra une meilleure maîtrise de R pour se lancer dans ce type de projets. Dans un contexte d'analyse d'enquêtes, on choisira dès lors *Empty project*.

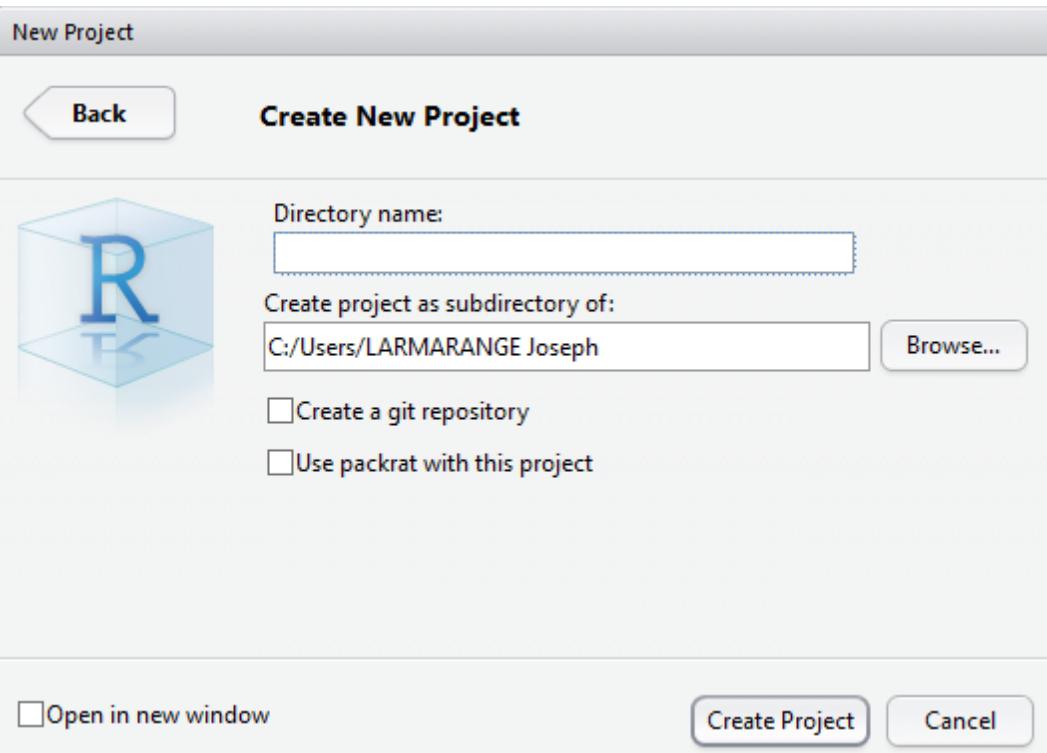


Figure 5. Création d’un projet dans un nouveau répertoire avec RStudio

En premier lieu, on indiquera le nom de notre projet, qui sera également le nom du répertoire qui sera créé pour stocker les données du projet. Puis, on indiquera le répertoire parent, c'est-à-dire le répertoire dans lequel le répertoire de notre projet sera créé.

Les deux options suivantes concernent que les utilisateurs avancés. **RStudio** nous demande s'il on veut activer **Git** sur ce projet (**Git** étant un gestionnaire de versions, l'option n'étant affichée que si **Git** est installé sur votre PC) et s'il on souhaite utiliser l'extension **packrat** sur ce projet. **packrat** permet une gestion des extensions utilisées, projet par projet, ce qui n'est vraiment utile que dans le cadre d'analyses bien spécifiques.

Il ne nous reste plus qu'à cliquer sur *Create Project*.

## Fonctionnement par défaut des projets

Lorsque l'on ouvre un projet, **RStudio** effectue différentes actions :

- le nom du projet est affiché en haut à droite à côté de l'icône projets ;
- une nouvelle session **R** est exécutée (ainsi s'il on passe d'un projet à un autre, les objets du projet qu'on vient de fermer ne sont plus en mémoire) ;

- le répertoire de travail de **R** est défini comme étant le répertoire du projet (d'où le fait que l'on n'a pas à se préoccuper de définir le répertoire de travail lorsque l'on travaille avec des projets **RStudio**) ;
- les objets créés (et sauvegardés dans le fichier `.Rdata`) lors d'une précédente séance de travail sont chargés en mémoire ;
- l'historique des commandes saisies lors de nos précédentes séances de travail sont chargées dans l'onglet *History* ;
- les scripts ouverts lors d'une précédente séance de travail sont automatiquement ouverts ;
- divers paramètres de **RStudio** sont restaurés dans l'état dans lequel ils étaient la dernière fois que l'on a travaillé sur ce projet.

Autrement dit, lorsque l'on ouvre un projet **RStudio**, on revient à l'état de notre projet tel qu'il était la dernière fois que l'on a travaillé dessus. Pratique, non ?

Petite précision toutefois, les extensions que l'on avait chargées en mémoire avec la fonction `library` ne sont pas systématiquement rechargées en mémoire. Il faudra donc les appeler à nouveau lors de notre séance de travail.

## Options des projets

Via le menu *Projects > Projects options* (accessible via l'icône projets en haut à droite), il est possible de personnaliser plusieurs options spécifiquement pour ce projet.

On retiendra surtout les 3 options principales de l'onglet *General* :

- à l'ouverture du projet, doit-on charger en mémoire les objets sauvegardés lors d'une précédente séance de travail ?
- à la fermeture du projet, doit-on sauvegarder (dans le fichier `.Rdata`) les différents objets en mémoire ? Si l'on choisit l'option *Ask*, alors une fenêtre vous demandera s'il faut faire cette sauvegarde chaque fois que vous fermerez le projet.
- à la fermeture du projet, faut-il sauver l'historique des commandes ?

## Naviguer d'un projet à un autre

**RStudio** se souvient des derniers projets sur lesquels vous avez travaillé. Lorsque vous cliquez sur le menu projets, vous verrez une liste de ces différents projets. Il suffit de cliquer sur le nom du projet désiré pour fermer automatiquement le projet en cours et ouvrir le projet désiré.

Votre projet n'apparaît pas dans la liste ? Pas de panique. Il suffit de sélectionner *Open project* puis de parcourir vos répertoires pour indiquer à **RStudio** le projet à ouvrir.

Vous pouvez noter au passage une option *Open project in new window* qui permet d'ouvrir un projet dans une nouvelle fenêtre. En effet, il est tout à fait possible d'avoir plusieurs projets ouverts en même temps. Dans ce cas là, chaque projet aura sa propre session **R**. Les objets chargés en mémoire pour le projet A ne

seront pas accessibles dans le cadre du projet B et inversement.

## Voir aussi

On pourra se référer à la documentation officielle de RStudio : <https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects>.

# Appeler un script depuis un autre script

Au sein d'un même projet, on peut avoir plusieurs scripts R. Cela permet de mieux organiser son code. Par exemple, on pourra avoir un premier script chargé d'importer les données, un second dédié à la création de nouvelles variables et un troisième dédié aux analyses statistiques.

Il est possible d'appeler un script au sein d'un autre script à l'aide de la fonction `source` à laquelle on précisera le nom de fichier du script en question.

Supposons par exemple que l'on ait préparé un script `preparation.R` chargé d'importer les données et de les mettre en forme. Au début de notre script `analyses.R`, on pourra indiquer :

```
R> source("preparation.R")
```

Si l'on exécute notre script `analyses.R`, au moment de l'appel à `source("preparation.R")`, le fichier `preparation.R` sera chargé en mémoire et exécuté, puis le programme continuera avec les commandes suivant du fichier `analyses.R`.

Ici, on a indiqué à `source` le fichier `preparation.R` sans mention de répertoire. Dès lors, R va aller chercher ce fichier dans le répertoire de travail. Sur un gros projet, on peut être amené à organiser ses fichiers en plusieurs sous-répertoires pour une meilleure lisibilité. Dès lors, il faudra indiquer le chemin relatif pour accéder à un fichier, c'est-à-dire le chemin à partir du répertoire de travail. Supposons que notre fichier `preparation.R` est enregistré dans un sous-répertoire `import`. Dans ce cas-là, on appellera notre fichier ainsi :

```
R> source("import/preparation.R")
```

#### NOTE

On remarquera qu'on a utilisé une barre oblique ou *slash* ( / ) entre le nom du répertoire et le nom du fichier, ce qui est l'usage courant sous **Linux** et **Mac OS X**, tandis que sous **Windows** on utilise d'ordinaire une barre oblique inversée ou *antislash* ( \ ). Sous **R**, on utilisera toujours la barre oblique simple ( / ), **R** sachant « retrouver ses petits » selon le système d'exploitation.

Par ailleurs, l'autocomplétion de **RStudio** fonctionne aussi pour les noms de fichiers. Essayez par exemple d'appuyer sur la touche **Tab** après avoir taper les premières lettres du nom de votre fichier.



# Import de données

Importer des fichiers texte . . . . .	146
Structure d'un fichier texte . . . . .	146
Interface graphique avec RStudio . . . . .	147
Dans un script . . . . .	148
Importer depuis des logiciels de statistique . . . . .	149
SPSS . . . . .	150
SAS . . . . .	152
Stata . . . . .	153
Excel . . . . .	154
dBase . . . . .	155
Feuille de calcul Google Sheets . . . . .	155
Données spatiales . . . . .	156
Shapefile . . . . .	156
Rasters . . . . .	157
Connexion à des bases de données . . . . .	157
Interfaçage via l'extension DBI . . . . .	157
Utilisation de dplyr et dbplyr . . . . .	159
Ressources . . . . .	160
Autres sources . . . . .	160
Sauver ses données . . . . .	161

## IMPORTANT

Importer des données est souvent l’une des premières opérations que l’on effectue lorsque l’on débute sous R, et ce n’est pas la moins compliquée. En cas de problème il ne faut donc pas hésiter à demander de l’aide par les différents moyens disponibles (voir le chapitre Où trouver de l’aide ?, page 163) avant de se décourager.

N’hésitez donc pas à relire régulièrement ce chapitre en fonction de vos besoins.

Avant toute chose, il est impératif de bien organiser ses différents fichiers (voir le chapitre dédié, page 135). Concernant les données sources que l’on utilisera pour ses analyses, je vous recommande de les placer dans un sous-répertoire dédié de votre projet.

Lorsque l’on importe des données, il est également impératif de vérifier que l’import s’est correctement déroulé (voir la section Inspecter les données, page 39 du chapitre Premier travail avec les données).

# Importer des fichiers texte

Les fichiers texte constituent un des formats les plus largement supportés par la majorité des logiciels statistiques. Presque tous permettent d’exporter des données dans un format texte, y compris les tableurs comme **Libre Office**, **Open Office** ou **Excel**.

Cependant, il existe une grande variété de format texte, qui peuvent prendre différents noms selon les outils, tels que texte tabulé ou texte ( séparateur : tabulation ), **CSV** (pour comma-separated value, sachant que suivant les logiciels le séparateur peut être une virgule ou un point-virgule).

## Structure d’un fichier texte

Dès lors, avant d’importer un fichier texte dans R, il est indispensable de regarder comment ce dernier est structuré. Il importe de prendre note des éléments suivants :

- La première ligne contient-elle le nom des variables ? Ici c’est le cas.
- Quel est le caractère séparateur entre les différentes variables (encore appelé séparateur de champs) ? Dans le cadre d’un fichier CSV, il aurait pu s’agir d’une virgule ou d’un point-virgule.
- Quel est le caractère utilisé pour indiquer les décimales (le séparateur décimal) ? Il s’agit en général d’un point (à l’anglo-saxonne) ou d’une virgule (à la française).
- Les valeurs textuelles sont-elles encadrées par des guillemets et, si oui, s’agit-il de guillements simples ( ‘ ) ou de guillemets doubles ( “ ) ?
- Pour les variables textuelles, y a-t-il des valeurs manquantes et si oui comment sont-elles

indiquées ? Par exemple, le texte `NA` est parfois utilisé.

Il ne faut pas hésitez à ouvrir le fichier avec un éditeur de texte pour le regarder de plus près.

## Interface graphique avec RStudio

RStudio fournit une interface graphique pour faciliter l'import d'un fichier texte. Pour cela, il suffit d'aller dans le menu `File > Import Dataset` et de choisir l'option `From CSV1`. Cette option est également disponible via l'onglet `Environment` dans le quadrant haut-droite.

Pour la suite de la démonstration, nous allons utiliser le fichier exemple [http://larmarange.github.io/analyse-R/data/exemple\\_texte\\_tabule.txt](http://larmarange.github.io/analyse-R/data/exemple_texte_tabule.txt).

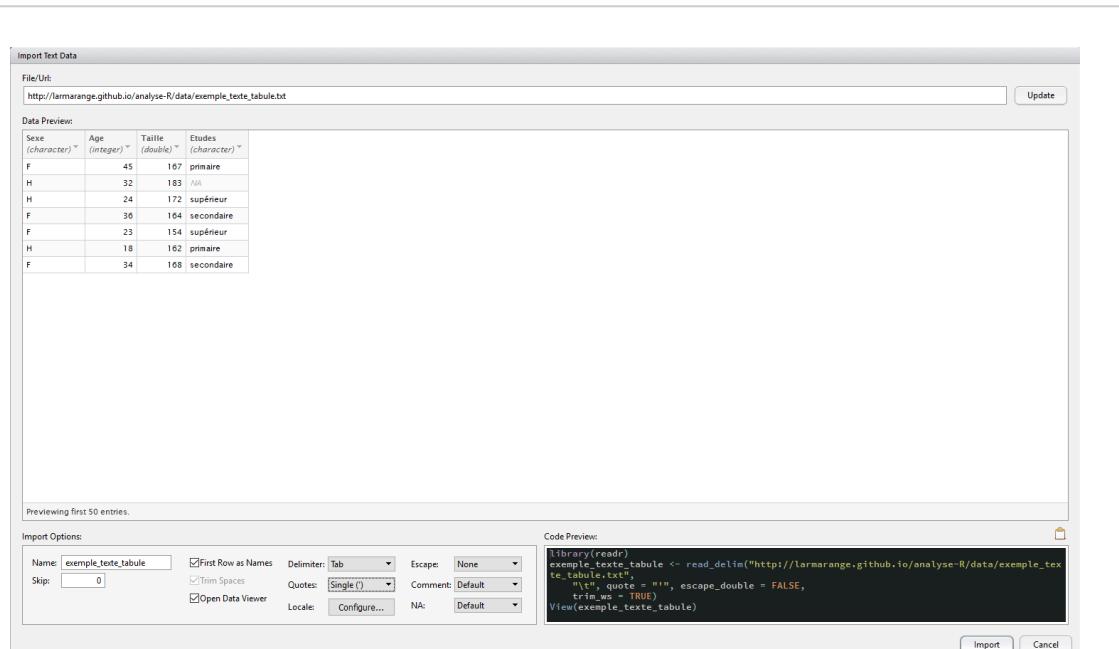


Figure 1. Importer un fichier texte avec RStudio

L'interface de RStudio vous présente sous `Import Options` les différentes options d'import disponible. La section `Data Preview` vous permet de voir en temps réel comment les données sont importées. La section `Code Preview` vous indique le code R correspondant à vos choix. Il n'y a plus qu'à le copier/coller dans un de vos scripts ou à cliquer sur `Import` pour l'exécuter.

Vous pourrez remarquer que RStudio fait appel à l'extension `readr` du tidyverse pour l'import des données via la fonction `read_csv`.

---

1. L'option CSV fonctionne pour tous les fichiers de type texte, même si votre fichier a une autre extension, `.txt` par exemple

`readr` essaie de deviner le type de chacune des colonnes, en se basant sur les premières observations. En cliquant sur le nom d’une colonne, il est possible de modifier le type de la variable importée. Il est également possible d’exclure une colonne de l’import (*skip*).

## Dans un script

L’interface graphique de **RStudio** fournit le code d’import. On peut également l’adapter à ces besoins en consultant la page d’aide de `read_csv` pour plus de détails. Par exemple :

```
R> library(readr)
d <- read_delim("http://larmarange.github.io/analyse-R/data/exemple_texte_table.txt",
  delim = "\t", quote = "")
```

```
-- Column specification -----
cols(
  Sexe = col_character(),
  Age = col_double(),
  Taille = col_number(),
  Etudes = col_character()
)
```

Le premier élément peut être un lien internet ou bien le chemin local vers un fichier. Afin d’organiser au mieux vos fichiers, voir le chapitre Organiser ses fichiers, page 135.

### NOTE

Certains caractères sont parfois précédés d’une barre oblique inversée ou *antislash* (\ ). Cela correspond à des caractères spéciaux. En effet, " " est utilisé pour délimiter dans le code le début et la fin d’une chaîne de caractères. Comment indiquer à R le caractère " " proprement dit. Et bien avec \" ". De même, \t sera interprété comme une tabulation et non comme la lettre t .

Pour une liste complète des caractères spéciaux, voir `?Quotes` .

```
R> class(d)
```

```
[1] "spec_tbl_df" "tbl_df"      "tbl"          "data.frame"
```

```
R> d
```

L'objet renvoyé est un tableau de données ou `data.frame`. Plus précisément, il s'agit d'un `tibble`, c'est-à-dire un tableau de données légèrement amélioré facilement utilisable avec les différentes extensions du `tidyverse`. Pas de panique, c'est un tableau de données comme les autres. Disons qu'il est possible de faire un peu plus de choses avec. Pour cela, voir le chapitre dédié à `dplyr`, page 233.

`readr` propose plusieurs fonctions proches : `read_delim`, `read_csv`, `read_csv2` et `read_tsv`. Elles fonctionnent toutes de manière identique et ont les mêmes arguments. Seule différence, les valeurs par défaut de certains paramètres.

#### NOTE

Dans des manuels ou des exemples en ligne, vous trouverez parfois mention des fonctions `read.table`, `read.csv`, `read.csv2`, `read.delim` ou encore `read.delim2`. Il s'agit des fonctions natives et historiques de R (extension `utils`) dédiées à l'import de fichiers textes. Elles sont similaires à celles de `readr` dans l'idée générale mais diffèrent dans leurs détails et les traitements effectués sur les données (pas de détection des dates par exemple). Pour plus d'information, vous pouvez vous référer à la page d'aide de ces fonctions.

## Importer depuis des logiciels de statistique

Plusieurs extensions existent pour importer des fichiers de données issus d'autres logiciels de statistiques. En premier lieu, il y a `foreign`, installée par défaut avec R et décrite en détails dans le manuel *R Data Import/Export* disponible sur <http://cran.r-project.org/manuals.html>. Un des soucis majeurs de cette extension réside dans la manière dont elle traite les métadonnées utilisées en particulier dans les fichiers **SAS**, **SPSS** et **Stata**, à savoir les étiquettes de variable, les étiquettes de valeur et les valeurs manquantes déclarées. En effet, chaque fonction va importer ces métadonnées sous la forme d'attributs dont le nom diffère d'une fonction à l'autre. Par ailleurs, selon les options retenues, les variables labellisées seront parfois transformées ou non en facteurs. Enfin, `foreign` ne sait pas toujours importer les différents types de variables représentant des dates et des heures.

L'extension `haven` (qui fait partie du "tidyverse") tente de remédier à plusieurs des limitations rencontrées avec `foreign` :

- le format des métadonnées importé est uniforme, quel que soit le type de fichier source (**SAS**, **SPSS** ou **Stata**) ;
- les variables labellisées ne sont pas transformées en facteurs, mais héritent d'une nouvelle classe `haven_labelled`, la valeur initiale restant inchangée ;
- les différents formats de date sont convertis dans des classes R appropriées, utilisables en particulier avec `lubridate` ;

- `haven` peut lire les fichiers **SAS natifs** (extension `.sas7bdat`) ce que ne peut pas faire `foreign` ;
- `haven` peut lire les fichiers **Stata** 13 et 14, alors que `foreign` ne sait lire ces fichiers que jusqu'à la version 12 ;
- les tableaux de données produits ont directement la classe `tbl_df` ce qui permet d'utiliser directement les fonctionnalités de l'extension `dplyr`.

À noter, il est également possible d'utiliser l'interface graphique de **RStudio** pour l'import des fichiers **SPSS, Stata, SAS et Excel**.

#### IMPORTANT

##### Données labellisées

À la différence de `foreign`, `haven` ne convertit pas les variables avec des étiquettes de valeurs en facteurs mais en vecteurs labellisés du type `haven_labelled` qui sont présentés en détail dans le chapitre Facteurs et vecteurs labellisés, page 123 et dans le chapitre recodage de variables, page 202 où est discutée la question de savoir à quel moment convertir les données labellisées.

## SPSS

Les fichiers générés par **SPSS** sont de deux types : les fichiers **SPSS natifs** natifs (extension `.sav`) et les fichiers au format **SPSS export** (extension `.por`).

Dans les deux cas, on aura recours à la fonction `read_spss` :

```
R> library(haven)
donnees <- read_spss("data/fichier.sav", user_na = TRUE)
```

## IMPORTANT

### Gestion des valeurs manquantes

Dans **SPSS**, il est possible de définir des valeurs à considérées comme manquantes. Plus précisément jusqu'à 3 valeurs spécifiques et/ou les valeurs comprises entre un minimum et un maximum. Par défaut, `read_spss` convertit toutes ces valeurs en `NA` lors de l'import.

Or, il est parfois important de garder les différentes valeurs originelles, notamment dans le cadre de l'analyse de données d'enquête, un manquant du type «ne sait pas» n'étant pas équivalent à un manquant du type «refus» ou du type «variable non collectée».

Dès lors, nous vous recommandons d'appeler `read_spss` avec l'option `user_na = TRUE`. Dans ce cas-là, les valeurs manquantes définies dans **SPSS** ne seront pas converties en `NA`, tout en conservant la définition des valeurs définies comme manquantes. Il sera alors toujours possible de convertir, dans un second temps et en fonction des besoins, ces valeurs à considérer comme manquantes en `NA` grâce aux fonctions de l'extension `labelled`, en particulier `user_na_to_na`, `na_values` et `na_range`.

À noter que les fonctions `describe` et `freq` de l'extension `questionr` que nous aborderons dans d'autres chapitres savent exploiter ces valeurs à considérer comme manquantes.

### NOTE

Si vous préférez utiliser l'extension `foreign`, la fonction correspondante est `read.spss`. On indiquera à la fonction de renvoyer un tableau de données avec l'argument `to.data.frame = TRUE`.

Par défaut, les variables numériques pour lesquelles des étiquettes de valeurs ont été définies sont transformées en variables de type `facteur`, les étiquettes définies dans **SPSS** étant utilisées comme `labels` du facteur. De même, si des `valeurs manquantes` ont été définies dans **SPSS**, ces dernières seront toutes transformées en `NA` (**R** ne permettant pas de gérer plusieurs types de valeurs manquantes). Ce comportement peut être modifié avec `use.value.labels` et `use.missing`.

```
R> library(foreign)
donnees <- read.spss("data/fichier.sav", to.data.frame = TRUE, use.value.labels = FALSE, use.missing = FALSE)
```

Il est important de noter que `read.spss` de l'extension `foreign` ne sait pas importer les dates. Ces dernières sont donc automatiquement transformées en valeurs numériques.

**SPSS** stocke les dates sous la forme du nombre de secondes depuis le début du calendrier grégorien, à savoir le 14 octobre 1582. Dès lors, si l'on des dates dans un fichier **SPSS** et que ces dernières ont été converties en valeurs numériques, on pourra essayer la commande suivante :

```
R> donnees$date <- as.POSIXlt(donnees$date, origin = "1582-10-14")
```

## SAS

Les fichiers **SAS** se présentent en général sous deux format : format **SAS export** (extension `.xport` ou `.xpt`) ou format **SAS natif** (extension `.sas7bdat`).

Les fichiers **SAS natifs** peuvent être importées directement avec `read_sas` de l'extension `haven` :

```
R> library(haven)
donnees <- read_sas("data/fichier.sas7bdat")
```

Au besoin, on pourra préciser en deuxième argument le nom d'un fichier **SAS catalogue** (extension `.sas7bcat`) contenant les métadonnées du fichier de données.

```
R> library(haven)
donnees <- read_sas("data/fichier.sas7bdat", "data/fichier.sas7bcat")
```

Les fichiers au format **SAS export** peuvent être importés via la fonction `read.xport` de l'extension `foreign`. Celle-ci s'utilise très simplement, en lui passant le nom du fichier en argument :

```
R> library(foreign)
donnees <- read.xport("data/fichier.xpt")
```

## Stata

Pour les fichiers **Stata** (extension `.dta`), on aura recours aux fonctions `read_dta` et `read_stata` de l'extension `haven`. Ces deux fonctions sont identiques.

```
R> library(haven)
donnees <- read_dta("data/fichier.dta")
```

### IMPORTANT

#### Gestion des valeurs manquantes

Dans **Stata**, il est possible de définir plusieurs types de valeurs manquantes, qui sont notées sous la forme `.a` à `.z`. Pour conserver cette information lors de l'import, `haven` a introduit dans R le concept de `tagged NA` ou `tagged missing value`. Plus de détails sur ces données manquantes «étiquetées», on se référera à la page d'aide de la fonction `tagged_na`.

## NOTE

Si l’on préfère utiliser l’extension **foreign**, on aura recours à la fonction `read.dta`.

L’option `convert.factors` indique si les variables labellisées doivent être converties automatiquement en facteurs. Pour un résultat similaire à celui de **haven**, on choisira donc :

```
R> library(foreign)
donnees <- read.dta("data/fichier.dta", convert.factors = FALSE)
```

L’option `convert.dates` permet de convertir les dates du format **Stata** dans un format de dates géré par **R**. Cependant, cela ne marche pas toujours. Dans ces cas là, l’opération suivante peut fonctionner. Sans garantie néanmoins, il est toujours vivement conseillé de vérifier le résultat obtenu !

```
R> donnees$date <- as.Date(donnees$Date, origin = "1960-01-01")
```

## Excel

Une première approche pour importer des données **Excel** dans **R** consiste à les exporter depuis **Excel** dans un fichier texte (texte tabulé ou **CSV**) puis de suivre la procédure d’importation d’un fichier texte.

Une feuille **Excel** peut également être importée directement avec l’extension **readxl** qui appartient à la même famille que **haven** et **readr**.

La fonction `read_excel` permet d’importer à la fois des fichiers `.xls` (**Excel** 2003 et précédents) et `.xlsx` (**Excel** 2007 et suivants).

```
R> library(readxl)
donnees <- read_excel("data/fichier.xlsx")
```

Une seule feuille de calculs peut être importée à la fois. On pourra préciser la feuille désirée avec `sheet` en indiquant soit le nom de la feuille, soit sa position (première, seconde, ...).

```
R> donnees <- read_excel("data/fichier.xlsx", sheet = 3)
donnees <- read_excel("data/fichier.xlsx", sheet = "mes_donnees")
```

On pourra préciser avec `col_names` si la première ligne contient le nom des variables.

Par défaut, `read_excel` va essayer de deviner le type (numérique, textuelle, date) de chaque colonne. Au besoin, on pourra indiquer le type souhaité de chaque colonne avec `col_types`.

RStudio propose également pour les fichiers **Excel** un assistant d'importation, similaire à celui pour les fichiers texte, permettant de faciliter l'import.

#### NOTE

Une première alternative est l'extension `openxlsx` et sa fonction `read.xlsx`.

Une seconde alternative est l'extension `xlsx` qui propose deux fonctions différentes pour importer des fichiers **Excel**: `read.xlsx` et `read.xlsx2`. La finalité est la même mais leur fonctionnement interne est différent. En cas de difficultés d'import, on pourra tester l'autre. Il est impératif de spécifier la position de la feuille de calculs que l'on souhaite importer.

```
R> library(xlsx)
donnees <- read.xlsx("data/fichier.xlsx", 1)
```

## dBase

L'Insee et d'autres producteur de données diffusent leurs fichiers au format **dBase** (extension `.dbf`). Ceux-ci sont directement lisibles dans R avec la fonction `read.dbf` de l'extension `foreign`.

```
R> library(foreign)
donnees <- read.dbf("data/fichier.dbf")
```

La principale limitation des fichiers **dBase** est de ne pas gérer plus de 256 colonnes. Les tables des enquêtes de l'Insee sont donc parfois découpées en plusieurs fichiers `.dbf` qu'il convient de fusionner avec la fonction `merge`. L'utilisation de cette fonction est détaillée dans le chapitre sur la fusion de tables, page 269.

## Feuille de calcul Google Sheets

Pour importer des données stockées sous formes de feuilles de calcul **Google**, on pourra se référer à l'extension `googlesheets`.

# Données spatiales

## Shapefile

Les fichiers **Shapefile** sont couramment utilisés pour échanger des données géoréférencées. La majorité des logiciels de **SIG** (systèmes d’informations géographiques) sont en capacité d’importer et d’exporter des données dans ce format.

Un **shapefile** contient toute l’information liée à la géométrie des objets décrits, qui peuvent être :

- des points
- des lignes
- des polygones

Son extension est classiquement `.shp` et il est toujours accompagné de deux autres fichiers de même nom et d’extensions :

- un fichier `.dbf`, qui contient les données attributaires relatives aux objets contenus dans le **shapefile**
- un fichier `.shx`, qui stocke l’index de la géométrie

D’autres fichiers peuvent également être fournis :

- `.sbn` et `.sbx` - index spatial des formes.
- `.fbn` et `.fbx` - index spatial des formes pour les shapefile en lecture seule
- `.ain` et `.aih` - index des attributs des champs actifs dans une table ou dans une table d’attributs du thème
- `.prj` - information sur le système de coordonnées
- `.shp.xml` - métadonnées du shapefile.
- `.atx` - fichier d’index des attributs pour le fichier `.dbf`
- `.qix`

En premier lieu, il importe que tous les fichiers qui compose un même **shapefile** soit situés dans le même répertoire et aient le même nom (seule l’extension étant différente).

L’extension **maptools** fournit les fonctions permettant d’importer un **shapefile** dans **R**. Le résultat obtenu utilisera l’une des différentes classes spatiales fournies par l’extension **sp**.

La fonction générique d’import est `readShapeSpatial` :

```
R> library(maptools)
donnees_spatiales <- readShapeSpatial("data/fichier.shp")
```

Si l'on connaît déjà le type de données du **shapefile** (points, lignes ou polygones), on pourra utiliser directement `readShapePoints`, `readShapeLines` ou `readShapePoly`.

## Rasters

Il existe de multiples formats pour stocker des données matricielles spatiales. L'un des plus communs est le format **ASCII grid** aussi connu sous le nom de **Arc/Info ASCII grid** ou **ESRI grid**. L'extension de ce format n'est pas toujours uniforme. On trouve parfois `.asc` ou encore `.ag` voir même `.txt`.

Pour importer ce type de fichier, on pourra avoir recours à la fonction `readAsciiGrid` de l'extension `maptools`. Le résultat sera, par défaut, au format `SpatialGridDataFrame` de l'extension `sp`.

```
R> library(maptools)
donnees_spatiales <- readAsciiGrid("data/fichier.asc")
```

L'extension `raster` permet d'effectuer de multiples manipulations sur les données du type `raster`. Elle est en capacité d'importer des données depuis différents formats (plus précisément les formats pris en charge par la librairie **GDAL**, <http://www.gdal.org/>).

De plus, les fichiers raster pouvant être particulièrement volumineux (jusqu'à plusieurs Go de données), l'extension `raster` est capable de travailler sur un fichier raster sans avoir à le charger intégralement en mémoire.

Pour plus d'informations, voir les fonctions `raster` et `getValues`.

## Connexion à des bases de données

### Interfaçage via l'extension DBI

R est capable de s'interfacer avec différents systèmes de bases de données relationnelles, dont **SQLite**, **MS SQL Server**, **PostgreSQL**, **MariaDB**, etc.

Pour illustrer rapidement l'utilisation de bases de données, on va créer une base SQLite d'exemple à l'aide du code R suivant, qui copie la table du jeu de données `mtcars` dans une base de données `bdd.sqlite` :

```
[1] TRUE
```

```
R> library(DBI)
library(RSQLite)
con <- DBI::dbConnect(RSQLite::SQLite(), dbname = "bdd.sqlite")
data(mtcars)
mtcars$name <- rownames(mtcars)
dbWriteTable(con, "mtcars", mtcars)
dbDisconnect(con)
```

Si on souhaite se connecter à cette base de données par la suite, on peut utiliser l’extension **DBI**, qui propose une interface générique entre \*\*R// et différents systèmes de bases de données. On doit aussi avoir installé et chargé l’extension spécifique à notre base, ici **RSQLite**. On commence par ouvrir une connexion à l’aide de la fonction `dbConnect` de **DBI** :

```
R> library(DBI)
library(RSQLite)
con <- DBI::dbConnect(RSQLite::SQLite(), dbname = "bdd.sqlite")
```

La connexion est stockée dans un objet `con`, qu’on va utiliser à chaque fois qu’on voudra interroger la base.

On peut vérifier la liste des tables présentes et les champs de ces tables avec `dbListTables` et `dbListFields` :

```
R> dbListTables(con)
```

```
[1] "mtcars"
```

```
R> dbListFields(con, "mtcars")
```

```
[1] "mpg"   "cyl"   "disp"  "hp"    "drat"  "wt"    "qsec" "vs"
[9] "am"    "gear"  "carb"  "name"
```

On peut également lire le contenu d’une table dans un objet de notre environnement avec `dbReadTable` :

```
R> cars <- dbReadTable(con, "mtcars")
```

On peut également envoyer une requête SQL directement à la base et récupérer le résultat avec `dbGetQuery` :

```
R> dbGetQuery(con, "SELECT * FROM mtcars WHERE cyl = 4")
```

Enfin, quand on a terminé, on peut se déconnecter à l'aide de `dbDisconnect` :

```
R> dbDisconnect(con)
```

Ceci n'est évidemment qu'un tout petit aperçu des fonctionnalités de **DBI**.

## Utilisation de **dplyr** et **dbplyr**

L'extension **dplyr** est dédiée à la manipulation de données, elle est présentée dans un chapitre dédié, page 233. En installant l'extension complémentaire **dbplyr**, on peut utiliser **dplyr** directement sur une connection à une base de données générée par **DBI** :

```
R> library(DBI)
library(RSQLite)
library(dplyr)
con <- DBI::dbConnect(RSQLite::SQLite(), dbname = "bdd.sqlite")
```

La fonction `tbl` notamment permet de créer un nouvel objet qui représente une table de la base de données :

```
R> cars_tbl <- tbl(con, "mtcars")
```

### IMPORTANT

Ici l'objet `cars_tbl` n'est *pas* un tableau de données, c'est juste un objet permettant d'interroger la table de notre base de données.

On peut utiliser cet objet avec les verbes, page 233 de **dplyr** :

```
R> cars_tbl %>%
  filter(cyl == 4) %>%
  select(name, mpg, cyl)
```

**dbplyr** s'occupe, de manière transparente, de transformer les instructions **dplyr** en requête SQL, d'interroger la base de données et de renvoyer le résultat. De plus, tout est fait pour qu'un minimum d'opérations sur la base, parfois coûteuses en temps de calcul, ne soient effectuées.

### IMPORTANT

Il est possible de modifier des objets de type `tbl`, par exemple avec `mutate` :

```
cars_tbl <- cars_tbl %>% mutate(type = "voiture")
```

Dans ce cas la nouvelle colonne `type` est bien créée et on peut y accéder par la suite. Mais **cette création se fait dans une table temporaire** : elle n'existe que le temps de la connexion à la base de données. À la prochaine connexion, cette nouvelle colonne n'apparaîtra pas dans la table.

Bien souvent on utilisera une base de données quand les données sont trop volumineuses pour être gérées par un ordinateur de bureau. Mais si les données ne sont pas trop importantes, il sera toujours plus rapide de récupérer l'intégralité de la table dans notre session R pour pouvoir la manipuler comme les tableaux de données habituels. Ceci se fait grâce à la fonction `collect` de `dplyr` :

```
R> cars <- cars_tbl %>% collect()
```

Ici, `cars` est bien un tableau de données classique, copie de la table de la base au moment du `collect`.

Et dans tous les cas, on n'oubliera pas de se déconnecter avec :

```
R> dbDisconnect(con)
```

## Ressources

Pour plus d'informations, voir la [documentation très complète](#) (en anglais) proposée par RStudio.

Par ailleurs, depuis la version 1.1, RStudio facilite la connexion à certaines bases de données grâce à l'onglet *Connections*. Pour plus d'informations on pourra se référer à l'article (en anglais) [Using RStudio Connections](#).

## Autres sources

R offre de très nombreuses autres possibilités pour accéder aux données. Il est ainsi possible d'importer des données depuis d'autres applications qui n'ont pas été évoquées (**Epi Info**, **S-Plus**, etc.), de lire des données via **ODBC** ou des connexions réseau, etc.

Pour plus d'informations on consultera le manuel *R Data Import/Export* :  
<http://cran.r-project.org/manuals.html>.

La section [Database Management](#) du site [Awesome R](#) fournit également une liste d'extensions permettant de s'interfacer avec différents gestionnaires de bases de données.

## Sauver ses données

R dispose également de son propre format pour sauvegarder et échanger des données. On peut sauver n'importe quel objet créé avec R et il est possible de sauver plusieurs objets dans un même fichier. L'usage est d'utiliser l'extension `.RData` pour les fichiers de données R. La fonction à utiliser s'appelle tout simplement `save`.

Par exemple, si l'on souhaite sauvegarder son tableau de données `d` ainsi que les objets `tailles` et `poids` dans un fichier `export.RData` :

```
R> save(d, tailles, poids, file = "export.RData")
```

À tout moment, il sera toujours possible de recharger ces données en mémoire à l'aide de la fonction `load` :

```
R> load("export.RData")
```

### IMPORTANT

Si entre temps vous aviez modifié votre tableau `d`, vos modifications seront perdues. En effet, si lors du chargement de données, un objet du même nom existe en mémoire, ce dernier sera remplacé par l'objet importé.

La fonction `save.image` est un raccourci pour sauvegarder tous les objets de la session de travail dans le fichier `.RData` (un fichier un peu étrange car il n'a pas de nom mais juste une extension). Lors de la fermeture de RStudio, il vous sera demandé si vous souhaitez enregistrer votre session. Si vous répondez Oui, c'est cette fonction `save.image` qui sera appliquée.

```
R> save.image()
```

```
[1] FALSE
```



# Où trouver de l'aide ?

Aide en ligne .....	164
Aide sur une fonction .....	164
Naviguer dans l'aide .....	165
Ressources sur le Web .....	166
Moteur de recherche .....	166
Aide en ligne .....	166
Antisèches .....	166
Ressources officielles .....	167
Ouvrages, blogs, MOOCs .....	167
Revue .....	170
Ressources francophones .....	170
RStudio .....	171
Où poser des questions ? .....	171
Les forums d'analyse-R .....	171
Discussion instantanée .....	171
Liste R-soc .....	172
Liste Quanti .....	172
StackOverflow .....	172
Forum Web en français .....	172
Canaux IRC (chat) .....	173
Listes de discussion officielles .....	173
Où trouver des extensions intéressantes ? .....	174

## NOTE

La version originale de ce chapitre a été écrite par Julien Barnier dans le cadre du support de cours [Introduction à R](#).

## Aide en ligne

R dispose d’une aide en ligne très complète, mais dont l’usage n’est pas forcément très simple. D’une part car elle est intégralement en anglais, d’autre part car son organisation prend un certain temps à être maîtrisée.

### Aide sur une fonction

La fonction la plus utile est sans doute `help` (ou son équivalent `?`) qui permet d’afficher la page d’aide liée à une ou plusieurs fonctions. Celle-ci permet de lister les arguments de la fonction, d’avoir des informations détaillées sur son fonctionnement, les résultats qu’elle retourne, etc.

Pour accéder à l’aide de la fonction `mean`, par exemple, il vous suffit de saisir directement :

```
R> ?mean
```

ou bien

```
R> help("mean")
```

Sous **RStudio**, la page d’aide correspondante s’affichera sous l’onglet *Help* dans le quadrant inférieur droit.

Chaque page d’aide comprend plusieurs sections, en particulier :

Section	Contenu
Description	donne un résumé en une phrase de ce que fait la fonction
Usage	indique la ou les manières de l'utiliser
Arguments	détaille tous les arguments possibles et leur signification
Value	indique la forme du résultat renvoyé par la fonction
Details	apporte des précisions sur le fonctionnement de la fonction
Note	pour des remarques éventuelles
References	pour des références bibliographiques ou des URL associées
See Also	très utile, renvoie vers d'autres fonctions semblables ou liées, ce qui peut être très utile pour découvrir ou retrouver une fonction dont on a oublié le nom
Examples	série d'exemples d'utilisation

Les exemples peuvent être directement exécutés en utilisant la fonction `example` :

```
R> example(mean)
```

```
mean> x <- c(0:10, 50)
mean> xm <- mean(x)

mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50
```

## Naviguer dans l'aide

La fonction `help.start` permet d'afficher le sommaire de l'aide en ligne. Saisissez simplement :

```
R> help.start()
```

Si vous souhaitez rechercher quelque chose dans le contenu de l'aide, vous pouvez utiliser la fonction `help.search` (ou `??` qui est équivalente) qui renvoie une liste des pages d'aide contenant les termes recherchés.

Par exemple :

```
R> help.search("logistic")
```

ou

```
R> ??logistic
```

pour rechercher les pages de l’aide qui contiennent le terme *logistic*.

## Ressources sur le Web

De nombreuses ressources existent en ligne, mais la plupart sont en anglais.

### Moteur de recherche

Le fait que le logiciel s’appelle **R** ne facilite malheureusement pas les recherches sur le Web... La solution à ce problème a été trouvée grâce à la constitution d’un moteur de recherche *ad hoc* à partir de **Google**, nommé **Rseek** :

<http://www.rseek.org/>.

Les requêtes saisies dans **Rseek** sont exécutées dans des corpus prédéfinis liés à **R**, notamment les documents et manuels, les listes de discussion ou le code source du programme.

Les requêtes devront cependant être formulées en anglais.

### Aide en ligne

Les sites **RDocumentation** et **rdrr.io** proposent un accès clair et rapide à la documentation de **R** et des extensions hébergées sur le **CRAN** (ainsi que certaines extensions hébergées sur **GitHub**). Il permettent notamment de rechercher et naviguer facilement entre les pages des différentes fonctions :

- <http://www.rdocumentation.org/>
- <https://rdrr.io/>

### Antisèches

**RStudio** propose plusieurs *cheat sheets* (antisèches) en anglais qui proposent sur deux pages une synthèse compacte de fonctions et de leur usage selon différentes thématiques, notamment :

- Manipulation des données avec `dplyr`

- [Visualisation avec ggplot2](#)
- [Export avec RMarkdown](#)
- etc.

La liste complète est disponible en ligne :

<https://www.rstudio.com/resources/cheatsheets/>

ou directement depuis RStudio, via le menu *Help*, puis *Cheatsheets*.

À noter que certaines antisèches sont également disponibles en français.

## Ressources officielles

La documentation officielle de R est accessible en ligne depuis le site du projet :

<http://www.r-project.org/>.

Les liens de l'entrée *Documentation* du menu de gauche vous permettent d'accéder à différentes ressources.

### Manuels

Les *manuels* sont des documents complets de présentation de certains aspects de R. Ils sont accessibles en ligne, ou téléchargeables au format PDF :

<http://cran.r-project.org/manuals.html>.

On notera plus particulièrement *An introduction to R*, normalement destiné aux débutants, mais qui nécessite quand même un minimum d'aisance en informatique et en statistiques :

<http://cran.r-project.org/doc/manuals/R-intro.html>.

*R Data Import/Export* explique notamment comment importer des données depuis d'autres logiciels :  
<http://cran.r-project.org/doc/manuals/R-data.html>.

## Ouvrages, blogs, MOOCs...

### Francophones

Parmi les ressources en français, on peut citer notamment [R et espace](#), manuel d'initiation à la programmation avec R appliquée à l'analyse de l'information géographique, librement téléchargeable en ligne.

La très bonne *Introduction à R et au tidyverse* de Julien Barnier disponible sur <https://juba.github.io/tidyverse/>.

La section [Contributed documentation](#) du site officiel de R contient également des liens vers différents documents en français, plus ou moins accessibles et plus ou moins récemment mis à jour.

Le pôle bioinformatique lyonnais (PBIL) propose depuis longtemps une somme très importante de documents, qui comprend des cours complets de statistiques utilisant R :

- <http://pbil.univ-lyon1.fr/R/>

Plusieurs blogs francophones autour de R sont également actifs, parmi lesquels :

- [ElementR](#), le blog du groupe du même nom, qui propose de nombreuses ressources sur R en général et en particulier sur la cartographie ou l’analyse de réseaux.
- [R-atique](#), blog animé par Lise Vaudor, propose régulièrement des articles intéressants et accessibles sur des méthodes d’analyse ou sur des extensions R.

Enfin, le site *France Université Numérique* propose régulièrement des sessions de cours en ligne, parmi lesquels une [Introduction à la statistique avec R](#) et un cours sur [l’Analyse des données multidimensionnelles](#).

On peut aussi citer :

- [Logiciel R et programmation](#) par @3wen
- [Programmer en R](#), wikibook collaboratif (licence CC-BY-SA)
- [R et espace](#) : manuel d’initiation à la programmation avec R appliqué à l’analyse de l’information géographique, librement téléchargeable en ligne.
- [Introduction à la programmation en R](#)
- [C'est l'enfeR](#) donne des exemples de code simple avec R qui donnent des résultats surprenants.

Et enfin, une liste de ressources francophones : <https://github.com/frrrenchies/frrrenchies/>.

## Anglophones

Les ressources anglophones sont évidemment très nombreuses.

On citera essentiellement l’ouvrage en ligne [R for data science](#), très complet, et qui fournit une introduction très complète et progressive à R, et aux packages du *tidyverse*. Il existe également en version papier.

Pour aborder des aspects beaucoup plus avancés, l’ouvrage également en ligne [Advanced R](#), d’Hadley Wickham, est extrêmement bien fait et très complet.

On notera également l’existence du [R journal](#), revue en ligne consacrée à R, et qui propose régulièrement des articles sur des méthodes d’analyse, des extensions, et l’actualité du langage.

La plateforme [R-bloggers](#) agrège les contenus de plusieurs centaines de blogs parlant de R, très pratique pour suivre l’actualité de la communauté.

Enfin, sur Twitter, les échanges autour de R sont regroupés autour du hashtag [#rstats](#).

On peut aussi citer le site **Awesome R** (<https://awesome-r.com/>) qui fournit une liste d'extensions choisies et triées par thématique et le site **R Data Science Tutorials** qui recense des tutoriels en anglais.

## FAQ

Les *FAQ* (*frequently asked questions*) regroupent des questions fréquemment posées et leurs réponses. À lire donc ou, au moins, à parcourir avant toute chose :

<http://cran.r-project.org/faqs.html>.

La FAQ la plus utile est la FAQ généraliste sur **R** :

<http://cran.r-project.org/doc/FAQ/R-FAQ.html>.

Mais il existe également une FAQ dédiée aux questions liées à **Windows** et une autre à la plateforme **Mac OS X**.

### NOTE

Les manuels et les FAQ sont accessibles même si vous n'avez pas d'accès à Internet en utilisant la fonction `help.start` décrite précédemment.

## R-announce

**R-announce** est la liste de diffusion électronique officielle du projet. Elle ne comporte qu'un nombre réduit de messages (quelques-uns par mois tout au plus) et diffuse les annonces concernant de nouvelles versions de R ou d'autres informations particulièrement importantes. On peut s'y abonner à l'adresse suivante :

<https://stat.ethz.ch/mailman/listinfo/r-announce>

## R Journal

**R Journal** est la « revue » officielle du projet **R**, qui a succédé début 2009 à la lettre de nouvelles **R News**. Elle paraît entre deux et cinq fois par an et contient des informations sur les nouvelles versions du logiciel, des articles présentant des extensions, des exemples d'analyse... Les parutions sont annoncées sur la liste de diffusion **R-announce** et les numéros sont téléchargeables à l'adresse suivante :

<http://journal.r-project.org/>.

## Autres documents

On trouvera de nombreux documents dans différentes langues, en général au format **PDF**, dans le

répertoire suivant :

<http://cran.r-project.org/doc/contrib/>.

Parmi ceux-ci, les cartes de référence peuvent être très utiles, ce sont des aides-mémoire recensant les fonctions les plus courantes :

<http://cran.r-project.org/doc/contrib/Short-refcard.pdf>

On notera également un document d’introduction en anglais progressif et s’appuyant sur des méthodes statistiques relativement simples :

<http://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf>

Pour les utilisateurs déjà habitués à **SAS** ou **SPSS**, le livre *R for SAS and SPSS Users* et le document gratuit qui en est tiré peuvent être de bonnes ressources, tout comme le site web **Quick-R** :

<http://rforsasandspssusers.com/> et <http://www.statmethods.net/>.

## Revue

La revue *Journal of Statistical Software* est une revue électronique anglophone, dont les articles sont en accès libre, et qui traite de l’utilisation de logiciels d’analyse de données dans un grand nombre de domaines. De nombreux articles (la majorité) sont consacrés à **R** et à la présentation d’extensions plus ou moins spécialisées.

Les articles qui y sont publiés prennent souvent la forme de tutoriels plus ou moins accessibles mais qui fournissent souvent une bonne introduction et une ressource riche en informations et en liens.

Adresse de la revue :

<http://www.jstatsoft.org/>

## Ressources francophones

Il existe des ressources en français sur l’utilisation de **R**, mais peu sont réellement destinées aux débutants, elles nécessitent en général des bases à la fois en informatique et en statistique.

Le document le plus abordable et le plus complet est sans doute *R pour les débutants*, d’Emmanuel Paradis, accessible au format **PDF** :

[http://cran.r-project.org/doc/contrib/Paradis-rdebuts\\_fr.pdf](http://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf).

La somme de documentation en français la plus importante liée à **R** est sans nulle doute celle mise à disposition par le Pôle bioinformatique lyonnais. Leur site propose des cours complets de statistique utilisant **R** :

<http://pbil.univ-lyon1.fr/R/enseignement.html>.

La plupart des documents sont assez pointus niveau mathématique et plutôt orientés biostatistique, mais on trouvera des documents plus introductifs ici :

<http://pbil.univ-lyon1.fr/R/html/cours1>.

Dans tous les cas la somme de travail et de connaissances mise à disposition librement est impressionnante... Enfin, le site de Vincent Zoonekynd ([http://zoonek2.free.fr/UNIX/48\\_R\\_2004/all.html](http://zoonek2.free.fr/UNIX/48_R_2004/all.html)) comprend de nombreuses notes prises au cours de sa découverte du logiciel. On notera cependant que l'auteur est normalien et docteur en mathématiques...

## RStudio

La documentation officielle de **RStudio** est disponible sur <https://support.rstudio.com> (catégorie *Documentation* disponible en milieu de page).

## Où poser des questions ?

La communauté des utilisateurs de **R** est très active et en général très contente de pouvoir répondre aux questions (nombreuses) des débutants et à celles (tout aussi nombreuses) des utilisateurs plus expérimentés. Dans tous les cas, les règles de base à respecter avant de poser une question sont toujours les mêmes : avoir cherché soi-même la réponse auparavant, notamment dans les FAQ et dans l'aide en ligne, et poser sa question de la manière la plus claire possible, de préférence avec un exemple de code posant problème.

## Les forums d'analyse-R

En premier lieu (autopromotion oblige), chaque chapitre du site d'**analyse-R** (<http://larmarange.github.io/analyse-R/>) comporte en bas de page une fonctionnalité permettant de laisser des commentaires. On peut donc y poser une question en lien avec le chapitre concerné.

## Discussion instantanée

**Grrr** ("pour quand votre R fait Grrr") est un groupe Slack (plateforme de discussion instantanée) francophone dédié aux échanges et à l'entraide autour de R. Il est ouvert à tous et se veut accessible aux débutants. Vous pouvez même utiliser un pseudonyme si vous préférez.

Pour rejoindre la discussion, il suffit de suivre le lien d'invitation suivant :

[https://join.slack.com/t/r-grrr/shared\\_invite/zt-46utbgb9-uvo\\_bg5cbuxOV~H10YUX8w](https://join.slack.com/t/r-grrr/shared_invite/zt-46utbgb9-uvo_bg5cbuxOV~H10YUX8w)

## Liste R-soc

Une liste de discussion a été créée spécialement pour permettre aide et échanges autour de l'utilisation de R en sciences sociales. Elle est hébergée par RENATER et on peut s'y abonner à l'adresse suivante : <https://groupes.renater.fr/sympa/subscribe/r-soc>.

Grâce aux services offerts par le site [gmane.org](http://gmane.org), la liste est également disponible sous d'autres formes (forum Web, blog, **NNTP**, flux **RSS**) permettant de lire et de poster sans avoir à s'inscrire et à recevoir les messages sous forme de courrier électronique. Pour plus d'informations : <http://dir.gmane.org/gmane.comp.lang.r.user.french>.

## Liste Quanti

La liste **Quanti** également hébergée par Renater a pour vocation d'accueillir les contributions et les échanges de toutes celles et tous ceux qui s'intéressent aux questions d'enseignement des méthodes quantitatives dans les sciences sociales. Cela inclue l'usage de R.

Pour s'abonner : [https://groupes.renater.fr/sympa/subscribe/quanti?previous\\_action=info](https://groupes.renater.fr/sympa/subscribe/quanti?previous_action=info)

## StackOverflow

Le site **StackOverflow** (qui fait partie de la famille des sites **StackExchange**) comprend une section (anglophone) dédiée à R qui permet de poser des questions et en général d'obtenir des réponses assez rapidement :

<http://stackoverflow.com/questions/tagged/r>.

La première chose à faire, évidemment, est de vérifier que sa question n'a pas déjà été posée.

## Forum Web en français

Le Cirad a mis en ligne un forum dédié aux utilisateurs de R, très actif :

<http://forums.cirad.fr/logiciel-R/index.php>.

Les questions diverses et variées peuvent être posées dans la rubrique *Questions en cours* :

<http://forums.cirad.fr/logiciel-R/viewforum.php?f=3>.

Il est tout de même conseillé de faire une recherche rapide sur le forum avant de poser une question, pour voir si la réponse ne s'y trouverait pas déjà.

## Canaux IRC (chat)

L'**IRC**, ou *Internet Relay Chat* est le vénérable ancêtre toujours très actif des messageries instantanées actuelles. Un canal (en anglais) est notamment dédié aux échanges autour de **R** (`#R`).

Si vous avez déjà l'habitude d'utiliser **IRC**, il vous suffit de pointer votre client préféré sur **Freenode** (`irc.freenode.net`) puis de rejoindre l'un des canaux en question.

Sinon, le plus simple est certainement d'utiliser l'interface web de **Mibbit**, accessible à l'adresse <http://www.mibbit.com/>.

Dans le champ *Connect to IRC*, sélectionnez *Freenode.net*, puis saisissez un pseudonyme dans le champ *Nick* et `#R` dans le champ *Channel*. Vous pourrez alors discuter directement avec les personnes présentes.

Le canal `#R` est normalement peuplé de personnes qui seront très heureuses de répondre à toutes les questions, et en général l'ambiance y est très bonne. Une fois votre question posée, n'hésitez pas à être patient et à attendre quelques minutes, voire quelques heures, le temps qu'un des habitués vienne y faire un tour.

## Listes de discussion officielles

La liste de discussion d'entraide (par courrier électronique) officielle du logiciel **R** s'appelle **R-help**. On peut s'y abonner à l'adresse suivante, mais il s'agit d'une liste avec de nombreux messages :  
<https://stat.ethz.ch/mailman/listinfo/r-help>.

Pour une consultation ou un envoi ponctuels, le mieux est sans doute d'utiliser les interfaces Web fournies par **gmane.org** :

<http://blog.gmane.org/gmane.comp.lang.r.general>.

**R-help** est une liste avec de nombreux messages, suivie par des spécialistes de **R**, dont certains des développeurs principaux. Elle est cependant à réservier aux questions particulièrement techniques qui n'ont pas trouvé de réponses par d'autres biais.

Dans tous les cas, il est nécessaire avant de poster sur cette liste de bien avoir pris connaissance du *posting guide* correspondant :

<http://www.r-project.org/posting-guide.html>.

Plusieurs autres listes plus spécialisées existent également, elles sont listées à l'adresse suivante :

<http://www.r-project.org/mail.html>.

## Où trouver des extensions intéressantes ?

Il existe plusieurs milliers d’extensions pour R, et il n’est pas toujours facile de savoir laquelle choisir pour une tâche donnée.

Si un des meilleurs moyens reste le bouche à oreille, on peut aussi se reporter à la page [CRAN Task view](#) qui liste un certain nombre de domaines (classification, sciences sociales, séries temporelles...) et indique, pour chacun d’entre eux, une liste d’extensions potentiellement intéressantes accompagnées d’une courte description. On peut même installer l’ensemble des extensions d’une catégorie avec la fonction `install.views()`.

Une autre possibilité est de consulter la page [listant l’ensemble des packages existant](#). S’il n’est évidemment pas possible de passer en revue les milliers d’extensions une à une, on peut toujours effectuer une recherche dans la page avec des mots-clés correspondant aux fonctionnalités recherchées.

Un autre site intéressant est [Awesome R](#), une liste élaborée collaborativement des extensions les plus utiles ou les plus populaires classées par grandes catégories : manipulation des données, graphiques interactifs, etc.

La page [frrrenchies](#) liste des packages pouvant être utiles pour des utilisateurs français (géolocalisation, traitement du langage, accès à des API...), ainsi que des ressources francophones.

Enfin, certaines extensions fournissent des “galeries” permettant de repérer ou découvrir certains packages. C’est notamment le cas de `htmlwidgets`, qui propose une [galerie d’extensions proposant des graphiques interactifs](#), ou de [R Markdown](#).

# Visualiser ses données

Inspection visuelle des données . . . . .	175
summary . . . . .	177
str . . . . .	178
glimpse (dplyr) . . . . .	179
look_for (labelled) . . . . .	180
describe (questionr) . . . . .	184
Addins de codebook . . . . .	189
skim (skimr) . . . . .	190
create_report (DataExplorer) . . . . .	195
makeCodebook (dataMaid) . . . . .	195

Au fil des différents chapitres, nous avons abordé diverses fonctions utiles au quotidien et permettant de visualiser ses données. Ce chapitre se propose de les regrouper.

Chargeons tout d'abord quelques fichiers de données à titre d'exemple.

```
R> library(questionr)
    data(hdv2003)
    data(rp99)
    data(fecondite)
```

## Inspection visuelle des données

La particularité de **R** par rapport à d'autres logiciels comme **Modalisa** ou **SPSS** est de ne pas proposer, par défaut, de vue des données sous forme de tableau. Ceci peut parfois être un peu déstabilisant dans les premiers temps d'utilisation, même si l'on perd vite l'habitude et qu'on finit par se rendre compte que « voir » les données n'est pas forcément un gage de productivité ou de rigueur dans le traitement.

Néanmoins, **R** propose une interface permettant de visualiser le contenu d'un tableau de données à l'aide de la fonction `View` :

```
R> View(hdv2003)
```

Sous **RStudio**, on peut aussi afficher la visionneuse (*viewer*) en cliquant sur la petite icône en forme de tableau située à droite de la ligne d'un tableau de données dans l'onglet *Environment* du quadrant supérieur droit (cf. figure ci-après).

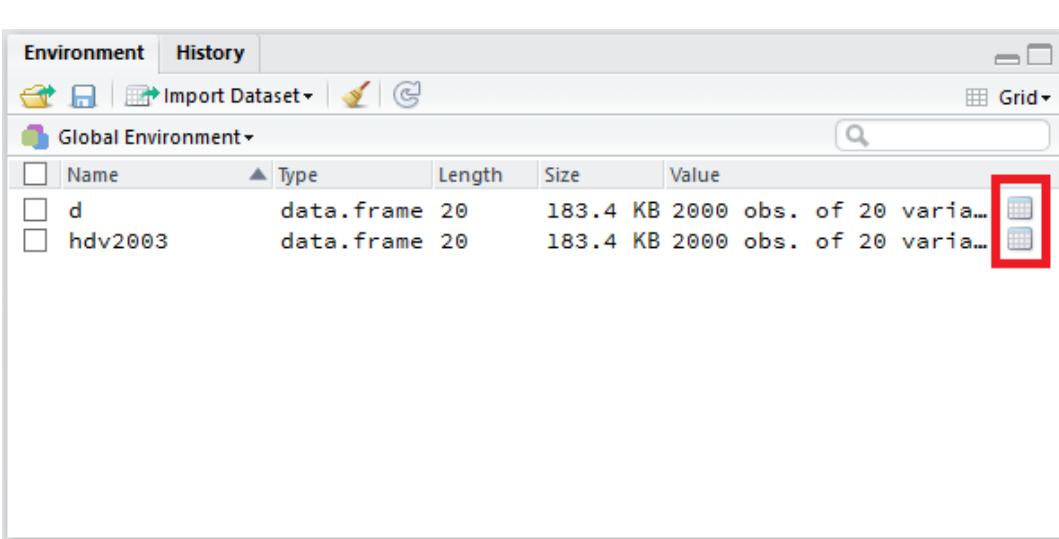


Figure 1. Icône pour afficher une vue du contenu d'un tableau

Dans tous les cas, **RStudio** lancera le *viewer* dans un onglet dédié dans le quadrant supérieur gauche. Le visualiseur de **RStudio** est plus avancé que celui-de base fournit par **R**. Il est possible de trier les données selon une variable en cliquant sur le nom de cette dernière. Il y a également un champs de recherche et un bouton *Filter* donnant accès à des options de filtrage avancées.

	<b>id</b>	<b>age</b>	<b>sexe</b>	<b>nivetud</b>	<b>poids</b>	<b>occup</b>	<b>qualif</b>	<b>freres.soeurs</b>	<b>ciso</b>
1	1	28	Femme	Enseignement supérieur y compris technique supérieur	2634.3982	Exerce une profession	Employe	8	Oui
2	2	23	Femme	NA	9738.3958	Etudiant, eleve	NA	2	Oui
3	3	59	Homme	Dernière année d'études primaires	3994.1025	Exerce une profession	Technicien	2	Non
4	4	34	Homme	Enseignement supérieur y compris technique supérieur	5731.6615	Exerce une profession	Technicien	1	Non
5	5	71	Femme	Dernière année d'études primaires	4329.0940	Retraite	Employe	0	Oui
6	6	35	Femme	Enseignement technique ou professionnel court	8674.6994	Exerce une profession	Employe	5	Non
7	7	60	Femme	Dernière année d'études primaires	6165.8035	Au foyer	Ouvrier qualifié	1	Oui
8	8	47	Homme	Enseignement technique ou professionnel court	12891.6408	Exerce une profession	Ouvrier qualifié	5	Non
9	9	20	Femme	NA	7808.8721	Etudiant, eleve	NA	4	Oui
10	10	28	Homme	Enseignement technique ou professionnel long	2277.1605	Exerce une profession	Autre	2	Non
11	11	65	Femme	Enseignement supérieur y compris technique supérieur	704.3227	Retraite	Employe	3	Oui
12	12	47	Homme	2eme cycle	6697.8682	Exerce une profession	Ouvrier qualifié	4	Oui
13	13	63	Femme	Dernière année d'études primaires	7118.4659	Retraite	Employe	1	Oui
14	14	67	Femme	Enseignement technique ou professionnel court	586.7714	Exerce une profession	NA	5	Oui
15	15	76	Femme	A arrete ses etudes, avant la dernière année d'études ...	11042.0774	Retraite	NA	2	Oui
16	16	49	Femme	Enseignement technique ou professionnel court	9958.2287	Exerce une profession	Employe	3	Non
17	17	62	Homme	Enseignement supérieur y compris technique supérieur	4836.1393	Retraite	Cadre	4	Non

Figure 2. La visionneuse de données de RStudio

## summary

La fonction `summary` permet d'avoir une vue résumée d'une variable. Elle s'applique à tout type d'objets (y compris un tableau de données entier) et s'adapte à celui-ci.

```
R> summary(hdv2003$age)
```

<b>Min.</b>	<b>1st Qu.</b>	<b>Median</b>	<b>Mean</b>	<b>3rd Qu.</b>	<b>Max.</b>
18	35	48	48.157	60	97

```
R> summary(hdv2003$qualif)
```

Ouvrier specialise	Ouvrier qualifie
203	292
Technicien	Profession intermediaire
86	160
Cadre	Employe
260	594
Autre	NA's
58	347

```
R> summary(hdv2003)
```

<b>id</b>	<b>age</b>	<b>sexe</b>	<b>nivetud</b>	<b>poids</b>	<b>occup</b>	<b>qualif</b>
Min. : 1.0	Min. :18.00	Homme: 899	Enseignement technique ou professionnel court :463	Min. : 78.08	Exerce une profession:1049	Employe
1st Qu.: 500.8	1st Qu.:35.00	Femme:1101	Enseignement superieur y compris technique superieur:441	1st Qu.: 2221.82	Chomeur : 134	Ouvrier q :292
Median :1000.5	Median :48.00	NA	Derniere annee d'études primaires :341	Median : 4631.19	Etudiant, eleve : 94	Cadre :
Mean :1000.5	Mean :48.16	NA	1er cycle :204	Mean : 5535.61	Retraite : 392	Ouvrier sp :203
3rd Qu.:1500.2	3rd Qu.:60.00	NA	2eme cycle :183	3rd Qu.: 7626.53	Retire des affaires : 77	Profess intermedia
Max. :2000.0	Max. :97.00	NA	(Other) :256	Max. :31092.14	Au foyer : 171	(Other)
NA	NA	NA	NA's :112	NA	Autre inactif : 83	NA's :3

## str

La fonction `str` est plus complète que `names`. Elle liste les différentes variables, indique leur type et donne le cas échéant des informations supplémentaires ainsi qu'un échantillon des premières valeurs prises par cette variable :

```
R> str(hdv2003)
```

```
'data.frame': 2000 obs. of 20 variables:
 $ id          : int 1 2 3 4 5 6 7 8 9 10 ...
 $ age         : int 28 23 59 34 71 35 60 47 20 28 ...
 $ sexe        : Factor w/ 2 levels "Homme","Femme": 2 2 1 1 2 2 2 1 2 1 ...
 $ nivetud     : Factor w/ 8 levels "N'a jamais fait d'etudes",...: 8 NA 3 8 3
 6 3 6 NA 7 ...
 $ poids        : num 2634 9738 3994 5732 4329 ...
 $ occup        : Factor w/ 7 levels "Exerce une profession",...: 1 3 1 1 4 1 6
 1 3 1 ...
 $ qualif       : Factor w/ 7 levels "Ouvrier specialise",...: 6 NA 3 3 6 6 2 2 N
 A 7 ...
 $ freres.soeurs: int 8 2 2 1 0 5 1 5 4 2 ...
 $ cuso         : Factor w/ 3 levels "Oui","Non","Ne sait pas": 1 1 2 2 1 2 1 2
 1 2 ...
 $ relig         : Factor w/ 6 levels "Pratiquant regulier",...: 4 4 4 3 1 4 3 4
 3 2 ...
 $ trav.imp      : Factor w/ 4 levels "Le plus important",...: 4 NA 2 3 NA 1 NA 4
 NA 3 ...
 $ trav.satisf  : Factor w/ 3 levels "Satisfaction",...: 2 NA 3 1 NA 3 NA 2 NA 1
 ...
 $ hard.rock    : Factor w/ 2 levels "Non","Oui": 1 1 1 1 1 1 1 1 1 1 ...
 $ lecture.bd   : Factor w/ 2 levels "Non","Oui": 1 1 1 1 1 1 1 1 1 1 ...
 $ peche.chasse : Factor w/ 2 levels "Non","Oui": 1 1 1 1 1 1 2 2 1 1 ...
 $ cuisine       : Factor w/ 2 levels "Non","Oui": 2 1 1 2 1 1 2 2 1 1 ...
 $ bricol        : Factor w/ 2 levels "Non","Oui": 1 1 1 2 1 1 1 2 1 1 ...
 $ cinema        : Factor w/ 2 levels "Non","Oui": 1 2 1 2 1 2 1 1 2 2 ...
 $ sport         : Factor w/ 2 levels "Non","Oui": 1 2 2 2 1 2 1 1 1 2 ...
 $ heures.tv     : num 0 1 0 2 3 2 2.9 1 2 2 ...
```

## IMPORTANT

La fonction `str` est essentielle à connaître et peut s’appliquer à n’importe quel type d’objet. C’est un excellent moyen de connaître en détail la structure d’un objet. Cependant, les résultats peuvent être parfois trop détaillés et on lui priviliera dans certains cas les fonctions suivantes.

## glimpse (dplyr)

L’extension `dplyr` (voir le chapitre dédié, page 233), propose une fonction `glimpse` (ce qui signifie «aperçu» en anglais) qui permet de visualiser rapidement et de manière condensée le contenu d’un

tableau de données.

```
R> library(dplyr)
  glimpse(hdv2003)
```

```
Rows: 2,000
Columns: 20
$ id          [3m[38;5;246m<int>[39m[23m 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11...
$ age         [3m[38;5;246m<int>[39m[23m 28, 23, 59, 34, 71, 35, 60, 47, 2...
$ sexe        [3m[38;5;246m<fct>[39m[23m Femme, Femme, Homme, Homme, Femme...
$ nivetud     [3m[38;5;246m<fct>[39m[23m "Enseignement superieur y compris...
$ poids        [3m[38;5;246m<dbl>[39m[23m 2634.3982, 9738.3958, 3994.1025, ...
$ occup        [3m[38;5;246m<fct>[39m[23m "Exerce une profession", "Etudian...
$ qualif       [3m[38;5;246m<fct>[39m[23m Employe, NA, Technicien, Technici...
$ freres.soeurs [3m[38;5;246m<int>[39m[23m 8, 2, 2, 1, 0, 5, 1, 5, 4, 2, 3, ...
$ cuso         [3m[38;5;246m<fct>[39m[23m Oui, Oui, Non, Non, Oui, Non, Oui...
$ relig         [3m[38;5;246m<fct>[39m[23m Ni croyance ni appartenance, Ni c...
$ trav.imp      [3m[38;5;246m<fct>[39m[23m Peu important, NA, Aussi importan...
$ trav.satisf  [3m[38;5;246m<fct>[39m[23m Insatisfaction, NA, Equilibre, Sa...
$ hard.rock    [3m[38;5;246m<fct>[39m[23m Non, Non, Non, Non, Non, Non, Non...
$ lecture.bd   [3m[38;5;246m<fct>[39m[23m Non, Non, Non, Non, Non, Non, Non...
$ peche.chasse [3m[38;5;246m<fct>[39m[23m Non, Non, Non, Non, Non, Non, Oui...
$ cuisine       [3m[38;5;246m<fct>[39m[23m Oui, Non, Non, Oui, Non, Non, Oui...
$ bricol        [3m[38;5;246m<fct>[39m[23m Non, Non, Non, Oui, Non, Non, Non...
$ cinema         [3m[38;5;246m<fct>[39m[23m Non, Oui, Non, Oui, Non, Oui, Non...
$ sport          [3m[38;5;246m<fct>[39m[23m Non, Oui, Oui, Oui, Non, Oui, Non...
$ heures.tv     [3m[38;5;246m<dbl>[39m[23m 0.0, 1.0, 0.0, 2.0, 3.0, 2.0, 2.9...
```

## look\_for (labelled)

L'extension **labelled** propose une fonction `look_for`, inspirée de Stata, qui permet de lister les différentes variables d'un fichier de données, ainsi que leurs principales caractéristiques :

```
R> library(labelled)
  look_for(hdv2003)
```

pos	variable	label	col_type	value
s				
<chr>	<chr>	<chr>	<chr>	<ch
r>				
1	id	-	int	range: 1 - 200
0				

2	age	-	int	range: 18 - 9
7	sexe	-	fct	Homm
e				Femm
e	nivetud	-	fct	N'a jamais fait d'etude
s	d'etudes primaires			A arrete ses etudes, avant la dernière année
s				Dernière année d'études primaire
e				1er cycl
e				2eme cycl
t				Enseignement technique ou professionnel cour
g				Enseignement technique ou professionnel lon
perieur				Enseignement supérieur y compris technique su
5	poids	-	dbl	range: 78.0783403 - 31092.1413
2	occup	-	fct	Exerce une professio
n				Chomeu
r				Etudiant, elev
e				Retrait
s				Retire des affaire
r				Au foye
f				Autre inacti
7	qualif	-	fct	Ouvrier specialis
e				Ouvrier qualifi
e				Technicie
n				Profession intermediair
e				Cadr

```

e                               Employ
e                               Autr
e
8   freres.soeurs -      int    range: 0 - 2
2
9   cleso          -      fct    Ou
i
n                               No
n                               Ne sait pa
s
10  relig           -      fct    Pratiquant regulie
r
l                               Pratiquant occasionne
e
e                               Appartenance sans pratiqu
e
e                               Ni croyance ni appartenanc
e
t                               Reje
t
R                               NSP ou NVP
11  trav.imp         -      fct    Le plus importan
t
e                               Aussi important que le rest
e
e                               Moins important que le rest
e
t                               Peu importan
t
12  trav.satisf     -      fct    Satisfactio
n
n                               Insatisfactio
n
e                               Equilibr
e
13  hard.rock        -      fct    No
n
i                               Ou
i
14  lecture.bd       -      fct    No
n
i                               Ou
i

```

15	peche.chasse	-	fct	No
n				Ou
i				
16	cuisine	-	fct	No
n				Ou
i				
17	bricol	-	fct	No
n				Ou
i				
18	cinema	-	fct	No
n				Ou
i				
19	sport	-	fct	No
n				Ou
i				
20	heures.tv	-	dbl	range: 0 - 12

Lorsque l'on a un gros tableau de données avec de nombreuses variables, il peut être difficile de retrouver la ou les variables d'intérêt. Il est possible d'indiquer à `look_for` un mot-clé pour limiter la recherche. Par exemple :

```
R> look_for(hdv2003, "trav")
```

pos	variable	label	col_type	values
<chr>	<chr>	<chr>	<chr>	<chr>
11	trav.imp	-	fct	Le plus important
				Aussi important que le reste
				Moins important que le reste
				Peu important
12	trav.satisf	-	fct	Satisfaction
				Insatisfaction
				Equilibre

Il est à noter que si la recherche n'est pas sensible à la casse (i.e. aux majuscules et aux minuscules), elle est sensible aux accents. Il est aussi possible de fournir plusieurs expressions de recherche.

La fonction `look_for` est par ailleurs compatible avec les étiquettes de variable de l'extension `labelled`, les étiquettes étant prise en compte dans la recherche d'une variable.

```
R> look_for(femmes, "rés")
```

pos	variable	label	col_type	values
<chr>	<chr>	<chr>	<chr>	<chr>
7	milieu	Milieu de résidence	dbl+lbl	[1] urbain [2] rural
8	region	Région de résidence	dbl+lbl	[1] Nord [2] Est [3] Sud [4] Ouest

```
R> look_for(femmes, "rés", "nb")
```

pos	variable	label	col_type	values
<chr>	<chr>	<chr>	<chr>	<chr>
7	milieu	Milieu de résidence	dbl+lbl	[1] urbain [2] rural
8	region	Région de résidence	dbl+lbl	[1] Nord [2] Est [3] Sud [4] Ouest
16	nb_enf_ideal	Nombre idéal d'enfants	dbl+lbl	[96] Ne sait pas [99] manquant

À noter, le résultat renvoyé par `look_for` est un tableau de données qui peut ensuite être aisément manipulé (voir l'aide de la fonction).

## describe (questionr)

L'extension `questionr` fournit également une fonction bien pratique pour décrire les différentes variables d'un tableau de données. Il s'agit de `describe`. Faisons de suite un essai :

```
R> describe(hdv2003)
```

```
[2000 obs. x 20 variables] tbl_df tbl data.frame

$id:
integer: 1 2 3 4 5 6 7 8 9 10 ...
min: 1 - max: 2000 - NAs: 0 (0%) - 2000 unique values
```

```

$age:
integer: 28 23 59 34 71 35 60 47 20 28 ...
min: 18 - max: 97 - NAs: 0 (0%) - 78 unique values

$sexe:
nominal factor: "Femme" "Femme" "Homme" "Homme" "Femme" "Femme" "Femme" "Homme"
"Femme" "Homme" ...
2 levels: Homme | Femme
NAs: 0 (0%)

$nivetud:
nominal factor: "Enseignement superieur y compris technique superieur" NA "Derniere annee d'etudes primaires" "Enseignement superieur y compris technique superieur" "Derniere annee d'etudes primaires" "Enseignement technique ou professionnel court" "Derniere annee d'etudes primaires" "Enseignement technique ou professionnel court" NA "Enseignement technique ou professionnel long" ...
8 levels: N'a jamais fait d'etudes | A arrete ses etudes, avant la derniere annee d'etudes primaires | Derniere annee d'etudes primaires | 1er cycle | 2eme cycle | Enseignement technique ou professionnel court | Enseignement technique ou professionnel long | Enseignement superieur y compris technique superieur
NAs: 112 (5.6%)

$poids:
numeric: 2634.3982157 9738.3957759 3994.1024587 5731.6615081 4329.0940022 8674.6
993828 6165.8034861 12891.640759 7808.8720636 2277.160471 ...
min: 78.0783403 - max: 31092.14132 - NAs: 0 (0%) - 1877 unique values

$occup:
nominal factor: "Exerce une profession" "Etudiant, eleve" "Exerce une profession" "Exerce une profession" "Retraite" "Exerce une profession" "Au foyer" "Exerce une profession" "Etudiant, eleve" "Exerce une profession" ...
7 levels: Exerce une profession | Chomeur | Etudiant, eleve | Retraite | Retire des affaires | Au foyer | Autre inactif
NAs: 0 (0%)

$qualif:
nominal factor: "Employe" NA "Technicien" "Technicien" "Employe" "Employe" "Ouvrier qualifie" "Ouvrier qualifie" NA "Autre" ...
7 levels: Ouvrier specialise | Ouvrier qualifie | Technicien | Profession intermediaire | Cadre | Employe | Autre
NAs: 347 (17.3%)

$freres.soeurs:
integer: 8 2 2 1 0 5 1 5 4 2 ...
min: 0 - max: 22 - NAs: 0 (0%) - 19 unique values

$cloo:

```

```
nominal factor: "Oui" "Oui" "Non" "Non" "Oui" "Non" "Oui" "Non" "Oui" "Non" ...
3 levels: Oui | Non | Ne sait pas
NAs: 0 (0%)

$relig:
nominal factor: "Ni croyance ni appartenance" "Ni croyance ni appartenance" "Ni
croyance ni appartenance" "Appartenance sans pratique" "Pratiquant regulier" "N
i croyance ni appartenance" "Appartenance sans pratique" "Ni croyance ni apparte
nance" "Appartenance sans pratique" "Pratiquant occasionnel" ...
6 levels: Pratiquant regulier | Pratiquant occasionnel | Appartenance sans prati
que | Ni croyance ni appartenance | Rejet | NSP ou NVPR
NAs: 0 (0%)

$strav.imp:
nominal factor: "Peu important" NA "Aussi important que le reste" "Moins importa
nt que le reste" NA "Le plus important" NA "Peu important" NA "Moins important q
ue le reste" ...
4 levels: Le plus important | Aussi important que le reste | Moins important qu
e le reste | Peu important
NAs: 952 (47.6%)

$strav.satisf:
nominal factor: "Insatisfaction" NA "Equilibre" "Satisfaction" NA "Equilibre" N
A "Insatisfaction" NA "Satisfaction" ...
3 levels: Satisfaction | Insatisfaction | Equilibre
NAs: 952 (47.6%)

$shard.rock:
nominal factor: "Non" ...
2 levels: Non | Oui
NAs: 0 (0%)

$lecture.bd:
nominal factor: "Non" ...
2 levels: Non | Oui
NAs: 0 (0%)

$speche.chasse:
nominal factor: "Non" "Non" "Non" "Non" "Non" "Non" "Oui" "Oui" "Non" "Non" ...
2 levels: Non | Oui
NAs: 0 (0%)

$cuisine:
nominal factor: "Oui" "Non" "Non" "Oui" "Non" "Non" "Oui" "Oui" "Non" "Non" ...
2 levels: Non | Oui
NAs: 0 (0%)
```

```
$bricol:
nominal factor: "Non" "Non" "Non" "Oui" "Non" "Non" "Non" "Oui" "Non" "Non" ...
2 levels: Non | Oui
NAs: 0 (0%)

$cinema:
nominal factor: "Non" "Oui" "Non" "Oui" "Non" "Oui" "Non" "Non" "Oui" "Oui" ...
2 levels: Non | Oui
NAs: 0 (0%)

$sport:
nominal factor: "Non" "Oui" "Oui" "Oui" "Non" "Oui" "Non" "Non" "Non" "Oui" ...
2 levels: Non | Oui
NAs: 0 (0%)

$heures.tv:
numeric: 0 1 0 2 3 2 2.9 1 2 2 ...
min: 0 - max: 12 - NAs: 5 (0.2%) - 30 unique values
```

Comme on le voit sur cet exemple, `describe` nous affiche le type des variables, les premières valeurs de chacune, le nombre de valeurs manquantes, le nombre de valeurs différentes (uniques) ainsi que quelques autres informations suivant le type de variables.

Il est possible de restreindre l'affichage à seulement quelques variables en indiquant le nom de ces dernières ou une expression de recherche (comme avec `lookfor`).

```
R> describe(hdv2003, "age", "trav")
```

```
[2000 obs. x 20 variables] tbl_df tbl data.frame

$age:
integer: 28 23 59 34 71 35 60 47 20 28 ...
min: 18 - max: 97 - NAs: 0 (0%) - 78 unique values

$strav.imp:
nominal factor: "Peu important" NA "Aussi important que le reste" "Moins important que le reste" NA "Le plus important" NA "Peu important" NA "Moins important que le reste" ...
4 levels: Le plus important | Aussi important que le reste | Moins important que le reste | Peu important
NAs: 952 (47.6%)

$strav.satisf:
nominal factor: "Insatisfaction" NA "Equilibre" "Satisfaction" NA "Equilibre" NA "Insatisfaction" NA "Satisfaction" ...
```

```
3 levels: Satisfaction | Insatisfaction | Equilibre  
NAs: 952 (47.6%)
```

On peut également transmettre juste une variable :

```
R> describe(hdv2003$sexe)
```

```
[2000 obs.]  
nominal factor: "Femme" "Femme" "Homme" "Homme" "Femme" "Femme" "Femme" "Homme"  
"Femme" "Homme" ...  
2 levels: Homme | Femme  
NAs: 0 (0%)  
  
n %  
Homme 899 45  
Femme 1101 55  
Total 2000 100
```

Enfin, `describe` est également compatible avec les vecteurs labellisés, page 117.

```
R> describe(femmes, "milieu")
```

```
[2000 obs. x 17 variables] tbl_df tbl data.frame  
  
$milieu: Milieu de résidence  
labelled double: 2 2 2 2 2 2 2 2 2 2 ...  
min: 1 - max: 2 - NAs: 0 (0%) - 2 unique values  
2 value labels: [1] urbain [2] rural
```

À noter, l'argument `freq.n.max` permet d'indiquer le nombre de modalités en-dessous duquel `describe` renverra également un tri à plat de la variable.

```
R> describe(menages, freq.n.max = 6)
```

```
[1814 obs. x 5 variables] tbl_df tbl data.frame  
  
$id_menage: Identifiant du ménage  
numeric: 1 2 3 4 5 6 7 8 9 10 ...  
min: 1 - max: 1814 - NAs: 0 (0%) - 1814 unique values  
  
$taille: Taille du ménage (nombre de membres)  
numeric: 7 3 6 5 7 6 15 6 5 19 ...
```

```

min: 1 - max: 31 - NAs: 0 (0%) - 30 unique values

$sexe_chef: Sexe du chef de ménage
labelled double: 2 1 1 1 1 2 2 2 1 1 ...
min: 1 - max: 2 - NAs: 0 (0%) - 2 unique values
2 value labels: [1] homme [2] femme

      n      %
[1] homme 1420 78.3
[2] femme  394 21.7
Total     1814 100.0

$structure: Structure démographique du ménage
labelled double: 4 2 5 4 4 4 5 2 5 5 ...
min: 1 - max: 5 - NAs: 0 (0%) - 5 unique values
6 value labels: [0] pas d'adulte [1] un adulte [2] deux adultes de sexe opposé
[3] deux adultes de même sexe [4] trois adultes ou plus avec lien de parenté
[5] adultes sans lien de parenté

      n      %
[0] pas d'adulte          0 0.0
[1] un adulte              78 4.3
[2] deux adultes de sexe opposé 439 24.2
[3] deux adultes de même sexe   75 4.1
[4] trois adultes ou plus avec lien de parenté 920 50.7
[5] adultes sans lien de parenté    302 16.6
Total                     1814 100.0

$richesse: Niveau de vie (quintiles)
labelled double: 1 2 2 1 1 3 2 5 4 3 ...
min: 1 - max: 5 - NAs: 0 (0%) - 5 unique values
5 value labels: [1] très pauvre [2] pauvre [3] moyen [4] riche [5] très riche

      n      %
[1] très pauvre 335 18.5
[2] pauvre      357 19.7
[3] moyen       402 22.2
[4] riche        350 19.3
[5] très riche  370 20.4
Total         1814 100.0

```

## Addins de codebook

L'extension `codebook` fournit plusieurs addins permettant d'explorer les étiquettes des différentes variables d'un tableau de données. Ils sont accessibles par le menu **Addins de RStudio**.

On peut aussi lancer les addins manuellement, par exemple avec la commande suivante :

```
R> codebook::label_browser_static(femmes)
```

## skim (skimr)

L’extension **skimr** a pour objectif de fournir une fonction **skim** comme alternative à **summary** {base} pour les vecteurs et les tableaux de données afin de fournir plus de statistiques dans un format plus compact. Elle peut être appliquée à un vecteur donné ou directement à un tableau de données.

```
R> library(skimr)
skim(hdv2003$cuisine)
```

Data summary

Name	hdv2003\$cuisine
Number of rows	2000
Number of columns	1

Column type frequency:

factor	1
--------	---

Group variables	None
-----------------	------

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
data	0	1	FALSE	2	Non: 1119, Oui: 881

```
R> skim(hdv2003)
```

## Data summary

Name	hdv2003
Number of rows	2000
Number of columns	20
<hr/>	
Column type frequency:	
factor	15
numeric	5
<hr/>	
Group variables	None

**Variable type: factor**

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
sexe	0	1.00	FALSE	2	Fem: 1101, Hom: 899
nivetud	112	0.94	FALSE	8	Ens: 463, Ens: 441, Der: 341, 1er: 204
occup	0	1.00	FALSE	7	Exe: 1049, Ret: 392, Au : 171, Cho: 134
qualif	347	0.83	FALSE	7	Emp: 594, Ouv: 292, Cad: 260, Ouv: 203
clso	0	1.00	FALSE	3	Non: 1037, Oui: 936, Ne : 27
relig	0	1.00	FALSE	6	App: 760, Pra: 442, Ni : 399, Pra: 266
trav.imp	952	0.52	FALSE	4	Moi: 708, Aus: 259, Peu: 52, Le : 29
trav.satisf	952	0.52	FALSE	3	Sat: 480, Equ: 451, Ins: 117
hard.rock	0	1.00	FALSE	2	Non: 1986, Oui: 14
lecture.bd	0	1.00	FALSE	2	Non: 1953, Oui: 47
peche.chasse	0	1.00	FALSE	2	Non: 1776, Oui: 224
cuisine	0	1.00	FALSE	2	Non: 1119, Oui: 881
bricol	0	1.00	FALSE	2	Non: 1147, Oui: 853
cinema	0	1.00	FALSE	2	Non: 1174, Oui: 826
sport	0	1.00	FALSE	2	Non: 1277, Oui: 723

**Variable type: numeric**

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75
id	0	1	1000.50	577.49	1.00	500.75	1000.50	1500.00
age	0	1	48.16	16.94	18.00	35.00	48.00	65.00
poids	0	1	5535.61	4375.03	78.08	2221.82	4631.19	7622.00
freres.soeurs	0	1	3.28	2.77	0.00	1.00	2.00	6.00
heures.tv	5	1	2.25	1.78	0.00	1.00	2.00	6.00

On peut noter que les variables sont regroupées par type.

Il est possible de sélectionner des variables à la manière de `dplyr`. Voir l'aide de `contains`.

```
R> skim(hdv2003, contains("re"))
```

#### Data summary

Name	hdv2003
Number of rows	2000
Number of columns	20
<hr/>	
Column type frequency:	
factor	2
numeric	2
<hr/>	
Group variables	None

#### Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
relig	0	1	FALSE	6	App: 760, Pra: 442, Ni : 399, Pra: 266
lecture.bd	0	1	FALSE	2	Non: 1953, Oui: 47

**Variable type: numeric**

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
freres.soeurs	0	1	3.28	2.77	0	1	2	5	22	
heures.tv	5	1	2.25	1.78	0	1	2	3	12	

Le support des vecteurs labellisés est encore en cours d'intégration.

```
R> skim(menages)
```

```
Warning: Couldn't find skimmers for class: haven_labelled,
vctrs_vctr, double, numeric; No user-defined `sfl` provided.
Falling back to `character`.
```

```
Warning: Couldn't find skimmers for class: haven_labelled,
vctrs_vctr, double, numeric; No user-defined `sfl` provided.
Falling back to `character`.
```

```
Warning: Couldn't find skimmers for class: haven_labelled,
vctrs_vctr, double, numeric; No user-defined `sfl` provided.
Falling back to `character`.
```

**Data summary**

Name	menages
Number of rows	1814
Number of columns	5
<hr/>	

**Column type frequency:**

character	3
numeric	2
<hr/>	

Group variables      None

**Variable type: character**

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
sexe_chef	0	1	1	1	0	2	0
structure	0	1	1	1	0	5	0
richesse	0	1	1	1	0	5	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
id_menage	0	1	907.5	523.80	1	454.25	907.5	1360.75	1880.00
taille	0	1	7.5	4.42	1	4.00	6.0	9.00	12.00

## create\_report (DataExplorer)

L'extension **DataExplorer** fournit des outils d'exploration graphique d'un fichier de données. En premier lieu, sa fonction `create_report` génère un rapport automatique à partir d'un tableau de données.

```
R> library(DataExplorer)
    create_report(hdv2003)
```

Le résultat de ce rapport est visible sur [http://lamarange.github.io/analyse-R/data/hdv2003\\_DataExplorer\\_report.html](http://lamarange.github.io/analyse-R/data/hdv2003_DataExplorer_report.html).

L'extension fournit également différentes fonctions graphiques, présentées en détail dans la vignette inclue dans l'extension et visible sur <https://cran.r-project.org/web/packages/DataExplorer/vignettes/dataexplorer-intro.html>.

## makeCodebook (dataMaid)

L'extension **dataMaid** propose une fonction `makeCodebook` permettant de générer une présentation de l'ensemble des variables d'un tableau de données, au format PDF, Word ou HTML.

```
R> library(dataMaid)
    makeCodebook(hdv2003)
```

Vous pouvez cliquez sur ce lien pour voir le PDF produit par **dataMaid**.



# Recodage de variables

Renommer des variables . . . . .	198
Convertir une variable . . . . .	199
Variable numérique ou textuelle en facteur . . . . .	200
Conversion d'un facteur . . . . .	200
Conversion d'un vecteur labellisé . . . . .	202
Quand convertir les vecteurs labellisés ? . . . . .	202
Convertir un vecteur labellisé en facteur . . . . .	203
Convertir un vecteur labellisé en numérique ou en texte . . . . .	205
Conversion conditionnelle en facteurs . . . . .	207
Découper une variable numérique en classes . . . . .	209
Regrouper les modalités d'une variable . . . . .	213
Variables calculées . . . . .	219
Combiner plusieurs variables . . . . .	220
Variables scores . . . . .	222
Vérification des recodages . . . . .	223
Facteurs et forcats . . . . .	224
Modifier les modalités d'une variable qualitative . . . . .	224
Ordonner les modalités d'une variable qualitative . . . . .	226
Combiner plusieurs variables . . . . .	228
if_else . . . . .	228
case_when . . . . .	229
Recodage et data.table . . . . .	230

## NOTE

La version originale de ce chapitre a été écrite par Julien Barnier dans le cadre du support de cours [Introduction à R](#).

Le recodage de variables est une opération extrêmement fréquente lors du traitement d'enquête. Celui-ci utilise soit l'une des formes d'indexation décrites précédemment, soit des fonctions *ad hoc* de R.

On passe ici en revue différents types de recodage parmi les plus courants. Les exemples s'appuient,

comme précédemment, sur l’extract de l’enquête *Histoire de vie* :

```
R> library(questionr)
  data(hdv2003)
  d <- hdv2003
```

## Renommer des variables

Une opération courante lorsqu’on a importé des variables depuis une source de données externe consiste à renommer les variables importées. Sous R les noms de variables doivent être à la fois courts et explicites.

### IMPORTANT

Les noms de variables peuvent contenir des lettres, des chiffres (mais ils ne peuvent pas commencer par un chiffre), les symboles . et \_ et doivent commencer par une lettre. R fait la différence entre les majuscules et les minuscules, ce qui signifie que x et X sont deux noms de variable différents. On évitera également d’utiliser des caractères accentués dans les noms de variable. Comme les espaces ne sont pas autorisés, on pourra les remplacer par un point ou un tiret bas.

On peut lister les noms des variables d’un tableau de données (*data.frame*) à l’aide de la fonction `names` :

```
R> names(d)
```

```
[1] "id"          "age"         "sexe"
[4] "nivetud"     "poids"       "occup"
[7] "qualif"      "freres.soeurs" "clso"
[10] "relig"       "trav.imp"    "trav.satisf"
[13] "hard.rock"   "lecture.bd"  "peche.chasse"
[16] "cuisine"     "bricol"      "cinema"
[19] "sport"       "heures.tv"
```

Cette fonction peut également être utilisée pour renommer l’ensemble des variables. Si par exemple on souhaitait passer les noms de toutes les variables en majuscules, on pourrait faire :

```
R> d.maj <- d
  names(d.maj) <- c(
    "ID", "AGE", "SEXE", "NIVETUD", "POIDS", "OCCUP", "QUALIF",
    "FRERES.SOEURS", "CLSO", "RELIG", "TRAV.IMP", "TRAV.SATISF", "HARD.ROCK",
    "LECTURE.BD", "PECHE.CHASSE", "CUISINE", "BRICOL", "CINEMA", "SPORT", "HEURE
    S.TV"
  )
  summary(d.maj$SEXE)
```

Homme	Femme
899	1101

Ce type de renommage peut être utile lorsqu'on souhaite passer en revue tous les noms de variables d'un fichier importé pour les corriger le cas échéant. Pour faciliter un peu ce travail pas forcément passionnant, on peut utiliser la fonction `dput` :

```
R> dput(names(d))
```

```
c("id", "age", "sexe", "nivetud", "poids", "occup", "qualif",
  "freres.soeurs", "clso", "relig", "trav.imp", "trav.satisf",
  "hard.rock", "lecture.bd", "peche.chasse", "cuisine", "bricol",
  "cinema", "sport", "heures.tv")
```

On obtient en résultat la liste des variables sous forme de vecteur déclaré. On n'a plus alors qu'à copier/coller cette chaîne, rajouter `names(d) <-` devant et modifier un à un les noms des variables.

Si on souhaite seulement modifier le nom d'une variable, on peut utiliser la fonction `rename.variable` de l'extension `questionr`. Celle-ci prend en argument le tableau de données, le nom actuel de la variable et le nouveau nom. Par exemple, si on veut renommer la variable `bricol` du tableau de données `d` en `bricolage`:

```
R> d <- rename.variable(d, "bricol", "bricolage")
  table(d$bricolage)
```

Non	Oui
1147	853

## Convertir une variable

Il peut arriver qu'on veuille transformer une variable d'un type dans un autre.

## Variable numérique ou textuelle en facteur

Par exemple, on peut considérer que la variable numérique `freres.soeurs` est une « fausse » variable numérique et qu’une représentation sous forme de facteur serait plus adéquate. Dans ce cas il suffit de faire appel à la fonction `factor` :

```
R> d$fs.fac <- factor(d$freres.soeurs)
  levels(d$fs.fac)
```

```
[1] "0"   "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"
[12] "11"  "12"  "13"  "14"  "15"  "16"  "17"  "18"  "22"
```

La conversion d’une variable caractères en facteur se fait de la même manière.

La conversion d’un facteur ou d’une variable numérique en variable caractères peut se faire à l’aide de la fonction `as.character` :

```
R> d$fs.char <- as.character(d$freres.soeurs)
  d$qualif.char <- as.character(d$qualif)
```

## Conversion d’un facteur

La conversion d’un facteur en caractères est fréquemment utilisé lors des recodages du fait qu’il est impossible d’ajouter de nouvelles modalités à un facteur de cette manière. Par exemple, la première des commandes suivantes génère un message d’avertissement, tandis que les deux autres fonctionnent :

```
R> d.temp <- d
  d.temp$qualif[d.temp$qualif == "Ouvrier specialise"] <- "Ouvrier"
```

```
Warning in `<-factor`(`*tmp*`, d.temp$qualif == "Ouvrier
specialise", : invalid factor level, NA generated
```

```
R> d$qualif.char <- as.character(d$qualif)
  d$qualif.char[d$qualif.char == "Ouvrier specialise"] <- "Ouvrier"
```

Dans le premier cas, le message d’avertissement indique que toutes les modalités « Ouvrier specialise » de notre variable `qualif` ont été remplacées par des valeurs manquantes `NA`.

Enfin, une variable de type caractères dont les valeurs seraient des nombres peut être convertie en

variable numérique avec la fonction `as.numeric`.

```
R> v <- c("1", "3.1415", "4", "5.6", "1", "4")
v
```

```
[1] "1"      "3.1415" "4"      "5.6"      "1"      "4"
```

```
R> as.numeric(v)
```

```
[1] 1.0000 3.1415 4.0000 5.6000 1.0000 4.0000
```

### IMPORTANT

Lorsque l'on convertit un facteur avec `as.numeric`, on obtient le numéro de chaque facteur (première modalité, seconde modalité, etc.). Si la valeur numérique qui nous intéresse est en fait contenu dans le nom des modalités, il faut convertir au préalable notre facteur en variable textuelle.

```
R> vf <- factor(v)
vf
```

```
[1] 1      3.1415 4      5.6     1      4
Levels: 1 3.1415 4 5.6
```

```
R> as.numeric(vf)
```

```
[1] 1 2 3 4 1 3
```

```
R> as.numeric(as.character(vf))
```

```
[1] 1.0000 3.1415 4.0000 5.6000 1.0000 4.0000
```

**ATTENTION :** la valeur numérique associée à chaque étiquette d'un facteur change lorsque l'on modifie l'ordre des étiquettes d'un facteur. Dès lors, il est **fortement déconseillé** de convertir un facteur en variable numérique.

# Conversion d'un vecteur labellisé

Nous avons abordé dans un chapitre précédent, page 123 la gestion de données labellisées à l'aide de l'extension `labelled`. Les vecteurs labellisés sont beaucoup plus souples que les facteurs lors de la préparation des données, puisque la liste des modalités autorisées n'est pas fixée à l'avance. De plus, cela permet également de documenter au-fur-et-à-mesure les nouvelles variables que l'on créé.

**Nous verrons dans les chapitres d'analyse, notamment quand il s'agit de calculer des modèles, qu'il est nécessaire de coder les variables catégorielles sous forme de facteurs. Et il est préférable pour les variables continues de leur retirer la classe `haven_labelled`.**

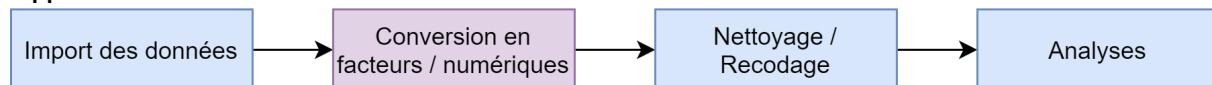
## Quand convertir les vecteurs labellisés ?

La classe `haven_labelled` permet d'ajouter des métadonnées aux variables sous la forme d'étiquettes de valeurs. Lorsque les données sont importées depuis **SAS**, **SPSS** ou **Stata**, cela permet notamment de conserver le codage original du fichier importé.

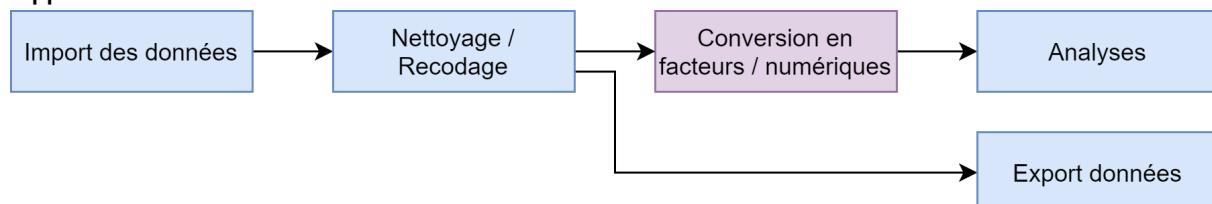
Mais il faut noter que ces étiquettes de valeur n'indique pas pour autant de manière systématique le type de variable (catégorielle ou continue). Les vecteurs labellisés n'ont donc pas vocation à être utilisés pour l'analyse, notamment le calcul de modèles statistiques. Ils doivent être convertis en facteurs (pour les variables catégorielles) ou en vecteurs numériques (pour les variables continues).

La question qui peut se poser est donc de choisir à quel moment cette conversion doit avoir lieu dans un processus d'analyse. On peut considérer deux approches principales.

### Approche A



### Approche B



Deux approches possibles

Dans l'**approche A**, les vecteurs labellisés sont convertis juste après l'import des données, en utilisant les fonctions `unlabelled`, `to_factor` ou `unclass` qui sont présentées ci-après. Dès lors, toute la partie de nettoyage et de recodage des données se fera en utilisant les fonctions classiques de R.

Dans l'**approche B**, les vecteurs labellisés sont conservés pour l'étape de nettoyage et de recodage des

données. Dans ce cas là, on pourra avoir recours aux fonctions de l'extension **labelled** qui facilitent la gestion des données labellisées. Cette approche est particulièrement intéressante quand (i) on veut pouvoir se référer au dictionnaire de codification fourni avec les données sources et donc on veut conserver le codage original et/ou (ii) quand les données devront faire l'objet d'un réexport après transformation. Par contre, comme dans l'approche A, il faudra prévoir une conversion des variables labellisées au moment de l'analyse.

## Convertir un vecteur labellisé en facteur

Il est très facile de convertir un vecteur labellisé en facteur à l'aide la fonction `to_factor` de l'extension **labelled**<sup>1</sup>.

```
R> library(labelled)
v <- labelled(c(1, 2, 9, 3, 3, 2, NA), c(oui = 1, "peut-être" = 2, non = 3, "ne sait pas" = 9))
v
```

```
<labelled<double>[7]>
[1] 1 2 9 3 3 2 NA

Labels:
  value      label
    1        oui
    2   peut-être
    3       non
    9 ne sait pas
```

```
R> to_factor(v)
```

```
[1] oui      peut-être  ne sait pas non
[5] non      peut-être  <NA>
Levels: oui peut-être non ne sait pas
```

Il possible d'indiquer si l'on souhaite, comme étiquettes du facteur, utiliser les étiquettes de valeur (par défaut), les valeurs elles-mêmes, ou bien les étiquettes de valeurs préfixées par la valeur d'origine indiquée entre crochets.

---

1. On priviliera la fonction `to_factor` à la fonction `as_factor` de l'extension **haven**, la première ayant plus de possibilités et un comportement plus consistant.

```
R> to_factor(v, "l")
```

```
[1] oui           peut-être   ne sait pas non  
[5] non           peut-être   <NA>  
Levels: oui peut-être non ne sait pas
```

```
R> to_factor(v, "v")
```

```
[1] 1    2    9    3    3    2    <NA>  
Levels: 1 2 3 9
```

```
R> to_factor(v, "p")
```

```
[1] [1] oui           [2] peut-être   [9] ne sait pas  
[4] [3] non           [3] non           [2] peut-être  
[7] <NA>  
4 Levels: [1] oui [2] peut-être ... [9] ne sait pas
```

Par défaut, les étiquettes du facteur seront triés selon l’ordre des étiquettes de valeur. Mais cela peut être modifié avec l’argument `sort_levels` si l’on préfère trier selon les valeurs ou selon l’ordre alphabétique des étiquettes.

```
R> to_factor(v, sort_levels = "v")
```

```
[1] oui           peut-être   ne sait pas non  
[5] non           peut-être   <NA>  
Levels: oui peut-être non ne sait pas
```

```
R> to_factor(v, sort_levels = "l")
```

```
[1] oui           peut-être   ne sait pas non  
[5] non           peut-être   <NA>  
Levels: ne sait pas non oui peut-être
```

D’autres options sont disponibles. On se réferra à la documentation complète de la fonction.

## Convertir un vecteur labellisé en numérique ou en texte

Pour rappel, il existe deux types de vecteurs labellisés : des vecteurs numériques labellisés ( `x` dans l'exemple ci-dessous) et des vecteurs textuels labellisés ( `y` dans l'exemple ci-dessous).

```
R> x <- labelled(c(1, 2, 9, 3, 3, 2, NA), c(oui = 1, "peut-être" = 2, non = 3, "ne sait pas" = 9))
y <- labelled(c("f", "f", "h", "f"), c(femme = "f", homme = "h"))
x
```

```
<labelled<double>[7]>
[1] 1 2 9 3 3 2 NA

Labels:
  value      label
    1        oui
    2   peut-être
    3       non
    9 ne sait pas
```

```
R> y
```

```
<labelled<character>[4]>
[1] f f h f

Labels:
  value label
    f femme
    h homme
```

Pour leur retirer leur caractère labellisé et revenir à leur classe d'origine, on peut utiliser la fonction `unclass`.

```
R> unclass(x)
```

```
[1] 1 2 9 3 3 2 NA
attr(,"labels")
  oui   peut-être       non ne sait pas
    1           2           3           9
```

```
R> unclass(y)
```

```
[1] "f" "f" "h" "f"  
attr(,"labels")  
femme homme  
"f" "h"
```

À noter que dans ce cas-là, les étiquettes sont conservées comme attributs du vecteur.

Une alternative est d'utiliser `remove_labels` qui supprimera toutes les étiquettes, y compris les étiquettes de variable. Pour conserver les étiquettes de variables et ne supprimer que les étiquettes de valeurs, on indiquera `keep_var_label = TRUE`.

```
R> var_label(x) <- "Etiquette de variable"  
remove_labels(x)
```

```
[1] 1 2 9 3 3 2 NA
```

```
R> remove_labels(x, keep_var_label = TRUE)
```

```
[1] 1 2 9 3 3 2 NA  
attr(,"label")  
[1] "Etiquette de variable"
```

```
R> remove_labels(y)
```

```
[1] "f" "f" "h" "f"
```

Dans le cas d'un vecteur numérique labellisé que l'on souhaiterait convertir en variable textuelle, on pourra utiliser `to_character` à la place de `to_factor` qui comme sa soeur utilisera les étiquettes de valeurs.

```
R> to_character(x)
```

```
[1] "oui"           "peut-être"    "ne sait pas" "non"  
[5] "non"          "peut-être"    NA  
attr(,"label")  
[1] "Etiquette de variable"
```

## Conversion conditionnelle en facteurs

Il n'est pas toujours possible de déterminer la nature d'une variable (continue ou catégorielle) juste à partir de la présence ou l'absence d'étiquettes de valeur. En effet, on peut utiliser des étiquettes de valeur dans le cadre d'une variable continue pour indiquer certaines valeurs spécifiques.

Une bonne pratique est de vérifier chaque variable inclue dans une analyse, une à une.

Cependant, une règle qui fonctionne dans 90% des cas est de convertir un vecteur labellisé en facteur seulement si toutes les valeurs observées dans le vecteur disposent d'une étiquette de valeur correspondante. C'est ce que propose la fonction `unlabelled` qui peut même être appliqué à tout un tableau de données. Par défaut, elle fonctionne ainsi :

1. les variables non labellisées restent inchangées (variables `f` et `g` dans l'exemple ci-dessous);
2. si toutes les valeurs observées d'une variable labellisées, elles sont converties en facteurs (variables `b` et `c`);
3. sinon, on leur applique `unclass` (variables `a`, `d` et `e`).

```
R> df <- data.frame(
  a = labelled(c(1, 1, 2, 3), labels = c(No = 1, Yes = 2)),
  b = labelled(c(1, 1, 2, 3), labels = c(No = 1, Yes = 2, DK = 3)),
  c = labelled(c(1, 1, 2, 2), labels = c(No = 1, Yes = 2, DK = 3)),
  d = labelled(c("a", "a", "b", "c"), labels = c(No = "a", Yes = "b")),
  e = labelled_spss(
    c(1, 9, 1, 2),
    labels = c(No = 1, Yes = 2),
    na_values = 9
  ),
  f = 1:4,
  g = as.Date(c("2020-01-01", "2020-02-01", "2020-03-01", "2020-04-01"))
)
dplyr::glimpse(df)
```

```
Rows: 4
Columns: 7
$ a <dbl+lbl> 1, 1, 2, 3
$ b <dbl+lbl> 1, 1, 2, 3
$ c <dbl+lbl> 1, 1, 2, 2
$ d <chr+lbl> "a", "a", "b", "c"
$ e <dbl+lbl> 1, 9, 1, 2
$ f <int> 1, 2, 3, 4
$ g <date> 2020-01-01, 2020-02-01, 2020-03-01, 2020-04-01
```

```
R> dplyr::glimpse(unlabelled(df))
```

```
Rows: 4
Columns: 7
$ a <dbl> 1, 1, 2, 3
$ b <fct> No, No, Yes, DK
$ c <fct> No, No, Yes, Yes
$ d <chr> "a", "a", "b", "c"
$ e <fct> No, NA, No, Yes
$ f <int> 1, 2, 3, 4
$ g <date> 2020-01-01, 2020-02-01, 2020-03-01, 2020-04-01
```

On peut indiquer certaines options, par exemple `drop_unused_labels = TRUE` pour supprimer des facteurs créés les niveaux non observées dans les données (voir la variable `c`) ou `user_na_to_na = TRUE` pour convertir les valeurs manquantes de type SPSS en `NA` (voir la variable `e`, notez qu’ici, du fait de cette conversion, toutes les valeurs ont une étiquette et la variable est alors convertie en facteur.)

```
R> dplyr::glimpse(unlabelled(df, levels = "prefixed"))
```

```
Rows: 4
Columns: 7
$ a <dbl> 1, 1, 2, 3
$ b <fct> [1] No, [1] No, [2] Yes, [3] DK
$ c <fct> [1] No, [1] No, [2] Yes, [2] Yes
$ d <chr> "a", "a", "b", "c"
$ e <fct> [1] No, NA, [1] No, [2] Yes
$ f <int> 1, 2, 3, 4
$ g <date> 2020-01-01, 2020-02-01, 2020-03-01, 2020-04-01
```

```
R> dplyr::glimpse(unlabelled(df, user_na_to_na = TRUE))
```

```
Rows: 4
Columns: 7
$ a <dbl> 1, 1, 2, 3
$ b <fct> No, No, Yes, DK
$ c <fct> No, No, Yes, Yes
$ d <chr> "a", "a", "b", "c"
$ e <fct> No, NA, No, Yes
$ f <int> 1, 2, 3, 4
$ g <date> 2020-01-01, 2020-02-01, 2020-03-01, 2020-04-01
```

```
R> dplyr::glimpse(unlabelled(df, drop_unused_labels = TRUE))
```

```
Rows: 4
Columns: 7
$ a <dbl> 1, 1, 2, 3
$ b <fct> No, No, Yes, DK
$ c <fct> No, No, Yes, Yes
$ d <chr> "a", "a", "b", "c"
$ e <fct> No, NA, No, Yes
$ f <int> 1, 2, 3, 4
$ g <date> 2020-01-01, 2020-02-01, 2020-03-01, 2020-04-01
```

## Découper une variable numérique en classes

Le premier type de recodage consiste à découper une variable de type numérique en un certain nombre de classes. On utilise pour cela la fonction `cut`.

Celle-ci prend, outre la variable à découper, un certain nombre d'arguments :

- `breaks` indique soit le nombre de classes souhaité, soit, si on lui fournit un vecteur, les limites des classes ;
- `labels` permet de modifier les noms de modalités attribués aux classes ;
- `include.lowest` et `right` influent sur la manière dont les valeurs situées à la frontière des classes seront incluses ou exclues ;
- `dig.lab` indique le nombre de chiffres après la virgule à conserver dans les noms de modalités.

Prenons tout de suite un exemple et tentons de découper notre variable `age` en cinq classes et de placer le résultat dans une nouvelle variable nommée `age5cl` :

```
R> d$age5cl <- cut(d$age, 5)
table(d$age5cl)
```

Intervalle d'âge	Nombre d'observations
(17.9,33.8]	454
(33.8,49.6]	628
(49.6,65.4]	556
(65.4,81.2]	319
(81.2,97.1]	43

Par défaut R nous a bien créé cinq classes d'amplitudes égales. La première classe va de 16,9 à 32,2 ans (en fait de 17 à 32), etc.

Les frontières de classe seraient plus présentables si elles utilisaient des nombres ronds. On va donc spécifier manuellement le découpage souhaité, par tranches de 20 ans :

```
R> d$age20 <- cut(d$age, c(0, 20, 40, 60, 80, 100))
table(d$age20)
```

(0,20]	(20,40]	(40,60]	(60,80]	(80,100]
72	660	780	436	52

On aurait pu tenir compte des âges extrêmes pour la première et la dernière valeur :

```
R> range(d$age)
```

[1]	18	97
-----	----	----

```
R> d$age20 <- cut(d$age, c(18, 20, 40, 60, 80, 97))
table(d$age20)
```

(18,20]	(20,40]	(40,60]	(60,80]	(80,97]
55	660	780	436	52

Les symboles dans les noms attribués aux classes ont leur importance : ( signifie que la frontière de la classe est exclue, tandis que [ signifie qu'elle est incluse. Ainsi, (20,40] signifie « strictement supérieur à 20 et inférieur ou égal à 40 ».

On remarque que du coup, dans notre exemple précédent, la valeur minimale, 18, est exclue de notre première classe, et qu'une observation est donc absente de ce découpage. Pour résoudre ce problème on peut soit faire commencer la première classe à 17, soit utiliser l'option `include.lowest=TRUE` :

```
R> d$age20 <- cut(d$age, c(17, 20, 40, 60, 80, 97))
table(d$age20)
```

(17,20]	(20,40]	(40,60]	(60,80]	(80,97]
72	660	780	436	52

```
R> d$age20 <- cut(d$age, c(18, 20, 40, 60, 80, 97), include.lowest = TRUE)
table(d$age20)
```

[18,20]	(20,40]	(40,60]	(60,80]	(80,97]
72	660	780	436	52

On peut également modifier le sens des intervalles avec l'option `right=FALSE`, et indiquer

manuellement les noms des modalités avec `labels` :

```
R> d$age20 <- cut(d$age, c(18, 20, 40, 60, 80, 97), right = FALSE, include.lowest = TRUE)
  table(d$age20)
```

```
[18,20) [20,40) [40,60) [60,80) [80,97]
    48      643     793     454      62
```

```
R> d$age20 <- cut(d$age, c(18, 20, 40, 60, 80, 97), include.lowest = TRUE, labels = c("<20ans", "21-40 ans", "41-60ans", "61-80ans", ">80ans"))
  table(d$age20)
```

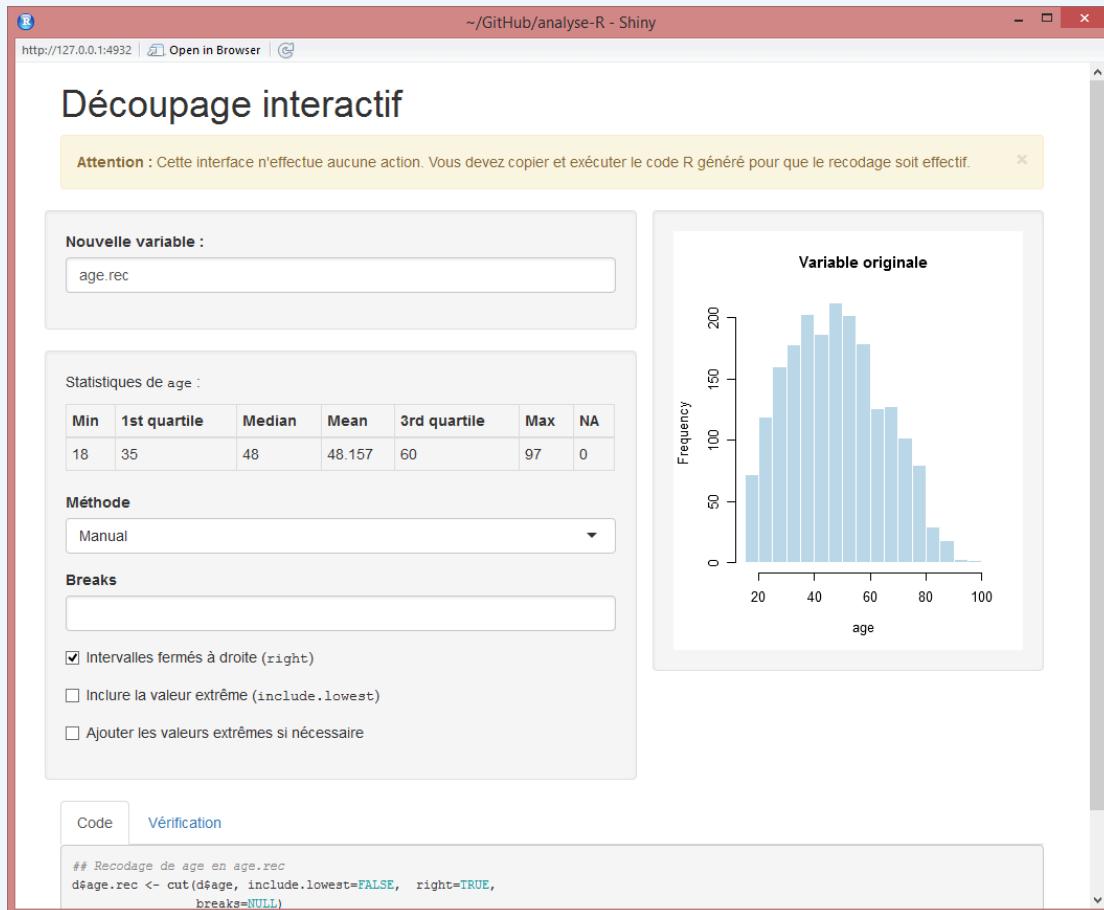
```
<20ans 21-40 ans 41-60ans 61-80ans >80ans
    72       660      780      436      52
```

## NOTE

L’extension `questionr` propose une interface interactive à la fonction `cut`, nommée `icut`. Elle s’utilise de la manière suivante :

```
R> icut(d, age)
```

RStudio devrait ouvrir une fenêtre semblable à l’image ci-dessous.



Capture d’écran d’icut

Vous pouvez alors indiquer les limites de vos classes ainsi que quelques options complémentaires. Ces limites sont représentées graphiquement sur l’histogramme de la variable d’origine.

L’onglet Vérification affiche un tri à plat et un graphique en barres de la nouvelle variable. Une fois le résultat satisfaisant, vous pouvez récupérer le code généré pour l’inclure dans votre script.

L’extension `questionr` propose aussi une fonction `quant.cut` permettant de découper une variable

numérique en un nombre de classes donné ayant des effectifs semblables. Il suffit de lui passer le nombre de classes en argument :

```
R> d$age6cl <- quant.cut(d$age, 6)
table(d$age6cl)
```

[18,30)	[30,39)	[39,48)	[48,55.667)	[55.667,66)
302	337	350	344	305
[66,97]				
362				

`quant.cut` admet les mêmes autres options que `cut` (`include.lowest`, `right`, `labels` ...).

## Regrouper les modalités d'une variable

Pour regrouper les modalités d'une variable qualitative (d'un facteur le plus souvent), on peut utiliser directement l'indexation.

Ainsi, si on veut recoder la variable `qualif` dans une variable `qualif.reg` plus « compacte », on peut utiliser :

```
R> table(d$qualif)
```

Ouvrier specialise	Ouvrier qualifie
203	292
Technicien	Profession intermediaire
86	160
Cadre	Employe
260	594
Autre	
58	

```
R> d$qualif.reg[d$qualif == "Ouvrier specialise"] <- "Ouvrier"
d$qualif.reg[d$qualif == "Ouvrier qualifie"] <- "Ouvrier"
d$qualif.reg[d$qualif == "Employe"] <- "Employe"
d$qualif.reg[d$qualif == "Profession intermediaire"] <- "Intermediaire"
d$qualif.reg[d$qualif == "Technicien"] <- "Intermediaire"
d$qualif.reg[d$qualif == "Cadre"] <- "Cadre"
d$qualif.reg[d$qualif == "Autre"] <- "Autre"
table(d$qualif.reg)
```

		Employe	Intermediaire
Autre	58	594	246
Ouvrier	495		

On aurait pu représenter ce recodage de manière plus compacte, notamment en commençant par copier le contenu de `qualif` dans `qualif.reg`, ce qui permet de ne pas s'occuper de ce qui ne change pas.

Il est cependant nécessaire de ne pas copier `qualif` sous forme de facteur, sinon on ne pourrait ajouter de nouvelles modalités. On copie donc la version caractères de `qualif` grâce à la fonction `as.character` :

```
R> d$qualif.reg <- as.character(d$qualif)
d$qualif.reg[d$qualif == "Ouvrier specialise"] <- "Ouvrier"
d$qualif.reg[d$qualif == "Ouvrier qualifie"] <- "Ouvrier"
d$qualif.reg[d$qualif == "Profession intermediaire"] <- "Intermediaire"
d$qualif.reg[d$qualif == "Technicien"] <- "Intermediaire"
table(d$qualif.reg)
```

		Employe	Intermediaire
Autre	58	594	246
Ouvrier	495		

On peut faire une version encore plus compacte en utilisant l'opérateur logique `ou` (`|`) :

```
R> d$qualif.reg <- as.character(d$qualif)
d$qualif.reg[d$qualif == "Ouvrier specialise" | d$qualif == "Ouvrier qualifi
e"] <- "Ouvrier"
d$qualif.reg[d$qualif == "Profession intermediaire" | d$qualif == "Technicie
n"] <- "Intermediaire"
table(d$qualif.reg)
```

Autre	Cadre	Employe	Intermediaire

58	260	594	246
Ouvrier			
495			

Enfin, pour terminer ce petit tour d'horizon, on peut également remplacer l'opérateur `|` par `%in%`, qui peut parfois être plus lisible :

```
R> d$qualif.reg <- as.character(d$qualif)
d$qualif.reg[d$qualif %in% c("Ouvrier specialise", "Ouvrier qualifie")] <- "Ouvrier"
d$qualif.reg[d$qualif %in% c("Profession intermediaire", "Technicien")] <- "Intermediaire"
table(d$qualif.reg)
```

Autre	Cadre	Employe	Intermediaire
58	260	594	246
Ouvrier			
495			

Dans tous les cas le résultat obtenu est une variable de type caractère. On pourra la convertir en facteur par un simple :

```
R> d$qualif.reg <- factor(d$qualif.reg)
```

Si on souhaite recoder les valeurs manquantes, il suffit de faire appel à la fonction `is.na` :

```
R> table(d$trav.satisf)
```

Satisfaction	Insatisfaction	Equilibre
480	117	451

```
R> d$trav.satisf.reg <- as.character(d$trav.satisf)
d$trav.satisf.reg[is.na(d$trav.satisf)] <- "Manquant"
table(d$trav.satisf.reg)
```

Equilibre	Insatisfaction	Manquant	Satisfaction
451	117	952	480

### NOTE

**questionr** propose une interface interactive pour le recodage d’une variable qualitative (renommage et regroupement de modalités). Cette fonction, nommée `irec`, s’utilise de la manière suivante :

```
R> irec(d, qualif)
```

RStudio va alors ouvrir une fenêtre semblable à l’image ci-dessous :

```
Recodage de qualif depuis d de classe factor.
## Recodage de d$qualif en d$qualif.rec
d$qualif.rec <- as.character(d$qualif)
d$qualif.rec[d$qualif == "Ouvrier specialise"] <- "Ouvrier specialise"
d$qualif.rec[d$qualif == "Ouvrier qualifie"] <- "Ouvrier qualifie"
d$qualif.rec[d$qualif == "Employe"] <- "Employe"
d$qualif.rec[d$qualif == "Technicien"] <- "Technicien"
d$qualif.rec[d$qualif == "Profession intermediaire"] <- "Profession intermediaire"
d$qualif.rec[d$qualif == "Cadre"] <- "Cadre"
d$qualif.rec[d$qualif == "Autre"] <- "Autre"
```

Capture de irec

Vous pouvez alors sélectionner différentes options, et pour chaque ancienne modalité, indiquer la nouvelle valeur correspondante. Pour regrouper des modalités, il suffit de leur assigner des nouvelles valeurs identiques. Dans tous les cas n’hésitez pas à expérimenter, l’interface se contente de générer du code R à copier/coller dans votre script mais ne l’exécute pas, et ne modifie donc jamais vos

données !

L'onglet *Vérification* affiche un tri croisé de l'ancienne et de la nouvelle variable pour vérifier que le recodage est correct. Une fois le résultat satisfaisant, vous pouvez récupérer le code généré dans l'onglet *Code* pour l'inclure dans votre script.

**NOTE**

Les exemples précédents montrent bien qu'il est parfois malaisé d'utiliser des facteurs lorsque l'on recode des variables. Les vecteurs labellisés sont, quant à eux, plus souples. **Attention**: avec des vecteurs labellisés, on utilisera les valeurs sous-jacentes et non les étiquettes pour écrire des conditions.

```
R> data(fecondite)
library(labelled)
describe(femmes$educ)
```

```
[2000 obs.] Niveau d'éducation
labelled double: 0 0 0 0 1 0 0 0 0 0 ...
min: 0 - max: 3 - NAs: 0 (0%) - 4 unique values
4 value labels: [0] aucun [1] primaire [2] secondaire [3] supérieur

      n      %
[0] aucun    1138  56.9
[1] primaire   460  23.0
[2] secondaire 348  17.4
[3] supérieur    54  2.7
Total        2000 100.0
```

```
R> femmes$educ2 <- 0
femmes$educ2[femmes$educ >= 2] <- 1
var_label(femmes$educ2) <- "A atteint un niveau secondaire ou supérieur ?"
val_labels(femmes$educ2) <- c(non = 0, oui = 1)
describe(femmes$educ2)
```

```
[2000 obs.] A atteint un niveau secondaire ou supérieur ?
labelled double: 0 0 0 0 0 0 0 0 0 ...
min: 0 - max: 1 - NAs: 0 (0%) - 2 unique values
2 value labels: [0] non [1] oui

      n      %
[0] non 1598  79.9
[1] oui  402  20.1
Total    2000 100.0
```

Il est aussi possible d'avoir recours à la méthode `recode` de `dplyr` pour les vecteurs labelisés à condition que l'extension `labelled` soit chargée en mémoire. Il faut indiquer les recodifications sous

la forme `ancienne_valeur = nouvelle_valeur`. Par défaut, les étiquettes de valeurs ne sont pas modifiées, sauf si l'on indique `.combine_value_labels = FALSE` (voir l'aide de `recode.haven_labelled`).

```
R> 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
```

```
R> library(labelled)
freq(femmes$educ)
```

```
R> femmes <- femmes %>% mutate(educ3 = recode(educ, `3` = 2))
freq(femmes$educ3)
```

```
R> femmes <- femmes %>% mutate(educ4 = recode(educ, `3` = 2, .combine_value_labels = TRUE))
freq(femmes$educ4)
```

## Variables calculées

La création d'une variable numérique à partir de calculs sur une ou plusieurs autres variables numériques se fait très simplement.

Supposons que l'on souhaite calculer une variable indiquant l'écart entre le nombre d'heures passées à regarder la télévision et la moyenne globale de cette variable. On pourrait alors faire :

```
R> range(d$heures.tv, na.rm = TRUE)
```

```
[1] 0 12
```

```
R> mean(d$heures.tv, na.rm = TRUE)
```

```
[1] 2.246566
```

```
R> d$ecart.heures.tv <- d$heures.tv - mean(d$heures.tv, na.rm = TRUE)
range(d$ecart.heures.tv, na.rm = TRUE)
```

```
[1] -2.246566 9.753434
```

```
R> mean(d$ecart.heures.tv, na.rm = TRUE)
```

```
[1] 4.714578e-17
```

Autre exemple tiré du jeu de données `rp99` : si on souhaite calculer le pourcentage d’actifs dans chaque commune, on peut diviser la population active `pop.act` par la population totale `pop.tot`.

```
R> data("rp99")
rp99$part.actifs <- rp99$pop.act / rp99$pop.tot * 100
```

## Combiner plusieurs variables

La combinaison de plusieurs variables se fait à l’aide des techniques d’indexation déjà décrites précédemment. Le plus compliqué est d’arriver à formuler des conditions parfois complexes de manière rigoureuse.

On peut ainsi vouloir combiner plusieurs variables qualitatives en une seule :

```
R> d$act.manuelles <- NA
d$act.manuelles[d$cuisine == "Oui" & d$bricol == "Oui"] <- "Cuisine et Bricolage"
d$act.manuelles[d$cuisine == "Oui" & d$bricol == "Non"] <- "Cuisine seulement"
d$act.manuelles[d$cuisine == "Non" & d$bricol == "Oui"] <- "Bricolage seulement"
d$act.manuelles[d$cuisine == "Non" & d$bricol == "Non"] <- "Ni cuisine ni bricolage"
table(d$act.manuelles)
```

Bricolage seulement	Cuisine et Bricolage
437	416
Cuisine seulement	Ni cuisine ni bricolage
465	682

On peut également combiner variables qualitatives et variables quantitatives :

```
R> d$age.sex <- NA
d$age.sex[d$sex == "Homme" & d$age < 40] <- "Homme moins de 40 ans"
d$age.sex[d$sex == "Homme" & d$age >= 40] <- "Homme plus de 40 ans"
d$age.sex[d$sex == "Femme" & d$age < 40] <- "Femme moins de 40 ans"
d$age.sex[d$sex == "Femme" & d$age >= 40] <- "Femme plus de 40 ans"
table(d$age.sex)
```

Femme moins de 40 ans	Femme plus de 40 ans
376	725
Homme moins de 40 ans	Homme plus de 40 ans
315	584

Les combinaisons de variables un peu complexes nécessitent parfois un petit travail de réflexion. En particulier, l'ordre des commandes de recodage a parfois une influence dans le résultat final.

Pour combiner rapidement plusieurs variables entre elles, on peut aussi avoir recours à la fonction `interaction` qui créera un facteur avec un niveau pour chaque combinaison de modalités des variables sources.

```
R> d$age20.sex <- interaction(d$sex, d$age20)
table(d$age20.sex)
```

Homme.<20ans	Femme.<20ans	Homme.21-40 ans
34	38	291
Femme.21-40 ans	Homme.41-60ans	Femme.41-60ans
369	352	428

Homme .61-80ans	Femme .61-80ans	Homme .>80ans
205	231	17
Femme .>80ans		
35		

## Variables scores

Une variable score est une variable calculée en additionnant des poids accordés aux modalités d'une série de variables qualitatives.

Pour prendre un exemple tout à fait arbitraire, imaginons que nous souhaitons calculer un score d'activités extérieures. Dans ce score on considère que le fait d'aller au cinéma « pèse » 10, celui de pêcher ou chasser vaut 30 et celui de faire du sport vaut 20. On pourrait alors calculer notre score de la manière suivante :

```
R> d$score.ext <- 0
d$score.ext[d$cinema == "Oui"] <- d$score.ext[d$cinema == "Oui"] + 10
d$score.ext[d$peche.chasse == "Oui"] <- d$score.ext[d$peche.chasse == "Oui"]
+ 30
d$score.ext[d$sport == "Oui"] <- d$score.ext[d$sport == "Oui"] + 20
table(d$score.ext)
```

0	10	20	30	40	50	60
800	342	229	509	31	41	48

Cette notation étant un peu lourde, on peut l'alléger un peu en utilisant la fonction `ifelse`. Celle-ci prend en argument une condition et deux valeurs. Si la condition est vraie elle retourne la première valeur, sinon elle retourne la seconde.

```
R> d$score.ext <- 0
d$score.ext <- ifelse(d$cinema == "Oui", 10, 0) + ifelse(d$peche.chasse == "Ou
i", 30, 0) + ifelse(d$sport == "Oui", 20, 0)
table(d$score.ext)
```

0	10	20	30	40	50	60
800	342	229	509	31	41	48

## Vérification des recodages

Il est très important de vérifier, notamment après les recodages les plus complexes, qu'on a bien obtenu le résultat escompté. Les deux points les plus sensibles étant les valeurs manquantes et les erreurs dans les conditions.

Pour vérifier tout cela, le plus simple est sans doute de faire des tableaux croisés entre la variable recodée et celles ayant servi au recodage, à l'aide des fonctions `table` ou `xtabs`, et de vérifier le nombre de valeurs manquantes dans la variable recodée avec `summary`, `freq` ou `table`.

Par exemple :

```
R> d$act.manuelles <- NA
d$act.manuelles[d$cuisine == "Oui" & d$bricol == "Oui"] <- "Cuisine et Bricolage"
d$act.manuelles[d$cuisine == "Oui" & d$bricol == "Non"] <- "Cuisine seulement"
d$act.manuelles[d$cuisine == "Non" & d$bricol == "Oui"] <- "Bricolage seulement"
d$act.manuelles[d$cuisine == "Non" & d$bricol == "Non"] <- "Ni cuisine ni bricolage"
table(d$act.manuelles, d$cuisine)
```

	Non	Oui
Bricolage seulement	437	0
Cuisine et Bricolage	0	416
Cuisine seulement	0	465
Ni cuisine ni bricolage	682	0

```
R> table(d$act.manuelles, d$bricol)
```

	Non	Oui
Bricolage seulement	0	437
Cuisine et Bricolage	0	416
Cuisine seulement	465	0
Ni cuisine ni bricolage	682	0

## Facteurs et forcats

**forcats** est une extension facilitant la manipulation des variables qualitatives, qu’elles soient sous forme de vecteurs `character` ou de facteurs. Elle fait partie du **tidyverse**, et est donc automatiquement chargée par :

```
R> library(tidyverse)

-- Attaching packages ----- tidyverse 1.3.0 --

v ggplot2 3.3.2     v purrr    0.3.4
v tibble   3.0.4     v stringr  1.4.0
v tidyverse 1.1.2     v forcats  0.5.0
v readr    1.4.0

-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

## Modifier les modalités d’une variable qualitative

Une opération courante consiste à modifier les valeurs d’une variable qualitative, que ce soit pour avoir des intitulés plus courts ou plus clairs ou pour regrouper des modalités entre elles.

Il existe plusieurs possibilités pour effectuer ce type de recodage, mais ici on va utiliser la fonction `fct_recode` de l’extension **forcats**. Celle-ci prend en argument une liste de recodages sous la forme `"Nouvelle valeur" = "Ancienne valeur"`.

Un exemple :

```
R> f <- c("Pomme", "Poire", "Pomme", "Cerise")
f <- fct_recode(f,
  "Fraise" = "Pomme",
  "Ananas" = "Poire"
)
f
```

```
[1] Fraise Ananas Fraise Cerise
Levels: Cerise Ananas Fraise
```

Autre exemple sur une “vraie” variable :

```
R> freq(hdv2003$qualif)

R> hdv2003$qualif5 <- fct_recode(hdv2003$qualif,
  "Ouvrier" = "Ouvrier specialise",
  "Ouvrier" = "Ouvrier qualifie",
  "Interm" = "Technicien",
  "Interm" = "Profession intermediaire"
)
freq(hdv2003$qualif5)
```

Attention, les anciennes valeurs saisies doivent être exactement égales aux valeurs des modalités de la variable recodée : toute différence d’accent ou d’espace fera que ce recodage ne sera pas pris en compte. Dans ce cas, `forcats` affiche un avertissement nous indiquant qu’une valeur saisie n’a pas été trouvée dans les modalités de la variable :

```
R> hdv2003$qualif_test <- fct_recode(hdv2003$qualif,
  "Ouvrier" = "Ouvrier spécialisé",
  "Ouvrier" = "Ouvrier qualifié"
)
```

```
Warning: Unknown levels in `f`: Ouvrier spécialisé, Ouvrier
qualifié
```

Si on souhaite recoder une modalité de la variable en `NA`, il faut (contre intuitivement) lui assigner la valeur `NULL` :

```
R> hdv2003$qualif_rec <- fct_recode(hdv2003$qualif, NULL = "Autre")

freq(hdv2003$qualif_rec)
```

À l’inverse, si on souhaite recoder les `NA` d’une variable, on utilisera la fonction `fct_explicit_na`, qui convertit toutes les valeurs manquantes (`NA`) d’un facteur en une modalité spécifique :

```
R> hdv2003$qualif_rec <- fct_explicit_na(hdv2003$qualif, na_level = "(Manquant)")

freq(hdv2003$qualif_rec)
```

D’autres fonctions sont proposées par `forcats` pour faciliter certains recodage, comme `fct_collapse`, qui propose une autre syntaxe pratique quand on doit regrouper ensemble des modalités :

```
R> hdv2003$qualif_rec <- fct_collapse(hdv2003$qualif,
  "Ouvrier" = c("Ouvrier specialise", "Ouvrier qualifie"),
  "Interm" = c("Technicien", "Profession intermediaire")
)

freq(hdv2003$qualif_rec)
```

`fct_other`, qui regroupe une liste de modalités en une seule modalité “Other” :

```
R> hdv2003$qualif_rec <- fct_other(hdv2003$qualif,
  drop = c(
    "Ouvrier specialise", "Ouvrier qualifie",
    "Cadre", "Autre"
  )
)

freq(hdv2003$qualif_rec)
```

`fct_lump`, qui regroupe automatiquement les modalités les moins fréquentes en une seule modalité “Other” (avec possibilité d’indiquer des seuils de regroupement) :

```
R> hdv2003$qualif_rec <- fct_lump(hdv2003$qualif)

freq(hdv2003$qualif_rec)
```

## Ordonner les modalités d’une variable qualitative

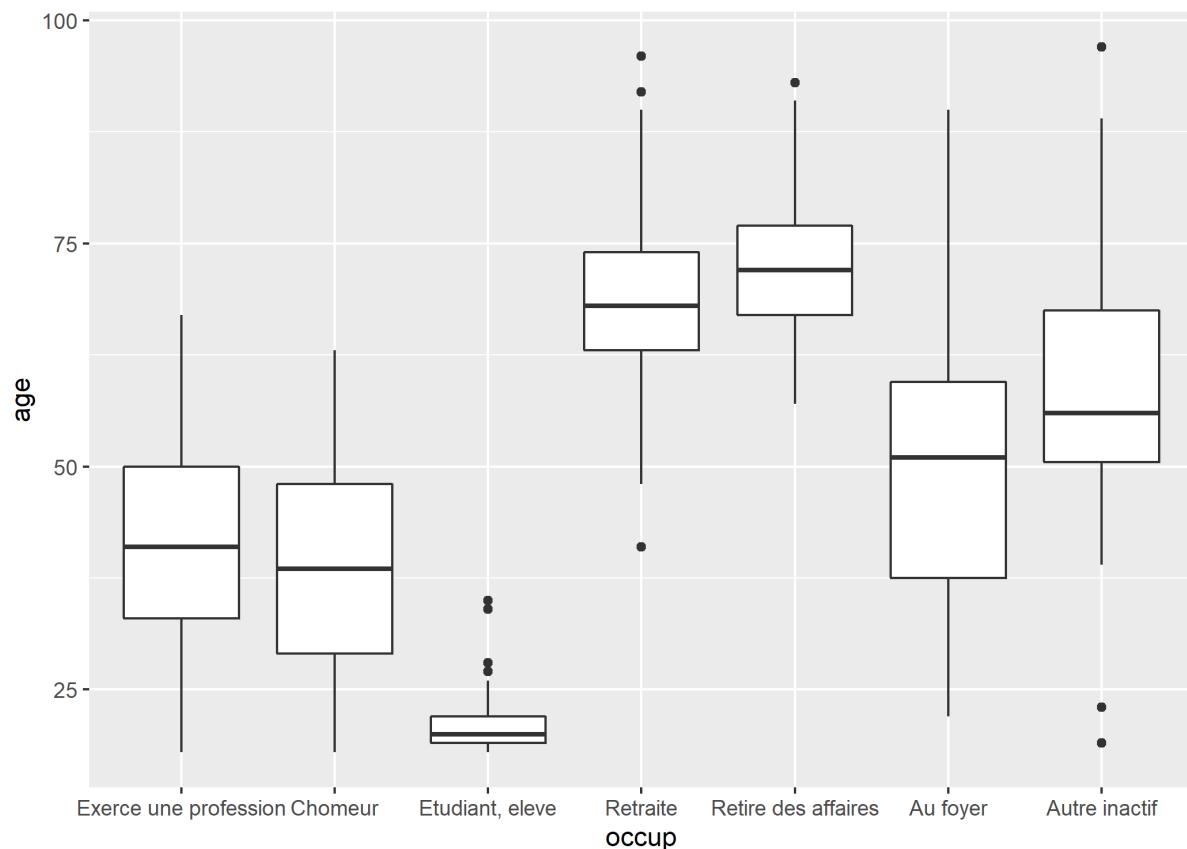
L’avantage des facteurs (par rapport aux vecteurs de type `character`) est que leurs modalités peuvent être ordonnées, ce qui peut faciliter la lecture de tableaux ou graphiques.

On peut ordonner les modalités d’un facteur manuellement, par exemple avec la fonction `fct_relevel()` de l’extension `forcats` :

```
R> hdv2003$qualif_rec <- fct_relevel(
  hdv2003$qualif,
  "Cadre", "Profession intermediaire", "Technicien",
  "Employe", "Ouvrier qualifie", "Ouvrier specialise",
  "Autre"
)
freq(hdv2003$qualif_rec)
```

Une autre possibilité est d’ordonner les modalités d’un facteur selon les valeurs d’une autre variable. Par exemple, si on représente le boxplot de la répartition de l’âge selon le statut d’occupation :

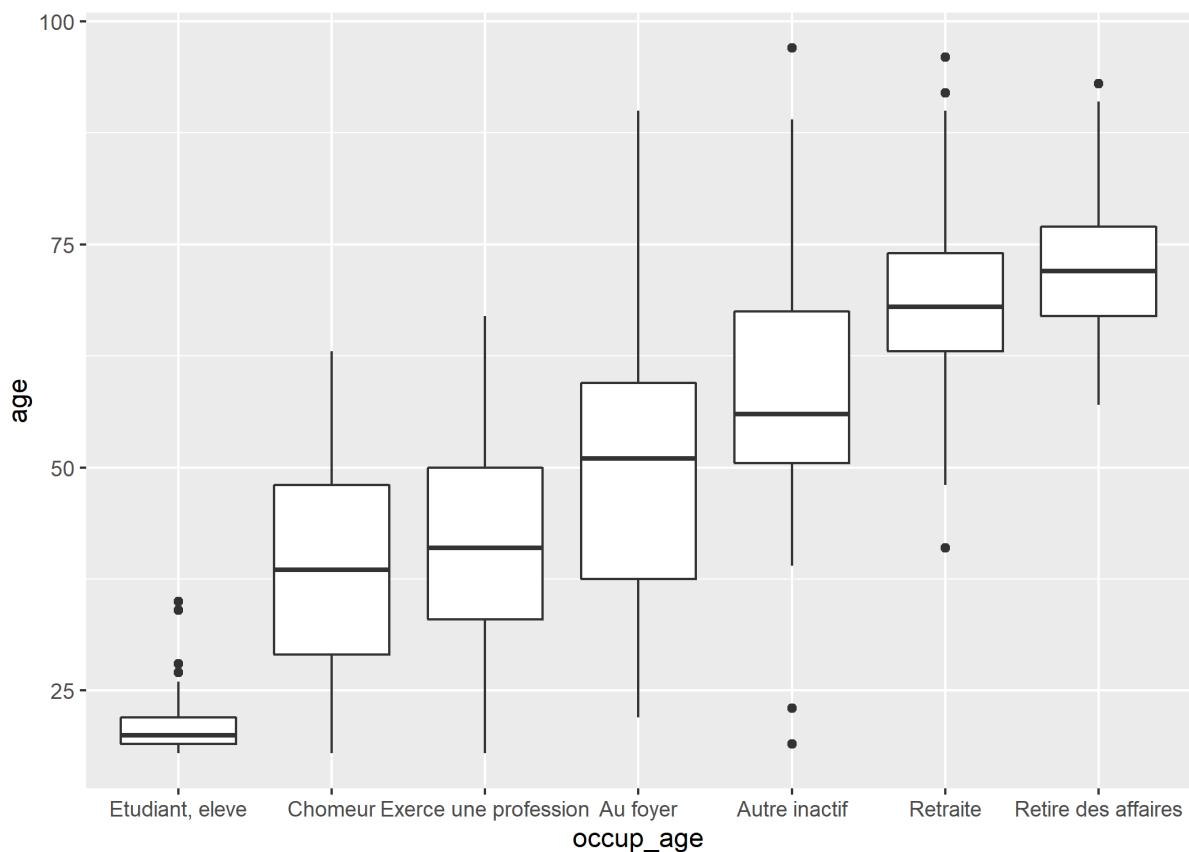
```
R> library(ggplot2)
ggplot(hdv2003) +
  geom_boxplot(aes(x = occup, y = age))
```



Le graphique pourrait être plus lisible si les modalités étaient triées par âge median croissant. Ceci est possible en utilisant `fct_reorder`. Celle-ci prend 3 arguments : le facteur à réordonner, la variable dont les valeurs doivent être utilisées pour ce réordonnement, et enfin une fonction à appliquer à cette deuxième variable.

```
R> hdv2003$occup_age <- fct_reorder(hdv2003$occup, hdv2003$age, median)

ggplot(hdv2003) +
  geom_boxplot(aes(x = occup_age, y = age))
```



## Combiner plusieurs variables

Parfois, on veut créer une nouvelle variable en partant des valeurs d'une ou plusieurs autres variables. Dans ce cas on peut utiliser les fonctions `if_else` pour les cas les plus simples, ou `case_when` pour les cas plus complexes. Ces deux fonctions sont incluses dans l'extension `dplyr`, qu'il faut donc avoir chargé précédemment (voir le chapitre consacré à `dplyr`, page 233).

### `if_else`

`if_else` prend trois arguments : un test, une valeur à renvoyer si le test est vrai, et une valeur à renvoyer si le test est faux.

Voici un exemple simple :

```
R> v <- c(12, 14, 8, 16)
  if_else(v > 10, "Supérieur à 10", "Inférieur à 10")
```

```
[1] "Supérieur à 10" "Supérieur à 10" "Inférieur à 10"
[4] "Supérieur à 10"
```

La fonction devient plus intéressante avec des tests combinant plusieurs variables. Par exemple, imaginons qu'on souhaite créer une nouvelle variable indiquant les hommes de plus de 60 ans :

```
R> hdv2003$statut <- if_else(hdv2003$sex == "Homme" & hdv2003$age > 60,
  "Homme de plus de 60 ans",
  "Autre"
)

freq(hdv2003$statut)
```

## case\_when

`case_when` est une génération du `if_else` qui permet d'indiquer plusieurs tests et leurs valeurs associées.

Imaginons qu'on souhaite créer une nouvelle variable permettant d'identifier les hommes de plus de 60 ans, les femmes de plus de 60 ans, et les autres. On peut utiliser la syntaxe suivante :

```
R> hdv2003$statut <- case_when(
  hdv2003$age > 60 & hdv2003$sex == "Homme" ~ "Homme de plus de 60 ans",
  hdv2003$age > 60 & hdv2003$sex == "Femme" ~ "Femme de plus de 60 ans",
  TRUE ~ "Autre"
)

freq(hdv2003$statut)
```

`case_when` prend en arguments une série d'instructions sous la forme `condition ~ valeur`. Il les exécute une par une, et dès qu'une `condition` est vraie, il renvoie la `valeur` associée.

La clause `TRUE ~ "Autre"` permet d'assigner une valeur à toutes les lignes pour lesquelles aucune des conditions précédentes n'est vraie.

## IMPORTANT

**Attention :** comme les conditions sont testées l’une après l’autre et que la valeur renvoyée est celle correspondant à la première condition vraie, l’ordre de ces conditions est très important. Il faut absolument aller du plus spécifique au plus général.

Par exemple le recodage suivant ne fonctionne pas :

```
R> hdv2003$statut <- case_when(  
  hdv2003$sexe == "Homme" ~ "Homme",  
  hdv2003$sexe == "Homme" & hdv2003$age > 60 ~ "Homme de plus de 60 ans",  
  TRUE ~ "Autre"  
)  
  
freq(hdv2003$statut)
```

Comme la condition `sexe == "Homme"` est plus générale que `sexe == "Homme" & age > 60`, cette deuxième condition n'est jamais testée ! On n'obtiendra jamais la valeur correspondante.

Pour que ce recodage fonctionne il faut donc changer l’ordre des conditions pour aller du plus spécifique au plus général :

```
R> hdv2003$statut <- case_when(  
  hdv2003$sexe == "Homme" & hdv2003$age > 60 ~ "Homme de plus de 60 ans",  
  hdv2003$sexe == "Homme" ~ "Homme",  
  TRUE ~ "Autre"  
)  
  
freq(hdv2003$statut)
```

Vous pouvez trouver des exercices avec leur solution dans l'[Introduction à R et au tidyverse](#) de Julien Barnier.

Pour aller plus loin, [R for Data Science](#) de Garrett Grolemund et Hadley Wickham.

## Recodage et data.table

Nous aborderons dans un prochain chapitre, page 251 l’extension [data.table](#) qui étend les tableaux de données et modifie complètement la syntaxe utilisée entre les crochets. Elle nécessite un petit temps d’adaptation mais, une fois maîtrisée, elle facilite le quotidien lorsqu’il s’agit de manipuler et recoder les données. Ci-dessous, un petit avant-goût, reprenons quelques exemples précédents.

```
R> library(data.table)
dt <- data.table(hdv2003)

dt[, score.ext := 0]
dt[cinema == "Oui", score.ext := score.ext + 10]
dt[peche.chasse == "Oui", score.ext := score.ext + 30]
dt[sport == "Oui", score.ext := score.ext + 20]
table(dt$score.ext)

dt[cuisine == "Oui" & bricol == "Oui", act.manuelles := "Cuisine et Bricolage"]
dt[cuisine == "Oui" & bricol == "Non", act.manuelles := "Cuisine seulement"]
dt[cuisine == "Non" & bricol == "Oui", act.manuelles := "Bricolage seulement"]
dt[cuisine == "Non" & bricol == "Non", act.manuelles := "Ni cuisine ni bricolage"]
```

```
R> table(dt$act.manuelles)
```

Bricolage seulement	Cuisine et Bricolage
437	416
Cuisine seulement	Ni cuisine ni bricolage
465	682

```
R> dt[cuisine == "Oui" & bricol == "Oui", act.manuelles := "Cuisine et Bricolage"]
dt[cuisine == "Oui" & bricol == "Non", act.manuelles := "Cuisine seulement"]
dt[cuisine == "Non" & bricol == "Oui", act.manuelles := "Bricolage seulement"]
dt[cuisine == "Non" & bricol == "Non", act.manuelles := "Ni cuisine ni bricolage"]
```

```
R> table(dt$act.manuelles)
```

Bricolage seulement	Cuisine et Bricolage
437	416
Cuisine seulement	Ni cuisine ni bricolage
465	682



# Manipuler les données avec dplyr

Préparation . . . . .	234
Les verbes de dplyr . . . . .	235
slice . . . . .	235
filter . . . . .	236
select, rename et relocate . . . . .	236
arrange . . . . .	238
mutate . . . . .	239
Enchaîner les opérations avec le «pipe» . . . . .	240
Opérations groupées . . . . .	242
group_by . . . . .	242
summarise et count . . . . .	244
Grouper selon plusieurs variables . . . . .	245
Autres fonctions utiles . . . . .	246
sample_n et sample_frac . . . . .	247
lead et lag . . . . .	247
tally . . . . .	248
distinct . . . . .	248
Ressources . . . . .	249
dplyr et data.table . . . . .	249
dplyr et survey . . . . .	249

## NOTE

La version originale de ce chapitre a été écrite par Julien Barnier dans le cadre de son [Introduction à R et au tidyverse](#).

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #04 (manipuler les données avec dplyr) sur [YouTube](#).

**dplyr** est une extension facilitant le traitement et la manipulation de données contenues dans une ou plusieurs tables (qu'il s'agisse de *data frame* ou de *tibble*). Elle propose une syntaxe claire et cohérente, sous formes de verbes, pour la plupart des opérations de ce type.

Par ailleurs, les fonctions de **dplyr** sont en général plus rapides que leur équivalent sous **R** de base, elles permettent donc de traiter des données de grande dimension<sup>1</sup>.

**dplyr** part du principe que les données sont *tidy* (voir la section consacrée aux tidy data, page 60). Les fonctions de l’extension peuvent s’appliquer à des tableaux de type `data.frame` ou `tibble`, et elles retournent systématiquement un `tibble` (voir la section dédiée, page 61).

# Préparation

**dplyr** fait partie du cœur du **tidyverse**, elle est donc chargée automatiquement avec :

```
R> library(tidyverse)
```

On peut également la charger individuellement avec :

```
R> library(dplyr)
```

Dans ce qui suit on va utiliser les données du jeu de données **nycflights13**, contenu dans l’extension du même nom (qu'il faut donc avoir installé). Celui-ci correspond aux données de tous les vols au départ d'un des trois aéroports de New-York en 2013. Il a la particularité d'être réparti en trois tables :

- `flights` contient des informations sur les vols : date, départ, destination, horaires, retard...
- `airports` contient des informations sur les aéroports
- `airlines` contient des données sur les compagnies aériennes

On va charger les trois tables du jeu de données :

---

1. Elles sont cependant moins rapides que les fonctions de `data.table`, voir le chapitre dédié, page 251

```
R> library(nycflights13)
## Chargement des trois tables du jeu de données
data(flights)
data(airports)
data(airlines)
```

Normalement trois objets correspondant aux trois tables ont dû apparaître dans votre environnement.

Ces trois tableaux sont au format **tibble**. Il s'agit d'une extension des tableaux de données utilisé par le **tidyverse**. Les **tibble** {data-pkg="tibble"} s'utilisent comme des **data.frame**, avec justes quelques différentes :

- leur classe est `c("tbl_df", "tbl", "data.frame")` ;
- leur présentation dans la console est améliorée ;
- `df[, j]` renvoie toujours un **tibble** avec une seule colonne (et non le contenu de cette colonne que l'on obtient avec `df[[j]]`) ;
- les colonnes d'un **tibble** peuvent être des listes ;
- à la différence d'un tableau de données classique où il est possible d'utiliser un nom partiel (par exemple écrire `df$ab` pour obtenir `df$abc`), il est obligatoire d'utiliser les noms complets avec un **tibble**.

Pour convertir un tableau de données en **tibble**, on utilisera la fonction **as\_tibble**.

## Les verbes de dplyr

La manipulation de données avec **dplyr** se fait en utilisant un nombre réduit de verbes, qui correspondent chacun à une action différente appliquée à un tableau de données.

### slice

Le verbe **slice** sélectionne des lignes du tableau selon leur position. On lui passe un chiffre ou un vecteur de chiffres.

Si on souhaite sélectionner la 345e ligne du tableau **airports** :

```
R> slice(airports, 345)
```

Si on veut sélectionner les 5 premières lignes :

```
R> slice(airports, 1:5)
```

## filter

`filter` sélectionne des lignes d'un tableau de données selon une condition. On lui passe en paramètre un test, et seules les lignes pour lesquelles ce test renvoie `TRUE` (vrai) sont conservées.

Par exemple, si on veut sélectionner les vols du mois de janvier, on peut filtrer sur la variable `month` de la manière suivante :

```
R> filter(flights, month == 1)
```

Si on veut uniquement les vols avec un retard au départ (variable `dep_delay`) compris entre 10 et 15 minutes :

```
R> filter(flights, dep_delay >= 10 & dep_delay <= 15)
```

Si on passe plusieurs arguments à `filter`, celui-ci rajoute automatiquement une condition `et` entre les conditions. La ligne ci-dessus peut donc également être écrite de la manière suivante, avec le même résultat :

```
R> filter(flights, dep_delay >= 10, dep_delay <= 15)
```

Enfin, on peut également placer des fonctions dans les tests, qui nous permettent par exemple de sélectionner les vols avec la plus grande distance :

```
R> filter(flights, distance == max(distance))
```

## select, rename et relocate

`select` permet de sélectionner des colonnes d'un tableau de données. Ainsi, si on veut extraire les colonnes `lat` et `lon` du tableau `airports` :

```
R> select(airports, lat, lon)
```

Si on fait précéder le nom d'un `-`, la colonne est éliminée plutôt que sélectionnée :

```
R> select(airports, -lat, -lon)
```

`select` comprend toute une série de fonctions facilitant la sélection de multiples colonnes. Par exemple, `starts_with`, `ends_width`, `contains` ou `matches` permettent d'exprimer des conditions sur les

noms de variables :

```
R> select(flights, starts_with("dep_"))
```

La syntaxe `colonne1:colonne2` permet de sélectionner toutes les colonnes situées entre `colonne1` et `colonne2` incluses<sup>2</sup>:

```
R> select(flights, year:day)
```

`all_of` et `any_of` permettent de fournir une liste de variables à extraire sous forme de vecteur textuel. Alors que `all_of` renverra une erreur si une variable n'est pas trouvée dans le tableau de départ, `any_of` sera moins stricte.

```
R> select(flights, all_of(c("year", "month", "day")))
```

```
R> select(flights, all_of(c("century", "year", "month", "day")))
```

```
Erreur : Can't subset columns that don't exist.  
x Column `century` doesn't exist.
```

```
R> select(flights, any_of(c("century", "year", "month", "day")))
```

`where` permet de sélectionner des variables à partir d'une fonction qui renvoie une valeur logique. Par exemple, pour sélectionner seulement les variables textuelles.

```
R> select(flights, where(is.character))
```

`select` peut être utilisée pour réordonner les colonnes d'une table en utilisant la fonction `everything()`, qui sélectionne l'ensemble des colonnes non encore sélectionnées. Ainsi, si on souhaite faire passer la colonne `name` en première position de la table `airports`, on peut faire :

```
R> select(airports, name, everything())
```

Pour réordonner des colonnes, on pourra aussi avoir recours à `relocate` en indiquant les premières variables. Il n'est pas nécessaire d'ajouter `everything()` car avec `relocate` toutes les variables sont conservées.

---

2. À noter que cette opération est un peu plus "fragile" que les autres, car si l'ordre des colonnes change elle peut renvoyer un résultat différent.

```
R> relocate(airports, lon, lat, name)
```

Une variante de `select` est `rename`<sup>3</sup>, qui permet de renommer facilement des colonnes. On l'utilise en lui passant des paramètres de la forme `nouveau_nom = ancien_nom`. Ainsi, si on veut renommer les colonnes `lon` et `lat` de `airports` en `longitude` et `latitude`:

```
R> rename(airports, longitude = lon, latitude = lat)
```

Si les noms de colonnes comportent des espaces ou des caractères spéciaux, on peut les entourer de guillemets (" ") ou de quotes inverses (` ):

```
R> tmp <- rename(flights,
  "retard départ" = dep_delay,
  "retard arrivée" = arr_delay)
select(tmp, `retard départ`, `retard arrivée`)
```

## arrange

`arrange` réordonne les lignes d'un tableau selon une ou plusieurs colonnes.

Ainsi, si on veut trier le tableau `flights` selon le retard au départ croissant :

```
R> arrange(flights, dep_delay)
```

On peut trier selon plusieurs colonnes. Par exemple selon le mois, puis selon le retard au départ :

```
R> arrange(flights, month, dep_delay)
```

Si on veut trier selon une colonne par ordre décroissant, on lui applique la fonction `desc()` :

```
R> arrange(flights, desc(dep_delay))
```

Combiné avec `slice`, `arrange` permet par exemple de sélectionner les trois vols ayant eu le plus de retard :

```
R> tmp <- arrange(flights, desc(dep_delay))
slice(tmp, 1:3)
```

---

3. Il est également possible de renommer des colonnes directement avec `select`, avec la même syntaxe que pour `rename`.

## mutate

`mutate` permet de créer de nouvelles colonnes dans le tableau de données, en général à partir de variables existantes.

Par exemple, la table `airports` contient l'altitude de l'aéroport en pieds. Si on veut créer une nouvelle variable `alt_m` avec l'altitude en mètres, on peut faire :

```
R> airports <- mutate(airports, alt_m = alt / 3.2808)
   select(airports, name, alt, alt_m)
```

On peut créer plusieurs nouvelles colonnes en une seule fois, et les expressions successives peuvent prendre en compte les résultats des calculs précédents. L'exemple suivant convertit d'abord la distance en kilomètres dans une variable `distance_km`, puis utilise cette nouvelle colonne pour calculer la vitesse en km/h.

```
R> flights <- mutate(flights,
  distance_km = distance / 0.62137,
  vitesse = distance_km / air_time * 60)
  select(flights, distance, distance_km, vitesse)
```

À noter que `mutate` est évidemment parfaitement compatible avec les fonctions vues dans le chapitre sur les recodages, page 197 : fonctions de `forcats`, `if_else`, `case_when` ...

L'avantage d'utiliser `mutate` est double. D'abord il permet d'éviter d'avoir à saisir le nom du tableau de données dans les conditions d'un `if_else` ou d'un `case_when` :

```
R> flights <- mutate(flights,
  type_retard = case_when(
    dep_delay > 0 & arr_delay > 0 ~ "Retard départ et arrivé",
    dep_delay > 0 & arr_delay <= 0 ~ "Retard départ",
    dep_delay <= 0 & arr_delay > 0 ~ "Retard arrivée",
    TRUE ~ "Aucun retard"))
```

Utiliser `mutate` pour les recodages permet aussi de les intégrer dans un *pipeline* de traitement de données, concept présenté dans la section suivante.

Citons également les fonctions `recode` et `recode_factor`.

```
R> flights$month_name <- recode_factor(flights$month,
  "1" = "Jan",
  "2" = "Feb",
  "3" = "Mar",
  "4" = "Apr",
  "5" = "May",
  "6" = "Jun",
  "7" = "Jul",
  "8" = "Aug",
  "9" = "Sep",
  "10" = "Oct",
  "11" = "Nov",
  "12" = "Dec"
)
```

## Enchaîner les opérations avec le «pipe»

Quand on manipule un tableau de données, il est très fréquent d’enchaîner plusieurs opérations. On va par exemple filtrer pour extraire une sous-population, sélectionner des colonnes puis trier selon une variable.

Dans ce cas on peut le faire de deux manières différentes. La première est d’effectuer toutes les opérations en une fois en les «emboîtant» :

```
R> arrange(select(filter(flights, dest == "LAX"), dep_delay, arr_delay), dep dela
y)
```

Cette notation a plusieurs inconvénients :

- elle est peu lisible
- les opérations apparaissent dans l’ordre inverse de leur réalisation. Ici on effectue d’abord le `filter`, puis le `select`, puis le `arrange`, alors qu’à la lecture du code c’est le `arrange` qui apparaît en premier.
- Il est difficile de voir quel paramètre se rapporte à quelle fonction

Une autre manière de faire est d’effectuer les opérations les unes après les autres, en stockant les résultats intermédiaires dans un objet temporaire :

```
R> tmp <- filter(flights, dest == "LAX")
tmp <- select(tmp, dep_delay, arr_delay)
arrange(tmp, dep_delay)
```

C'est nettement plus lisible, l'ordre des opérations est le bon, et les paramètres sont bien rattachés à leur fonction. Par contre, ça reste un peu “verbeux”, et on crée un objet temporaire `tmp` dont on n'a pas

réellement besoin.

Pour simplifier et améliorer encore la lisibilité du code, on va utiliser un nouvel opérateur, baptisé *pipe*<sup>4</sup>. Le *pipe* se note `%>%`, et son fonctionnement est le suivant : si j'exécute `expr %>% f`, alors le résultat de l'expression `expr`, à gauche du *pipe*, sera passé comme premier argument à la fonction `f`, à droite du *pipe*, ce qui revient à exécuter `f(expr)`.

Ainsi les deux expressions suivantes sont rigoureusement équivalentes :

```
R> filter(flights, dest == "LAX")
```

```
R> flights %>% filter(dest == "LAX")
```

Ce qui est intéressant dans cette histoire, c'est qu'on va pouvoir enchaîner les *pipes*. Plutôt que d'écrire :

```
R> select(filter(flights, dest == "LAX"), dep_delay, arr_delay)
```

On va pouvoir faire :

```
R> flights %>% filter(dest == "LAX") %>% select(dep_delay, arr_delay)
```

À chaque fois, le résultat de ce qui se trouve à gauche du *pipe* est passé comme premier argument à ce qui se trouve à droite : on part de l'objet `flights`, qu'on passe comme premier argument à la fonction `filter`, puis on passe le résultat de ce `filter` comme premier argument du `select`.

Le résultat final est le même avec les deux syntaxes, mais avec le *pipe* l'ordre des opérations correspond à l'ordre naturel de leur exécution, et on n'a pas eu besoin de créer d'objet intermédiaire.

Si la liste des fonctions enchaînées est longue, on peut les répartir sur plusieurs lignes à condition que l'opérateur `%>%` soit en fin de ligne :

```
R> flights %>%  
  filter(dest == "LAX") %>%  
  select(dep_delay, arr_delay) %>%  
  arrange(dep_delay)
```

#### NOTE

On appelle une suite d'instructions de ce type un *pipeline*.

---

4. Le *pipe* a été introduit à l'origine par l'extension `magrittr`, et repris par `dplyr`

Évidemment, il est naturel de vouloir récupérer le résultat final d’un *pipeline* pour le stocker dans un objet. Par exemple, on peut stocker le résultat du *pipeline* ci-dessus dans un nouveau tableau `delay_la` de la manière suivante :

```
R> delay_la <- flights %>%
  filter(dest == "LAX") %>%
  select(dep_delay, arr_delay) %>%
  arrange(dep_delay)
```

Dans ce cas, `delay_la` contiendra le tableau final, obtenu après application des trois instructions `filter`, `select` et `arrange`.

Cette notation n’est pas forcément très intuitive au départ. Il faut bien comprendre que c’est le résultat final, une fois application de toutes les opérations du *pipeline*, qui est renvoyé et stocké dans l’objet en début de ligne.

Une manière de le comprendre peut être de voir que la notation suivante :

```
R> delay_la <- flights %>%
  filter(dest == "LAX") %>%
  select(dep_delay, arr_delay)
```

est équivalente à :

```
R> delay_la <- (flights %>% filter(dest == "LAX") %>% select(dep_delay, arr_delay))
```

#### NOTE

L’utilisation du *pipe* n’est pas obligatoire, mais elle rend les scripts plus lisibles et plus rapides à saisir. On l’utilisera donc dans ce qui suit.

## Opérations groupées

### group\_by

Un élément très important de `dplyr` est la fonction `group_by`. Elle permet de définir des groupes de lignes à partir des valeurs d’une ou plusieurs colonnes. Par exemple, on peut grouper les vols selon leur mois :

```
R> flights %>% group_by(month)
```

Par défaut ceci ne fait rien de visible, à part l'apparition d'une mention *Groups* dans l'affichage du résultat. Mais à partir du moment où des groupes ont été définis, les verbes comme `slice`, `mutate` ou `summarise` vont en tenir compte lors de leurs opérations.

Par exemple, si on applique `slice` à un tableau préalablement groupé, il va sélectionner les lignes aux positions indiquées pour chaque groupe. Ainsi la commande suivante affiche le premier vol de chaque mois, selon leur ordre d'apparition dans le tableau :

```
R> flights %>% group_by(month) %>% slice(1)
```

Idem pour `mutate` : les opérations appliquées lors du calcul des valeurs des nouvelles colonnes sont appliquée groupe de lignes par groupe de lignes. Dans l'exemple suivant, on ajoute une nouvelle colonne qui contient le retard moyen du mois correspondant :

```
R> flights %>%
  group_by(month) %>%
  mutate(mean_delay_month = mean(dep_delay, na.rm = TRUE)) %>%
  select(dep_delay, month, mean_delay_month)
```

Ceci peut permettre, par exemple, de déterminer si un retard donné est supérieur ou inférieur au retard moyen du mois en cours.

`group_by` peut aussi être utile avec `filter`, par exemple pour sélectionner les vols avec le retard au départ le plus important pour chaque mois :

```
R> flights %>%
  group_by(month) %>%
  filter(dep_delay == max(dep_delay, na.rm = TRUE))
```

### IMPORTANT

**Attention :** la clause `group_by` marche pour les verbes déjà vus précédemment, sauf pour `arrange`, qui par défaut trie la table sans tenir compte des groupes. Pour obtenir un tri par groupe, il faut lui ajouter l'argument `.by_group = TRUE`.

On peut voir la différence en comparant les deux résultats suivants :

```
R> flights %>%
  group_by(month) %>%
  arrange(desc(dep_delay))
```

```
R> flights %>%
  group_by(month) %>%
  arrange(desc(dep_delay), .by_group = TRUE)
```

## summarise et count

`summarise` permet d’agréger les lignes du tableau en effectuant une opération “résumée” sur une ou plusieurs colonnes. Par exemple, si on souhaite connaître les retards moyens au départ et à l’arrivée pour l’ensemble des vols du tableau `flights` :

```
R> flights %>%
  summarise(retard_dep = mean(dep_delay, na.rm=TRUE),
            retard_arr = mean(arr_delay, na.rm=TRUE))
```

Cette fonction est en général utilisée avec `group_by`, puisqu’elle permet du coup d’agréger et résumer les lignes du tableau groupe par groupe. Si on souhaite calculer le délai maximum, le délai minimum et le délai moyen au départ pour chaque mois, on pourra faire :

```
R> flights %>%
  group_by(month) %>%
  summarise(max_delay = max(dep_delay, na.rm=TRUE),
            min_delay = min(dep_delay, na.rm=TRUE),
            mean_delay = mean(dep_delay, na.rm=TRUE))
```

```
`summarise()` ungrouping output (override with `.`groups` argument)
```

`summarise` dispose d’un opérateur spécial, `n()`, qui retourne le nombre de lignes du groupe. Ainsi si on veut le nombre de vols par destination, on peut utiliser :

```
R> flights %>%
  group_by(dest) %>%
  summarise(nb = n())
```

```
`summarise()` ungrouping output (override with `.`groups` argument)
```

`n()` peut aussi être utilisée avec `filter` et `mutate`.

À noter que quand on veut compter le nombre de lignes par groupe, on peut utiliser directement la fonction `count`. Ainsi le code suivant est identique au précédent :

```
R> flights %>%
  count(dest)
```

## Grouper selon plusieurs variables

On peut grouper selon plusieurs variables à la fois, il suffit de les indiquer dans la clause du `group_by` :

```
R> flights %>%
  group_by(month, dest) %>%
  summarise(nb = n()) %>%
  arrange(desc(nb))
```

```
`summarise()` regrouping output by 'month' (override with `groups` argument)
```

On peut également compter selon plusieurs variables :

```
R> flights %>%
  count(origin, dest) %>%
  arrange(desc(n))
```

On peut utiliser plusieurs opérations de groupage dans le même *pipeline*. Ainsi, si on souhaite déterminer le couple origine/destination ayant le plus grand nombre de vols selon le mois de l'année, on devra procéder en deux étapes :

- d'abord grouper selon mois, origine et destination pour calculer le nombre de vols
- puis grouper uniquement selon le mois pour sélectionner la ligne avec la valeur maximale.

Au final, on obtient le code suivant :

```
R> flights %>%
  group_by(month, origin, dest) %>%
  summarise(nb = n()) %>%
  group_by(month) %>%
  filter(nb == max(nb))
```

```
`summarise()` regrouping output by 'month', 'origin' (override with `groups` argument)
```

Lorsqu'on effectue un `group_by` suivi d'un `summarise`, le tableau résultat est automatiquement

dégroupé de la dernière variable de regroupement. Ainsi le tableau généré par le code suivant est groupé par `month` et `origin`:

```
R> flights %>%
  group_by(month, origin, dest) %>%
  summarise(nb = n())
```

```
`summarise()` regrouping output by 'month', 'origin' (override with `groups` argument)
```

Cela peut permettre “d’enchaîner” les opérations groupées. Dans l’exemple suivant on calcule le pourcentage des trajets pour chaque destination par rapport à tous les trajets du mois :

```
R> flights %>%
  group_by(month, dest) %>%
  summarise(nb = n()) %>%
  mutate(pourcentage = nb / sum(nb) * 100)
```

```
`summarise()` regrouping output by 'month' (override with `groups` argument)
```

On peut à tout moment “dégroupier” un tableau à l’aide de `ungroup`. Ce serait par exemple nécessaire, dans l’exemple précédent, si on voulait calculer le pourcentage sur le nombre total de vols plutôt que sur le nombre de vols par mois :

```
R> flights %>%
  group_by(month, dest) %>%
  summarise(nb = n()) %>%
  ungroup() %>%
  mutate(pourcentage = nb / sum(nb) * 100)
```

```
`summarise()` regrouping output by 'month' (override with `groups` argument)
```

À noter que `count`, par contre, renvoie un tableau non groupé :

```
R> flights %>%
  count(month, dest)
```

## Autres fonctions utiles

`dplyr` contient beaucoup d’autres fonctions utiles pour la manipulation de données.

## sample\_n et sample\_frac

`sample_n` et `sample_frac` permettent de sélectionner un nombre de lignes ou une fraction des lignes d'un tableau aléatoirement. Ainsi si on veut choisir 5 lignes au hasard dans le tableau `airports` :

```
R> airports %>% sample_n(5)
```

Si on veut tirer au hasard 10% des lignes de `flights` :

```
R> flights %>% sample_frac(0.1)
```

Ces fonctions sont utiles notamment pour faire de "l'échantillonnage" en tirant au hasard un certain nombre d'observations du tableau.

## lead et lag

`lead` et `lag` permettent de décaler les observations d'une variable d'un cran vers l'arrière (pour `lead`) ou vers l'avant (pour `lag`).

```
R> lead(1:5)
```

```
[1] 2 3 4 5 NA
```

```
R> lag(1:5)
```

```
[1] NA 1 2 3 4
```

Ceci peut être utile pour des données de type "séries temporelles". Par exemple, on peut facilement calculer l'écart entre le retard au départ de chaque vol et celui du vol précédent :

```
R> flights %>%
  mutate(dep_delay_prev = lead(dep_delay),
        dep_delay_diff = dep_delay - dep_delay_prev) %>%
  select(dep_delay_prev, dep_delay, dep_delay_diff)
```

## tally

`tally` est une fonction qui permet de compter le nombre d'observations d'un groupe :

```
R> flights %>%
  group_by(month, origin, dest) %>%
  tally
```

Lors de son premier appel, elle sera équivalente à un `summarise(n = n())` ou à un `count()`. Là où la fonction est intelligente, c'est que si on l'appelle plusieurs fois successivement, elle prendra en compte l'existence d'un `n` déjà calculé et effectuera automatiquement un `summarise(n = sum(n))` :

```
R> flights %>%
  group_by(month, origin, dest) %>%
  tally %>%
  tally
```

## distinct

`distinct` filtre les lignes du tableau pour ne conserver que les lignes distinctes, en supprimant toutes les lignes en double.

```
R> flights %>%
  select(day, month) %>%
  distinct
```

On peut lui spécifier une liste de variables : dans ce cas, pour toutes les observations ayant des valeurs identiques pour les variables en question, `distinct` ne conservera que la première d'entre elles.

```
R> flights %>%
  distinct(month, day)
```

L'option `.keep_all` permet, dans l'opération précédente, de conserver l'ensemble des colonnes du tableau :

```
R> flights %>%
  distinct(month, day, .keep_all = TRUE)
```

# Ressources

Toutes les ressources ci-dessous sont en anglais...

Le livre *R for data science*, librement accessible en ligne, contient plusieurs chapitres très complets sur la manipulation des données, notamment :

- [Data transformation](#) pour les manipulations
- [Relational data](#) pour les tables multiples

Le site de l'extension comprend une [liste des fonctions](#) et les pages d'aide associées, mais aussi une [introduction](#) au package et plusieurs articles dont un spécifiquement sur les [jointures](#).

Une “antisèche” très synthétique est également accessible depuis RStudio, en allant dans le menu *Help* puis *Cheatsheets* et *Data Transformation with dplyr*.

Enfin, on trouvera des exercices dans l'[Introduction à R et au tidyverse](#) de Julien Barnier.

## dplyr et data.table

Pour ceux travaillant également avec l'extension [data.table](#), il est possible de concilier *tibble* et *data.table* avec l'extension [dtplyr](#). Cette extension a connu une profonde évolution en 2019. Pour plus d'informations, voir <https://dtplyr.tidyverse.org/>. Pour découvrir [data.table](#), voir le chapitre dédié, page 251.

## dplyr et survey

L'extension [srvyr](#) vise à permettre d'utiliser les verbes de [dplyr](#) avec les plans d'échantillonnage complexe définis avec [survey](#). Le fonctionnement de cette extension est expliqué dans une vignette dédiée : <https://cran.r-project.org/web/packages/srvyr/vignettes/srvyr-vs-survey.html>.



# Manipulations avancées avec data.table

Convertir un data.frame en data.table . . . . .	251
setDT et setDF . . . . .	252
dplyr et data.table . . . . .	253
La syntaxe des crochets . . . . .	253
Sélectionner des observations . . . . .	253
Sélectionner des variables . . . . .	253
Grouper les résultats . . . . .	254
Ajouter / Modifier / Supprimer une variable . . . . .	255
Enchaîner les opérations . . . . .	255
Réorganiser un tableau . . . . .	256

L'extension **data.table** permet d'étendre les tableaux de données. Elle modifie radicalement la syntaxe des crochets, permettant un code plus court et surtout plus puissant. Par ailleurs, elle est particulièrement rapide pour opérer des opérations sur les données et permet d'effectuer des opérations par assignation directe sans avoir à copier les objets en mémoire. Autrement dit, elle est particulièrement utile lorsque l'on travaille sur des gros fichiers de données.

Certes, l'apprentissage de cette nouvelle syntaxe peut faire peur au début, mais c'est un gain tellement notable une fois qu'on la maîtrise, qu'il est difficile de revenir en arrière.

Pour un tutoriel (en anglais et en ligne) écrit par les développeurs de **data.table**, voir <https://www.datacamp.com/courses/data-table-data-manipulation-r-tutorial>. On pourra aussi se référer au site officiel et ses différentes vignettes (en anglais) : <https://rdatatable.gitlab.io/>.

## Convertir un data.frame en data.table

Il suffit d'avoir recours à la fonction `as.data.table`.

```
R> library(data.table)
iris2 <- as.data.table(iris)
class(iris2)
```

```
[1] "data.table" "data.frame"
```

Comme on le voit, cela ajoute plusieurs classes additionnelles au tableau de données, celui-ci restant malgré tout toujours un *data.frame*. Cependant, la syntaxe des crochets simples `[]` change radicalement, tandis que les crochets doubles `[[]]` restent inchangés. Par contre, comme il s'agit toujours d'un tableau de données classique, on pourra l'utiliser avec les fonctions des autres extensions de R. Si jamais vous rencontriez un problème, il est toujours possible de reconvertir en tableau de données classique avec `setDF` (voir ci-dessous).

## setDT et setDF

Lors de l'utilisation de `as.data.table`, le tableau de données original a d'abord été copié en mémoire, converti puis il a fallu le sauvegarder dans un objet avec `<-`. Lorsqu'on l'on manipule de gros tableaux, cela est gourmand en ressources système et prend du temps.

C'est pour cela que `data.table` fournie plusieurs fonctions (commençant par le préfixe `set`) qui modifient directement l'objet sélectionné en mémoire, ce qu'on appelle «modification par assignation». Ce type de fonction est beaucoup plus rapide et efficace en termes de ressources système. On notera également qu'il est inutile de stocker le résultats dans un objet puisque l'objet a été modifié directement en mémoire.

`setDT` converti un tableau de données en *data.table* tandis que `setDF` fait l'opération opposée.

```
R> setDT(iris)
class(iris)
```

```
[1] "data.table" "data.frame"
```

```
R> setDF(iris)
class(iris)
```

```
[1] "data.frame"
```

## dplyr et data.table

Pour ceux travaillant également avec les extension **dplyr** et **tibble**, il est possible de concilier *tibble* et *data.table* avec l'extension **dtplyr**. Cette extension a connu une profonde évolution en 2019. Pour plus d'informations, voir <https://dtplyr.tidyverse.org/>.

## La syntaxe des crochets

La syntaxe des crochets change radicalement avec **data.table**. Elle est de la forme `objet[i, j, by]` (dans sa forme la plus simple, pour une présentation exhaustive, voir le fichier d'aide de `data.table-package`).

### Sélectionner des observations

Cela se fait en indiquant une condition au premier argument, à savoir `i`. Si l'on ne procède à une sélection en même temps sur les variables, il n'est pas nécessaire d'indiquer de virgule `,` dans les crochets.

```
R> iris2[Sepal.Length < 5]
```

On notera que les noms indiquer entre les crochets sont évalués en fonction du contexte, en l'occurrence la liste des variables de l'objet considéré. Ainsi, les noms des variables peuvent être indiqués tels quels, sans utilisation du symbole `$` ni des guillemets.

#### IMPORTANT

Une différence de taille : lorsqu'il y a des observations pour lesquelles la condition indiquée en `i` renvoie `NA`, elles ne sont pas sélectionnées par **data.table** tandis que, pour un *data.frame* classique cela renvoie des lignes manquantes.

### Sélectionner des variables

Pour sélectionner une variable, il suffit d'indiquer son nom dans la seconde partie, à savoir `j`. Noter la virgule qui permet d'indiquer que c'est une condition sur `j` et non sur `i`.

```
R> iris2[, Sepal.Length]
```

```
[1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8  
[14] 4.3 5.8 5.7 5.4 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0  
[27] 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0 5.5 4.9 4.4  
[40] 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4  
[53] 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6  
[66] 6.7 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7  
[79] 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5 5.5  
[92] 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3  
[105] 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5  
[118] 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2  
[131] 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8  
[144] 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```

Pour sélectionner plusieurs variables, on fournira une liste définie avec `list` (et non un vecteur défini avec `c`).

```
R> iris2[, list(Sepal.Length, Sepal.Width)]
```

`data.table` fournit un raccourci pour écrire une liste : `.()`. A l’intérieur des crochets (mais pas en dehors), `.()` sera compris comme `list()`.

```
R> iris2[, .(Sepal.Length, Sepal.Width)]
```

Il est possible de renommer une variable à la volée et même d’en calculer d’autres.

```
R> iris2[, .(espece = Species, aire_petal = Petal.Length * Petal.Width)]
```

Seul le retour est ici affecté. Cela n’impacte pas le tableau d’origine. Nous verrons plus loin comment créer / modifier une variable.

Attention : on ne peut pas directement sélectionner une variable par sa position ou en indiquant une chaîne de caractères. En effet, une valeur numérique ou textuelle est comprise comme une constante.

```
R> iris2[, .("Species", 3)]
```

## Grouper les résultats

Si en `j` on utilise des fonctions qui à partir d’un vecteur renvoient une valeur unique (telles que `mean`, `median`, `min`, `max`, `first`, `last`, `nth`, etc.), on peut ainsi obtenir un résumé. On pourra également

utiliser `.N` pour obtenir le nombre d'observations.

```
R> iris2[, .(min_sepal_width = min(Sepal.Width), max_sepal_width = max(Sepal.Width), n_observations = .N)]
```

Cela devient particulièrement intéressant en calculant ces mêmes valeurs par sous-groupe, grâce au troisième paramètre : `by`.

```
R> iris2[, .(min_sepal_width = min(Sepal.Width), max_sepal_width = max(Sepal.Width), n_observations = .N), by = Species]
```

## Ajouter / Modifier / Supprimer une variable

`data.table` introduit un nouvel opérateur `:=` permettant de modifier une variable par assignation directe. Cela signifie que la modification a lieu directement en mémoire dans le tableau de données, sans qu'il soit besoin réaffecter le résultat avec `<-`.

On peut également combiner `:=` avec une sélection sur les observations en `i` pour ne modifier que certaines observations. De même, le recours à `by` permet des calculs par groupe.

```
R> iris2[, group := "A"]
  iris2[Species == "virginica", group := "B"]
  iris2[, n_obs_per_species := .N, by = Species]
```

```
R> iris2
```

```
R> iris2[, .N, by = group]
```

## Enchaîner les opérations

Il est possible d'enchaîner les opérations avec une succession de crochets.

```
R> iris2[, .(petal_area = Petal.Width * Petal.Length, Species)][, .(min_petal_area = min(petal_area)), by = Species]
```

## Réorganiser un tableau

L’extension `data.table` fournie également deux fonctions, `melt` et `dcast`, dédiée à la réorganisation d’un tableau de données (respectivement *wide-to-long reshaping* et *long-to-wide reshaping*).

Pour plus de détails, voir la vignette dédiée sur le site de l’extension : <https://rdatatable.gitlab.io/data.table/articles/datatable-reshape.html>

# Tris

Fonctions R de base . . . . .	257
Extension dplyr . . . . .	260
Extension data.table . . . . .	260

Dans ce qui suit on travaillera sur le jeu de données tiré de l'enquête *Histoire de vie*, fourni avec l'extension **questionr**.

```
R> library(questionr)
  data(hdv2003)
  d <- hdv2003
```

## Fonctions R de base

La fonction **sort** permet de trier les éléments d'un vecteur.

```
R> sort(c(2, 5, 6, 1, 8))
```

```
[1] 1 2 5 6 8
```

On peut appliquer cette fonction à une variable, mais celle-ci ne permet que d'ordonner les valeurs de cette variable, et pas l'ensemble du tableau de données dont elle fait partie. Pour cela nous avons besoin d'une autre fonction, nommée **order**. Celle-ci ne renvoie pas les valeurs du vecteur triées, mais les emplacements de ces valeurs.

Un exemple pour comprendre :

```
R> order(c(15, 20, 10))
```

```
[1] 3 1 2
```

Le résultat renvoyé signifie que la plus petite valeur est la valeur située en 3<sup>e</sup> position, suivie de celle en 1<sup>ère</sup> position et de celle en 2<sup>e</sup> position. Tout cela ne paraît pas passionnant à première vue, mais si on mélange ce résultat avec un peu d'indexation directe, ça devient intéressant...

```
R> head(order(d$age))
```

```
[1] 162 215 346 377 511 646
```

Ce que cette fonction renvoie, c'est l'ordre dans lequel on doit placer les éléments de *age*, et donc par extension les lignes de *d*, pour que la variable soit triée par ordre croissant. Par conséquent, si on fait :

```
R> d.tri <- d[order(d$age), ]
```

Alors on a trié les lignes de *d* par ordre d'âge croissant ! Et si on fait un petit :

```
R> head(d.tri, 3)
```

	<b>id</b>	<b>age</b>	<b>sex</b>	<b>nivetud</b>	<b>poids</b>	<b>occup</b>	<b>qualif</b>	<b>freres.soeurs</b>	<b>c1so</b>	<b>reli</b>	
162	162	18	Homme	NA	4982.964	Etudiant, eleve	NA		2	Non	App san
215	215	18	Homme	NA	4631.188	Etudiant, eleve	NA		2	Oui	Nic app
346	346	18	Femme	NA	1725.410	Etudiant, eleve	NA		9	Non	Pra reg

On a les caractéristiques des trois enquêtés les plus jeunes.

On peut évidemment trier par ordre décroissant en utilisant l'option `decreasing=TRUE`. On peut donc afficher les caractéristiques des trois individus les plus âgés avec :

```
R> head(d[order(d$age, decreasing = TRUE), ], 3)
```

	<b>id</b>	<b>age</b>	<b>sexe</b>	<b>nivetud</b>	<b>poids</b>	<b>occup</b>	<b>qualif</b>	<b>freres.soeurs</b>	<b>ciso</b>	<b>retraite</b>
1916	1916	97	Femme	Derniere annee d'etudes primaires	2162.835	Autre inactif	Autre	5	Non	Pension
270	270	96	Femme	Derniere annee d'etudes primaires	9993.020	Retraite	NA	1	Oui	Non
1542	1542	93	Femme	Derniere annee d'etudes primaires	7107.841	Retire des affaires	NA	7	Non	Pension

On peut également trier selon plusieurs variables. Ainsi, si l'on souhaite trier le tableau par *sexe* puis, au sein de chaque sexe, par *age* :

```
R> d.tri <- d[order(d$sexe, d$age), ]
```

#### NOTE

Si l'on transmets une variable textuelle, le tri sera réalisé de manière alphabétique alors que si l'on transmets un facteur, le tri sera effectué selon l'ordre des facteurs (que l'on peut visualiser avec `levels`).

## Extension dplyr

On aura simplement recours à la fonction `arrange`. Un tri par ordre décroissant s’indique avec la fonction `desc`.

```
R> library(dplyr)
tbl <- as_tibble(hdv2003)
tbl <- tbl %>% arrange(sexe, desc(age))
```

## Extension data.table

On pourra utiliser la fonction `order` dans la condition sur les observations (attention à sauvegarder le résultats si nécessaire) ou bien la fonction `setorder` pour modifier l’ordre des observations directement par assignation (modification directe en mémoire de l’objet). Un tri décroissant s’indique avec le signe `-`.

```
R> library(data.table)
dt <- as.data.table(hdv2003)

# Option 1
dt <- dt[order(sexe, -age)]

# Option 2
setorder(dt, sexe, -age)
```

# Sous-ensembles

Par indexation . . . . .	261
Fonction subset . . . . .	263
Fonction tapply . . . . .	264
Extension dplyr . . . . .	266
Extension data.table . . . . .	266

Dans ce qui suit on travaillera sur le jeu de données tiré de l'enquête *Histoire de vie*, fourni avec l'extension [questionr](#).

```
R> library(questionr)
  data(hdv2003)
  d <- hdv2003
```

## Par indexation

La première manière de construire des sous-populations est d'utiliser l'indexation par conditions. On peut ainsi facilement sélectionner une partie des observations suivant un ou plusieurs critères et placer le résultat dans un nouveau tableau de données.

Par exemple si l'on souhaite isoler les hommes et les femmes :

```
R> dh <- d[d$sexe == "Homme", ]
  df <- d[d$sexe == "Femme", ]
  table(d$sexe)
```

Homme	Femme
899	1101

```
R> dim(dh)
```

```
[1] 899 20
```

```
R> dim(df)
```

```
[1] 1101 20
```

On a à partir de là trois tableaux de données, `d` comportant la population totale, `dh` seulement les hommes et `df` seulement les femmes.

On peut évidemment combiner plusieurs critères :

```
R> dh.25 <- d[d$sex == "Homme" & d$age <= 25, ]  
dim(dh.25)
```

```
[1] 86 20
```

Si on utilise directement l'indexation, il convient cependant d'être extrêmement prudent avec les valeurs manquantes. Comme indiqué précédemment, la présence d'une valeur manquante dans une condition fait que celle-ci est évaluée en `NA` et qu'au final la ligne correspondante est conservée par l'indexation :

```
R> summary(d$trav.satisf)
```

Satisfaction	Insatisfaction	Equilibre	NA's
480	117	451	952

```
R> d.satisf <- d[d$trav.satisf == "Satisfaction", ]  
dim(d.satisf)
```

```
[1] 1432 20
```

Comme on le voit, ici `d.satisf` contient les individus ayant la modalité *Satisfaction* mais aussi ceux ayant une valeur manquante `NA`. C'est pourquoi il faut toujours soit vérifier au préalable qu'on n'a pas de valeurs manquantes dans les variables de la condition, soit exclure explicitement les `NA` de la manière suivante :

```
R> d.satisf <- d[d$trav.satisf == "Satisfaction" & !is.na(d$trav.satisf), ]
dim(d.satisf)
```

```
[1] 480 20
```

C'est notamment pour cette raison qu'on préfèrera le plus souvent utiliser la fonction `subset`.

## Fonction subset

La fonction `subset` permet d'extraire des sous-populations de manière plus simple et un peu plus intuitive que l'indexation directe.

Celle-ci prend trois arguments principaux :

- le nom de l'objet de départ ;
- une condition sur les observations ( `subset` ) ;
- éventuellement une condition sur les colonnes ( `select` ).

Reprendons tout de suite un exemple déjà vu :

```
R> dh <- subset(d, sexe == "Homme")
df <- subset(d, sexe == "Femme")
```

L'utilisation de `subset` présente plusieurs avantages. Le premier est d'économiser quelques touches. On n'est en effet pas obligé de saisir le nom du tableau de données dans la condition sur les lignes. Ainsi les deux commandes suivantes sont équivalentes :

```
R> dh <- subset(d, d$sexe == "Homme")
dh <- subset(d, sexe == "Homme")
```

Le second avantage est que `subset` s'occupe du problème des valeurs manquantes évoquées précédemment et les exclut de lui-même, contrairement au comportement par défaut :

```
R> summary(d$trav.satisf)
```

Satisfaction	Insatisfaction	Equilibre	NA's
480	117	451	952

```
R> d.satisf <- d[d$trav.satisf == "Satisfaction", ]  
dim(d.satisf)
```

```
[1] 1432 20
```

```
R> d.satisf <- subset(d, trav.satisf == "Satisfaction")  
dim(d.satisf)
```

```
[1] 480 20
```

Dans le cas présent, l’extraction obtenue avec `subset` est équivalente à :

```
R> d.satisf <- d[d$trav.satisf == "Satisfaction" & !is.na(d$trav.satisf), ]  
dim(d.satisf)
```

```
[1] 480 20
```

Enfin, l’utilisation de l’argument `select` est simplifié pour l’expression de condition sur les colonnes. On peut ainsi spécifier les noms de variable sans guillemets et leur appliquer directement l’opérateur d’exclusion `-` :

```
R> d2 <- subset(d, select = c(sexe, sport))  
d2 <- subset(d, age > 25, select = -c(id, age, cinema))
```

## Fonction `tapply`

### NOTE

Cette section documente une fonction qui peut être très utile, mais pas forcément indispensable au départ.

La fonction `tapply` n’est qu’indirectement liée à la notion de sous-population, mais peut permettre d’éviter d’avoir à créer ces sous-populations dans certains cas.

Son fonctionnement est assez simple, mais pas forcément intuitif. La fonction prend trois arguments : un vecteur, un facteur et une fonction. Elle applique ensuite la fonction aux éléments du vecteur correspondant à un même niveau du facteur. Vite, un exemple !

```
R> tapply(d$age, d$sex, mean)
```

Homme	Femme
48.16129	48.15350

Qu'est-ce que ça signifie ? Ici `tapply` a sélectionné toutes les observations correspondant à « Homme », puis appliqué la fonction `mean` aux valeurs de `age` correspondantes. Puis elle a fait de même pour les observations correspondant à « Femme ». On a donc ici la moyenne d'âge chez les hommes et chez les femmes.

On peut fournir à peu près n'importe quelle fonction à `tapply` :

```
R> tapply(d$bricol, d$sex, freq)
```

\$Homme		
	n	% val%
Non	384	42.7 42.7
Oui	515	57.3 57.3

\$Femme		
	n	% val%
Non	763	69.3 69.3
Oui	338	30.7 30.7

Les arguments supplémentaires fournis à `tapply` sont en fait fournis directement à la fonction appelée.

```
R> tapply(d$bricol, d$sex, freq, total = TRUE)
```

\$Homme		
	n	% val%
Non	384	42.7 42.7
Oui	515	57.3 57.3
Total	899	100.0 100.0

\$Femme		
	n	% val%
Non	763	69.3 69.3
Oui	338	30.7 30.7
Total	1101	100.0 100.0

#### NOTE

La fonction `by` est un équivalent (pour les tableaux de données) de `tapply`. La présentation des résultats diffère légèrement.

```
R> tapply(d$age, d$sex, mean)
```

Homme	Femme
48.16129	48.15350

```
R> by(d$age, d$sex, mean)
```

```
d$sex: Homme  
[1] 48.16129  
-----  
d$sex: Femme  
[1] 48.1535
```

## Extension dplyr

On utilisera tout simplement la fonction `filter`.

```
R> library(dplyr)  
tbl <- as_tibble(hdv2003)  
hommes_jeunes <- tbl %>% filter(sexe == "Homme", age < 30)
```

Voir le chapitre dédié à dplyr, page 233 pour plus de détails.

## Extension data.table

Il suffit d’indiquer la condition entre crochets.

```
R> library(data.table)
dt <- as.data.table(hdv2003)
hommes_jeunes <- dt[sexe == "Hommes" & age < 30]
```

Il est également possible d'utiliser la fonction `subset` sur un **data.table**.

```
R> hommes_jeunes <- subset(dt, sexe == "Hommes" & age < 30)
```

Voir le chapitre dédié à **data.table**, page 251 pour plus de détails.



# Fusion de tables

La fonction merge et les jointures . . . . .	269
Jointures avec dplyr . . . . .	273
Clés implicites . . . . .	274
Clés explicites . . . . .	275
Types de jointures . . . . .	276
Jointures avec data.table . . . . .	282
Ajouter des observations . . . . .	282

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #04 (manipuler les données avec dplyr) sur [YouTube](#).

Lorsqu'on traite de grosses enquêtes, notamment les enquêtes de l'INSEE, on a souvent à gérer des données réparties dans plusieurs tables, soit du fait de la construction du questionnaire, soit du fait de contraintes techniques (fichiers **dbf** ou **Excel** limités à 256 colonnes, par exemple).

Cela arrive également lorsque l'on traite de données d'une enquête réalisée à différents niveaux (par exemple, un questionnaire ménage et un questionnaire individu).

On peut distinguer deux situations :

- l'ajout de variables (jointure entre tables)
- l'ajout d'observations (concaténation de tables)

## La fonction merge et les jointures

Une opération relativement courante consiste à fusionner plusieurs tables pour regrouper tout ou partie des données dans un unique tableau.

Nous allons simuler artificiellement une telle situation en créant deux tables à partir de l'extrait de l'enquête *Histoire de vie* :

```
R> library(questionr)
    data(hdv2003)
    d <- hdv2003
    dim(d)
```

[1] 2000 20

```
R> d1 <- subset(d, select = c("id", "age", "sex"))  
R> dim(d1)
```

```
[1] 2000      3
```

```
R> d2 <- subset(d, select = c("id", "clos"))
    dim(d2)
```

[1] 2000 2

On a donc deux tableaux de données, `d1` et `d2`, comportant chacun 2000 lignes et respectivement 3 et 2 colonnes. Comment les rassembler pour n'en former qu'un ?

Intuitivement, cela paraît simple. Il suffit de « coller »  $d_2$  à la droite de  $d_1$ , comme dans l'exemple suivant.

<b>id</b>	<b>v1</b>	<b>v2</b>	<b>+</b>	<b>id</b>	<b>v3</b>	<b>=</b>	<b>id</b>	<b>v1</b>	<b>v2</b>	<b>v3</b>
1	H	12		1	rouge		1	H	12	rouge
2	H	17		2	bleu		2	H	17	bleu
3	F	41		3	bleu		3	F	41	bleu
4	F	9		4	rouge		4	F	9	rouge
...	...	...		...	...		...	...	...	...

Cela semble fonctionner. La fonction qui permet d'effectuer cette opération sous R s'appelle `cbind`, elle « colle » des tableaux côte à côte en regroupant leurs colonnes.

```
R> head(cbind(d1, d2))
```

À part le fait qu'on a une colonne *id* en double, le résultat semble satisfaisant. À première vue seulement. Imaginons maintenant que nous avons travaillé sur *d1* et *d2*, et que nous avons ordonné les lignes de *d1* selon l'âge des enquêtés :

```
R> d1 <- d1[order(d1$age), ]
```

Répétons l'opération de collage :

```
R> head(cbind(d1, d2))
```

Que constate-t-on ? La présence de la variable *id* en double nous permet de voir que les identifiants ne coïncident plus ! En regroupant nos colonnes nous avons donc attribué à des individus les réponses d'autres individus.

La commande `cbind` ne peut en effet fonctionner que si les deux tableaux ont exactement le même nombre de lignes, et dans le même ordre, ce qui n'est pas le cas ici.

### IMPORTANT

Pour éviter toute erreur, il est préférable de ne jamais utiliser `cbind` ou son équivalent `bind_cols` fournis par `dplyr`.

On aura recours à la jointure entre tables présentée ci-dessous.

On va donc être obligé de procéder à une fusion des deux tableaux, qui va permettre de rendre à chaque ligne ce qui lui appartient. Pour cela nous avons besoin d'un identifiant qui permet d'identifier chaque ligne de manière unique et qui doit être présent dans tous les tableaux. Dans notre cas, c'est plutôt rapide, il s'agit de la variable *id*.

Une fois l'identifiant identifié<sup>1</sup>, on peut utiliser la commande `merge`. Celle-ci va fusionner les deux tableaux en supprimant les colonnes en double et en regroupant les lignes selon leurs identifiants :

```
R> d.complet <- merge(d1, d2, by = "id")
head(d.complet)
```

Ici l'utilisation de la fonction `merge` est plutôt simple car nous sommes dans le cas de figure idéal : les lignes correspondent parfaitement et l'identifiant est clairement identifié. Parfois les choses peuvent être un peu plus compliquées :

- parfois les identifiants n'ont pas le même nom dans les deux tableaux. On peut alors les spécifier par les options `by.x` et `by.y` ;
- parfois les deux tableaux comportent des colonnes (hors identifiants) ayant le même nom. `merge` conserve dans ce cas ces deux colonnes mais les renomme en les suffixant par `.x` pour celles provenant du premier tableau et `.y` pour celles du second ;
- parfois on n'a pas d'identifiant unique préétabli, mais on en construit un à partir de plusieurs

---

1. Si vous me passez l'expression...

variables. On peut alors donner un vecteur en paramètres de l'option `by`, par exemple `by=c("nom","prenom","date.naissance")`.

Une subtilité supplémentaire intervient lorsque les deux tableaux fusionnés n'ont pas exactement les mêmes lignes. Par défaut, `merge` ne conserve que les lignes présentes dans les deux tableaux :

<code>id</code>	<code>v1</code>		<code>id</code>	<code>v2</code>		<code>id</code>	<code>v1</code>	<code>v2</code>
1	H	+	1	10	=	1	H	10
2	H		2	15		2	H	15
3	F		5	31				

On peut cependant modifier ce comportement avec les options `all.x` et `all.y`.

Ainsi, `all.x = TRUE` indique de conserver toutes les lignes du premier tableau. Dans ce cas `merge` donne une valeur `NA` pour ces lignes aux colonnes provenant du second tableau. Ce qui donnerait :

<code>id</code>	<code>v1</code>		<code>id</code>	<code>v2</code>		<code>id</code>	<code>v1</code>	<code>v2</code>
1	H	+	1	10	=	1	H	10
2	H		2	15		2	H	15
3	F		5	31		3	F	NA

L'option `all.y = TRUE` fait la même chose en conservant toutes les lignes du second tableau.

<code>id</code>	<code>v1</code>		<code>id</code>	<code>v2</code>		<code>id</code>	<code>v1</code>	<code>v2</code>
1	H	+	1	10	=	1	H	10
2	H		2	15		2	H	15
3	F		5	31		5	NA	31

Enfin, on peut décider de conserver toutes les lignes des deux tableaux en utilisant à la fois `all.x = TRUE` et `all.y = TRUE`, ce qui donne :

<b>id</b>	<b>v1</b>		<b>id</b>	<b>v2</b>		<b>id</b>	<b>v1</b>	<b>v2</b>
1	H	+	1	10	=	1	H	10
2	H		2	15		2	H	15
3	F		5	31		3	F	NA
						5	NA	31

Parfois, l'un des identifiants est présent à plusieurs reprises dans l'un des tableaux (par exemple lorsque l'une des tables est un ensemble de ménages et que l'autre décrit l'ensemble des individus de ces ménages). Dans ce cas les lignes de l'autre table sont dupliquées autant de fois que nécessaires :

<b>id</b>	<b>v1</b>		<b>id</b>	<b>v2</b>		<b>id</b>	<b>v1</b>	<b>v2</b>
1	H	+	1	10	=	1	H	10
2	H		1	18		1	H	18
3	F		1	21		1	H	21
			2	15		2	H	15
			3	42		3	F	42

## Jointures avec dplyr

Le jeu de données [nycflights13](#) est un exemple de données réparties en plusieurs tables. Ici on en a trois : les informations sur les vols, celles sur les aéroports et celles sur les compagnies aériennes sont dans trois tables distinctes.

[dplyr](#) propose différentes fonctions permettant de travailler avec des données structurées de cette manière.

```
R> library(tidyverse)
  library(nycflights13)
  data(flights)
  data(airports)
  data(airlines)
```

## Clés implicites

Très souvent, les données relatives à une analyse sont réparties dans plusieurs tables différentes. Dans notre exemple, on peut voir que la table `flights` contient seulement le code de la compagnie aérienne du vol dans la variable `carrier`:

```
R> flights %>% select(carrier)
```

Et que par ailleurs la table `airlines` contient une information supplémentaire relative à ces compagnies, à savoir le nom complet.

```
R> airlines
```

Il est donc naturel de vouloir associer les deux, en l’occurrence pour ajouter les noms complets des compagnies à la table `flights`. Dans ce cas on va faire une *jointure*: les lignes d’une table seront associées à une autre en se basant non pas sur leur position, mais sur les valeurs d’une ou plusieurs colonnes. Ces colonnes sont appelées des *clés*.

Pour faire une jointure de ce type, on va utiliser la fonction `left_join` :

```
R> left_join(flights, airlines)
```

Pour faciliter la lecture, on va afficher seulement certaines colonnes du résultat :

```
R> left_join(flights, airlines) %>%
  select(month, day, carrier, name)
```

```
Joining, by = "carrier"
```

On voit que la table résultat est bien la fusion des deux tables d’origine selon les valeurs des deux colonnes clés `carrier`. On est parti de la table `flights`, et pour chaque ligne on a ajouté les colonnes de `airlines` pour lesquelles la valeur de `carrier` est la même. On a donc bien une nouvelle colonne `name` dans notre table résultat, avec le nom complet de la compagnie aérienne.

**NOTE**

À noter qu'on peut tout à fait utiliser le `pipe` avec les fonctions de jointure :

```
flights %>% left_join(airlines).
```

Nous sommes ici dans le cas le plus simple concernant les clés de jointure : les deux clés sont uniques et portent le même nom dans les deux tables. Par défaut, si on ne lui spécifie pas explicitement les clés, `dplyr` fusionne en utilisant l'ensemble des colonnes communes aux deux tables. On peut d'ailleurs voir dans cet exemple qu'un message a été affiché précisant que la jointure s'est faite sur la variable `carrier`.

## Clés explicites

La table `airports`, elle, contient des informations supplémentaires sur les aéroports : nom complet, altitude, position géographique, etc. Chaque aéroport est identifié par un code contenu dans la colonne `faa`.

Si on regarde la table `flights`, on voit que le code d'identification des aéroports apparaît à deux endroits différents : pour l'aéroport de départ dans la colonne `origin`, et pour celui d'arrivée dans la colonne `dest`. On a donc deux clés de jointures possibles, et qui portent un nom différent de la clé de `airports`.

On va commencer par fusionner les données concernant l'aéroport de départ. Pour simplifier l'affichage des résultats, on va se contenter d'un sous-ensemble des deux tables :

```
R> flights_ex <- flights %>% select(month, day, origin, dest)
R> airports_ex <- airports %>% select(faa, alt, name)
```

Si on se contente d'un `left_join` comme à l'étape précédente, on obtient un message d'erreur car aucune colonne commune ne peut être identifiée comme clé de jointure :

```
R> left_join(flights_ex, airports_ex)
```

```
Error: `by` must be supplied when `x` and `y` have no common variables.
i use by = character()` to perform a cross-join.
```

On doit donc spécifier explicitement les clés avec l'argument `by` de `left_join`. Ici la clé est nommée `origin` dans la première table, et `faa` dans la seconde. La syntaxe est donc la suivante :

```
R> left_join(flights_ex, airports_ex, by = c("origin" = "faa"))
```

On constate que les deux nouvelles colonnes `name` et `alt` contiennent bien les données correspondant à l'aéroport de départ.

On va stocker le résultat de cette jointure dans `flights_ex` :

```
R> flights_ex <- flights_ex %>%
  left_join(airports_ex, by = c("origin" = "faa"))
```

Supposons qu'on souhaite maintenant fusionner à nouveau les informations de la table `airports`, mais cette fois pour les aéroports d'arrivée de notre nouvelle table `flights_ex`. Les deux clés sont donc désormais `dest` dans la première table, et `faa` dans la deuxième. La syntaxe est donc la suivante :

```
R> left_join(flights_ex, airports_ex, by = c("dest" = "faa"))
```

Cela fonctionne, les informations de l'aéroport d'arrivée ont bien été ajoutées, mais on constate que les colonnes ont été renommées. En effet, ici les deux tables fusionnées contenaient toutes les deux des colonnes `name` et `alt`. Comme on ne peut pas avoir deux colonnes avec le même nom dans un tableau, `dplyr` a renommé les colonnes de la première table en `name.x` et `alt.x`, et celles de la deuxième en `name.y` et `alt.y`.

C'est pratique, mais pas forcément très parlant. On pourrait renommer manuellement les colonnes pour avoir des intitulés plus explicites avec `rename`, mais on peut aussi utiliser l'argument `suffix` de `left_join`, qui permet d'indiquer les suffixes à ajouter aux colonnes. Ainsi, on peut faire :

```
R> left_join(flights_ex, airports_ex,
  by = c("dest" = "faa"),
  suffix = c("_depart", "_arrivee"))
)
```

On obtient ainsi directement des noms de colonnes nettement plus clairs.

## Types de jointures

Jusqu'à présent nous avons utilisé la fonction `left_join`, mais il existe plusieurs types de jointures.

Partons de deux tables d'exemple, `personnes` et `voitures` :

```
R> personnes <- tibble(
  nom = c("Sylvie", "Sylvie", "Monique", "Gunter", "Rayan", "Rayan"),
  voiture = c("Twingo", "Ferrari", "Scenic", "Lada", "Twingo", "Clio")
)
```

nom	voiture
Sylvie	Twingo
Sylvie	Ferrari
Monique	Scenic
Gunter	Lada
Rayan	Twingo
Rayan	Clio

```
R> voitures <- tibble(
  voiture = c("Twingo", "Ferrari", "Clio", "Lada", "208"),
  vitesse = c("140", "280", "160", "85", "160")
)
```

voiture	vitesse
Twingo	140
Ferrari	280
Clio	160
Lada	85
208	160

## left\_join

Si on fait un `left_join` de `voitures` sur `personnes` :

```
R> left_join(personnes, voitures)
```

Joining, by = "voiture"

nom	voiture	vitesse
Sylvie	Twingo	140
Sylvie	Ferrari	280
Monique	Scenic	NA
Gunter	Lada	85
Rayan	Twingo	140
Rayan	Clio	160

On voit que chaque ligne de `personnes` est bien présente, et qu'on lui a ajouté une ligne de `voitures` correspondante si elle existe. Dans le cas du `Scenic`, il n'y a pas de ligne dans `voitures`, donc `vitesse` a été mise à `NA`. Dans le cas de `208`, présente dans `voitures` mais pas dans `personnes`, la ligne n'apparaît pas.

Si on fait un `left_join` cette fois de `personnes` sur `voitures`, c'est l'inverse :

```
R> left_join(voitures, personnes)
```

Joining, by = "voiture"

voiture	vitesse	nom
Twingo	140	Sylvie
Twingo	140	Rayan
Ferrari	280	Sylvie
Clio	160	Rayan
Lada	85	Gunter
208	160	NA

La ligne `208` est là, mais `nom` est à `NA`. Par contre `Monique` est absente. Et on remarquera que la ligne

`Twingo`, présente deux fois dans `personnes`, a été dupliquée pour être associée aux deux lignes de données de `Sylvie` et `Rayan`.

En résumé, quand on fait un `left_join(x, y)`, toutes les lignes de `x` sont présentes, et dupliquées si nécessaire quand elles apparaissent plusieurs fois dans `y`. Les lignes de `y` non présentes dans `x` disparaissent. Les lignes de `x` non présentes dans `y` se voient attribuer des `NA` pour les nouvelles colonnes.

Intuitivement, on pourrait considérer que `left_join(x, y)` signifie "ramener l'information de la table `y` sur la table `x`".

En général, `left_join` sera le type de jointures le plus fréquemment utilisé.

## right\_join

La jointure `right_join` est l'exacte symétrique de `left_join`, c'est-à-dire que `right_join(x, y)` est équivalent à `left_join(x,y)` :

```
R> right_join(personnes, voitures)
```

```
Joining, by = "voiture"
```

nom	voiture	vitesse
Sylvie	Twingo	140
Sylvie	Ferrari	280
Gunter	Lada	85
Rayan	Twingo	140
Rayan	Clio	160
NA	208	160

## inner\_join

Dans le cas de `inner_join`, seules les lignes présentes à la fois dans `x` et `y` sont présentes (et si nécessaire dupliquées) dans la table résultat :

```
R> inner_join(personnes, voitures)
```

```
Joining, by = "voiture"
```

nom	voiture	vitesse
Sylvie	Twingo	140
Sylvie	Ferrari	280
Gunter	Lada	85
Rayan	Twingo	140
Rayan	Clio	160

Ici la ligne `208` est absente, ainsi que la ligne `Monique`, qui dans le cas d'un `left_join` avait été conservée et s'était vue attribuer une `vitesse` à `NA`.

### full\_join

Dans le cas de `full_join`, toutes les lignes de `x` et toutes les lignes de `y` sont conservées (avec des `NA` ajoutés si nécessaire) même si elles sont absentes de l'autre table :

```
R> full_join(personnes, voitures)
```

```
Joining, by = "voiture"
```

nom	voiture	vitesse
Sylvie	Twingo	140
Sylvie	Ferrari	280
Monique	Scenic	NA
Gunter	Lada	85
Rayan	Twingo	140
Rayan	Clio	160
NA	208	160

## semi\_join et anti\_join

`semi_join` et `anti_join` sont des jointures *filtrantes*, c'est-à-dire qu'elles sélectionnent les lignes de `x` sans ajouter les colonnes de `y`.

Ainsi, `semi_join` ne conservera que les lignes de `x` pour lesquelles une ligne de `y` existe également, et supprimera les autres. Dans notre exemple, la ligne `Monique` est donc supprimée :

```
R> semi_join(personnes, voitures)
```

```
Joining, by = "voiture"
```

nom	voiture
Sylvie	Twingo
Sylvie	Ferrari
Gunter	Lada
Rayan	Twingo
Rayan	Clio

Un `anti_join` fait l'inverse, il ne conserve que les lignes de `x` absentes de `y`. Dans notre exemple, on ne garde donc que la ligne `Monique` :

```
R> anti_join(personnes, voitures)
```

```
Joining, by = "voiture"
```

nom	voiture
Monique	Scenic

## Jointures avec data.table

**data.table** fournit une fonction `merge` beaucoup plus rapide que celle standard de R mais fonctionnant de manière identique.

## Ajouter des observations

### IMPORTANT

La fonction `rbind`, fournie nativement avec R pour ajouter des observations à un tableau, doit être évitée car elle générera des résultats non pertinents si les tableaux que l’on concatènent n’ont pas exactement les mêmes colonnes dans le même ordre.

La fonction `bind_rows` de **dplyr** permet d’ajouter des lignes à une table à partir d’une ou plusieurs autres tables.

L’exemple suivant (certes très artificiel) montre l’utilisation de `bind_rows`. On commence par créer trois tableaux `t1`, `t2` et `t3` :

```
R> t1 <- airports %>%
  select(faa, name, lat, lon) %>%
  slice(1:2)
t1
```

```
R> t2 <- airports %>%
  select(name, faa, lon, lat) %>%
  slice(5:6)

t2
```

```
R> t3 <- airports %>%
  select(faa, name) %>%
  slice(100:101)
t3
```

On concaténe ensuite les trois tables avec `bind_rows` :

```
R> bind_rows(t1, t2, t3)
```

On remarquera que si des colonnes sont manquantes pour certaines tables, comme les colonnes `lat` et `lon` de `t3`, des `NA` sont automatiquement insérées.

De plus, peu importe l'ordre des variables entre les différentes tables, `bind_rows` les réassociera en considérant que deux colonnes ayant le même nom dans deux tableaux correspondent à la même variable.

Il peut être utile, quand on concatène des lignes, de garder une trace du tableau d'origine de chacune des lignes dans le tableau final. C'est possible grâce à l'argument `.id` de `bind_rows`. On passe à cet argument le nom d'une colonne qui contiendra l'indicateur d'origine des lignes :

```
R> bind_rows(t1, t2, t3, .id = "source")
```

Par défaut la colonne `.id` ne contient qu'un nombre, différent pour chaque tableau. On peut lui spécifier des valeurs plus explicites en "nommant" les tables dans `bind_rows` de la manière suivante :

```
R> bind_rows(table1 = t1, table2 = t2, table3 = t3, .id = "source")
```

#### NOTE

Une alternative à `bind_rows` est la fonction `rbind.fill` de l'extension `plyr` qui fonctionne de manière similaire.



# Gestion des dates

Si R fournit quelques fonctions natives pour la gestion des dates, l'extension **lubridate** est recommandée pour tout travail un peu plus fin sur des dates. On pourra se référer :

- au chapitre «Dates and Times» de l'ouvrage *R for Data Science* de Garrett Grolemund et Hadley Wickham (en anglais)
- à la vignette officielle (<https://cran.r-project.org/web/packages/lubridate/vignettes/lubridate.html>, en anglais)
- à ce tutoriel (<https://rpubs.com/davoodastaraky/lubridate>)



# Fonctions à fenêtre

## IMPORTANT

Ce chapitre est en cours d'écriture.

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #04 (manipuler les données avec dplyr) sur [YouTube](#).

Ces fonctions sont présentées en détail (en anglais) dans une vignette dédiée de l'extension `dplyr` (<https://dplyr.tidyverse.org/articles/window-functions.html>).



# Manipuler du texte avec `stringr`

Expressions régulières . . . . .	290
Concaténer des chaînes . . . . .	291
Convertir en majuscules / minuscules . . . . .	292
Découper des chaînes . . . . .	293
Extraire des sous-chaînes par position . . . . .	294
DéTECTer des motifs . . . . .	294
Extraire des motifs . . . . .	295
Remplacer des motifs . . . . .	296
Modificateurs de motifs . . . . .	297
Insérer une variable dans une chaîne de caractères . . . . .	298
Ressources . . . . .	298

## NOTE

La version originale de ce chapitre a été écrite par Julien Barnier dans le cadre de son [Introduction à R et au tidyverse](#).

Les fonctions de `forcats` vues précédemment permettent de modifier des modalités d'une variables qualitative globalement. Mais parfois on a besoin de manipuler le contenu même du texte d'une variable de type chaîne de caractères : combiner, rechercher, remplacer...

On va utiliser ici les fonctions de l'extension `stringr`. Celle-ci fait partie du cœur du `tidyverse`, elle est donc automatiquement chargée avec :

```
R> library(tidyverse)
```

#### NOTE

`stringr` est en fait une interface simplifiée aux fonctions d'une autre extension, `stringi`. Si les fonctions de `stringr` ne sont pas suffisantes ou si on manipule beaucoup de chaînes de caractères, ne pas hésiter à se reporter à la documentation de `stringi`.

Dans ce qui suit on va utiliser le court tableau d'exemple `d` suivant :

```
R> d <- tibble(
  nom = c("Mr Félicien Machin", "Mme Raymonde Bidule", "M. Martial Truc", "Mme Huguette Chose"),
  adresse = c("3 rue des Fleurs", "47 ave de la Libération", "12 rue du 17 octobre 1961", "221 avenue de la Libération"),
  ville = c("Nouméa", "Marseille", "Vénissieux", "Marseille")
)
```

nom	adresse	ville
Mr Félicien Machin	3 rue des Fleurs	Nouméa
Mme Raymonde Bidule	47 ave de la Libération	Marseille
M. Martial Truc	12 rue du 17 octobre 1961	Vénissieux
Mme Huguette Chose	221 avenue de la Libération	Marseille

## Expressions régulières

Les fonctions présentées ci-dessous sont pour la plupart prévues pour fonctionner avec des expressions régulières. Celles-ci constituent un mini-langage, qui peut paraître assez cryptique, mais qui est très puissant pour spécifier des motifs de chaînes de caractères.

Elles permettent par exemple de sélectionner le dernier mot avant la fin d'une chaîne, l'ensemble des suites alphanumériques commençant par une majuscule, des nombres de 3 ou 4 chiffres situés en début de chaîne, et beaucoup beaucoup d'autres choses encore bien plus complexes.

Pour donner un exemple concret, l'expression régulière suivante permet de détecter une adresse de courrier électronique<sup>1</sup> :

```
[\w\d+.-_]+@[\\w\d.-]+\.\[a-zA-Z]{2,}
```

Par souci de simplicité, dans ce qui suit les exemples seront donnés autant que possible avec de simples chaînes, sans expression régulière. Mais si vous pensez manipuler des données textuelles, il peut être très utile de s'intéresser à cette syntaxe.

## Concaténer des chaînes

La première opération de base consiste à concaténer des chaînes de caractères entre elles. On peut le faire avec la fonction `paste`.

Par exemple, si on veut concaténer l'adresse et la ville :

```
R> paste(d$adresse, d$ville)
```

```
[1] "3 rue des Fleurs Nouméa"  
[2] "47 ave de la Libération Marseille"  
[3] "12 rue du 17 octobre 1961 Vénissieux"  
[4] "221 avenue de la Libération Marseille"
```

Par défaut, `paste` concatène en ajoutant un espace entre les différentes chaînes. On peut spécifier un autre séparateur avec son argument `sep` :

```
R> paste(d$adresse, d$ville, sep = " - ")
```

```
[1] "3 rue des Fleurs - Nouméa"  
[2] "47 ave de la Libération - Marseille"  
[3] "12 rue du 17 octobre 1961 - Vénissieux"  
[4] "221 avenue de la Libération - Marseille"
```

Il existe une variante, `paste0`, qui concatène sans mettre de séparateur, et qui est légèrement plus rapide :

```
R> paste0(d$adresse, d$ville)
```

```
[1] "3 rue des FleursNouméa"  
[2] "47 ave de la LibérationMarseille"  
[3] "12 rue du 17 octobre 1961Vénissieux"
```

---

1. Il s'agit en fait d'une version très simplifiée, la «véritable» expression permettant de tester si une adresse mail est valide fait plus de 80 lignes...

```
[4] "221 avenue de la LibérationMarseille"
```

#### NOTE

À noter que `paste` et `paste0` sont des fonctions R de base. L'équivalent pour `stringr` se nomme `str_c`.

Parfois on cherche à concaténer les différents éléments d'un vecteur non pas avec ceux d'un autre vecteur, comme on l'a fait précédemment, mais *entre eux*. Dans ce cas `paste` seule ne fera rien :

```
R> paste(d$ville)
```

```
[1] "Nouméa"      "Marseille"   "Vénissieux"  "Marseille"
```

Il faut lui ajouter un argument `collapse`, avec comme valeur la chaîne à utiliser pour concaténer les éléments :

```
R> paste(d$ville, collapse = ", ")
```

```
[1] "Nouméa, Marseille, Vénissieux, Marseille"
```

## Convertir en majuscules / minuscules

Les fonctions `str_to_lower`, `str_to_upper` et `str_to_title` permettent respectivement de mettre en minuscules, mettre en majuscules, ou de capitaliser les éléments d'un vecteur de chaînes de caractères :

```
R> str_to_lower(d$nom)
```

```
[1] "mr félicien machin"  "mme raymonde bidule"  
[3] "m. martial truc"    "mme huguette chose"
```

```
R> str_to_upper(d$nom)
```

```
[1] "MR FÉLICIEN MACHIN"  "MME RAYMONDE BIDULE"
```

```
[3] "M. MARTIAL TRUC"      "MME HUGUETTE CHOSE"
```

```
R> str_to_title(d$nom)
```

```
[1] "Mr Félicien Machin"  "Mme Raymonde Bidule"
[3] "M. Martial Truc"     "Mme Huguette Chose"
```

## Découper des chaînes

La fonction `str_split` permet de “découper” une chaîne de caractère en fonction d'un délimiteur. On passe la chaîne en premier argument, et le délimiteur en second :

```
R> str_split("un-deux-trois", "-")
```

```
[[1]]
[1] "un"    "deux"   "trois"
```

On peut appliquer la fonction à un vecteur, dans ce cas le résultat sera une liste :

```
R> str_split(d$nom, " ")
```

```
[[1]]
[1] "Mr"       "Félicien" "Machin"

[[2]]
[1] "Mme"      "Raymonde" "Bidule"

[[3]]
[1] "M."       "Martial"  "Truc"

[[4]]
[1] "Mme"      "Huguette" "Chose"
```

Ou un tableau (plus précisément une matrice) si on ajoute `simplify = TRUE`.

```
R> str_split(d$nom, " ", simplify = TRUE)
```

```
[,1] [,2]      [,3]
```

```
[1,] "Mr"  "Félicien" "Machin"  
[2,] "Mme"  "Raymonde" "Bidule"  
[3,] "M."   "Martial"  "Truc"  
[4,] "Mme"  "Huguette" "Chose"
```

Si on souhaite créer de nouvelles colonnes dans un tableau de données en découplant une colonne de type texte, on pourra utiliser la fonction `separate` de l’extension `tidyR`. Celle-ci est expliquée section @ref(separate).

Voici juste un exemple de son utilisation :

```
R> library(tidyR)  
d %>% separate(nom, c("genre", "prenom", "nom"))
```

## Extraire des sous-chaînes par position

La fonction `str_sub` permet d’extraire des sous-chaînes par position, en indiquant simplement les positions des premier et dernier caractères :

```
R> str_sub(d$ville, 1, 3)
```

```
[1] "Nou" "Mar" "Vén" "Mar"
```

## DéTECTER DES MOTIFS

`str_detect` permet de détecter la présence d’un motif parmi les éléments d’un vecteur. Par exemple, si on souhaite identifier toutes les adresses contenant «Libération» :

```
R> str_detect(d$adresse, "Libération")
```

```
[1] FALSE  TRUE FALSE  TRUE
```

`str_detect` renvoie un vecteur de valeurs logiques et peut donc être utilisée, par exemple, avec le verbe `filter` de `dplyr` pour extraire des sous-populations.

Une variante, `str_count`, compte le nombre d’occurrences d’une chaîne pour chaque élément d’un vecteur :

```
R> str_count(d$ville, "s")
```

```
[1] 0 1 2 1
```

### IMPORTANT

Attention, les fonctions de `stringr` étant prévues pour fonctionner avec des expressions régulières, certains caractères n'auront pas le sens habituel dans la chaîne indiquant le motif à rechercher. Par exemple, le `.` ne sera pas un point mais le symbole représentant «n'importe quel caractère».

La section sur les modificateurs de motifs explique comment utiliser des chaînes «classiques» au lieu d'expressions régulières.

On peut aussi utiliser `str_subset` pour ne garder d'un vecteur que les éléments correspondant au motif :

```
R> str_subset(d$adresse, "Libération")
```

```
[1] "47 ave de la Libération"
[2] "221 avenue de la Libération"
```

## Extraire des motifs

`str_extract` permet d'extraire les valeurs correspondant à un motif. Si on lui passe comme motif une chaîne de caractère, cela aura peu d'intérêt :

```
R> str_extract(d$adresse, "Libération")
```

```
[1] NA          "Libération" NA          "Libération"
```

C'est tout de suite plus intéressant si on utilise des expressions régulières. Par exemple la commande suivante permet d'isoler les numéros de rue.

```
R> str_extract(d$adresse, "^\d+")
```

```
[1] "3"    "47"   "12"   "221"
```

`str_extract` ne récupère que la première occurrence du motif. Si on veut toutes les extraire on peut utiliser `str_extract_all`. Ainsi, si on veut extraire l’ensemble des nombres présents dans les adresses :

```
R> str_extract_all(d$adresse, "\\d+")
```

```
[[1]]
[1] "3"

[[2]]
[1] "47"

[[3]]
[1] "12"    "17"    "1961"

[[4]]
[1] "221"
```

#### NOTE

Si on veut faire de l’extraction de groupes dans des expressions régulières (identifiés avec des parenthèses), on pourra utiliser `str_match`.

À noter que si on souhaite extraire des valeurs d’une colonne texte d’un tableau de données pour créer de nouvelles variables, on pourra utiliser la fonction `extract` de l’extension `tidyR`, décrite plus haut.

Par exemple :

```
R> library(tidyR)
d %>% extract(adresse, "type_rue", "^\d+ (.*) ", remove = FALSE)
```

## Remplacer des motifs

La fonction `str_replace` permet de remplacer une chaîne ou un motif par une autre.

Par exemple, on peut remplacer les occurrences de “Mr” par “M.” dans les noms de notre tableau :

```
R> str_replace(d$nom, "Mr", "M.")
```

```
[1] "M. Félicien Machin"  "Mme Raymonde Bidule"
```

```
[3] "M. Martial Truc"      "Mme Huguette Chose"
```

La variante `str_replace_all` permet de spécifier plusieurs remplacements d'un coup :

```
R> str_replace_all(d$adresse, c("avenue" = "Avenue", "ave" = "Avenue", "rue" = "Rue"))
```

```
[1] "3 Rue des Fleurs"
[2] "47 Avenue de la Libération"
[3] "12 Rue du 17 octobre 1961"
[4] "221 Avenue de la Libération"
```

## Modificateurs de motifs

Par défaut, les motifs passés aux fonctions comme `str_detect`, `str_extract` ou `str_replace` sont des expressions régulières classiques.

On peut spécifier qu'un motif n'est pas une expression régulière mais une chaîne de caractères normale en lui appliquant la fonction `fixed`. Par exemple, si on veut compter le nombre de points dans les noms de notre tableau, le paramétrage par défaut ne fonctionnera pas car dans une expression régulière le `.` est un symbole signifiant “n'importe quel caractère” :

```
R> str_count(d$nom, ".")
```

```
[1] 18 19 15 18
```

Il faut donc spécifier que notre point est bien un point avec `fixed` :

```
R> str_count(d$nom, fixed("."))
```

```
[1] 0 0 1 0
```

On peut aussi modifier le comportement des expressions régulières à l'aide de la fonction `regex`. On peut ainsi rendre les motifs insensibles à la casse avec `ignore_case` :

```
R> str_detect(d$nom, "mme")
```

```
[1] FALSE FALSE FALSE FALSE
```

```
R> str_detect(d$nom, regex("mme", ignore_case = TRUE))
```

```
[1] FALSE TRUE FALSE TRUE
```

On peut également permettre aux regex d'être multilignes avec l'option `multiline = TRUE`, etc.

## Insérer une variable dans une chaîne de caractères

La fonction `str_glue` repose sur l'extension `glue`. Elle permet, à l'aide d'une syntaxe un peu spécifique, de pouvoir insérer facilement les valeurs d'une ou plusieurs variables dans une chaîne de caractères. Prenons un exemple :

```
R> prenom <- "Fred"
age <- 28
anniversaire <- as.Date("1991-10-12")
str_glue(
  "Je m'appelle {prenom}. ",
  "L'année prochaine j'aurai {age + 1} ans, ",
  "car je suis né le {format(anniversaire, '%A %d %B %Y')}."
)
```

```
Je m'appelle Fred. L'année prochaine j'aurai 29 ans, car je suis né le samedi 1
2 octobre 1991.
```

Sa variante `str_glue_data` est adaptée lorsque l'on travaille sur un tableau de données avec `dplyr`.

```
R> d %>% mutate(phrase = str_glue_data(d, "{nom} habite à {ville}."))
```

## Ressources

L'ouvrage *R for Data Science*, accessible en ligne, contient [un chapitre entier](#) sur les chaînes de caractères et les expressions régulières (en anglais).

Le site officiel de `stringr` contient une [liste des fonctions](#) et les pages d'aide associées, ainsi qu'un [article dédié aux expressions régulières](#).

Pour des besoins plus pointus, on pourra aussi utiliser [l'extension stringi](#) sur laquelle est elle-même basée `stringr`.

# Réorganiser ses données avec `tidyverse`

Tidy data . . . . .	299
Trois règles pour des données bien rangées . . . . .	302
Les verbes de <code>tidyverse</code> . . . . .	305
pivot_longer : rassembler des colonnes . . . . .	305
pivot_wider : disperser des lignes . . . . .	307
separate : séparer une colonne en plusieurs . . . . .	310
unite : regrouper plusieurs colonnes en une seule . . . . .	311
extract : créer de nouvelles colonnes à partir d'une colonne de texte . . . . .	312
complete : compléter des combinaisons de variables manquantes . . . . .	313
Ressources . . . . .	314
Fichiers volumineux . . . . .	315

## NOTE

La version originale de ce chapitre a été écrite par Julien Barnier dans le cadre de son [Introduction à R et au tidyverse](#).

## Tidy data

Comme indiqué dans l'introduction au `tidyverse`, page 60, les extensions du `tidyverse` comme `dplyr` ou `ggplot2` partent du principe que les données sont “bien rangées” sous forme de *tidy data*.

Prenons un exemple avec les données suivantes, qui indique la population de trois pays pour quatre années différentes :

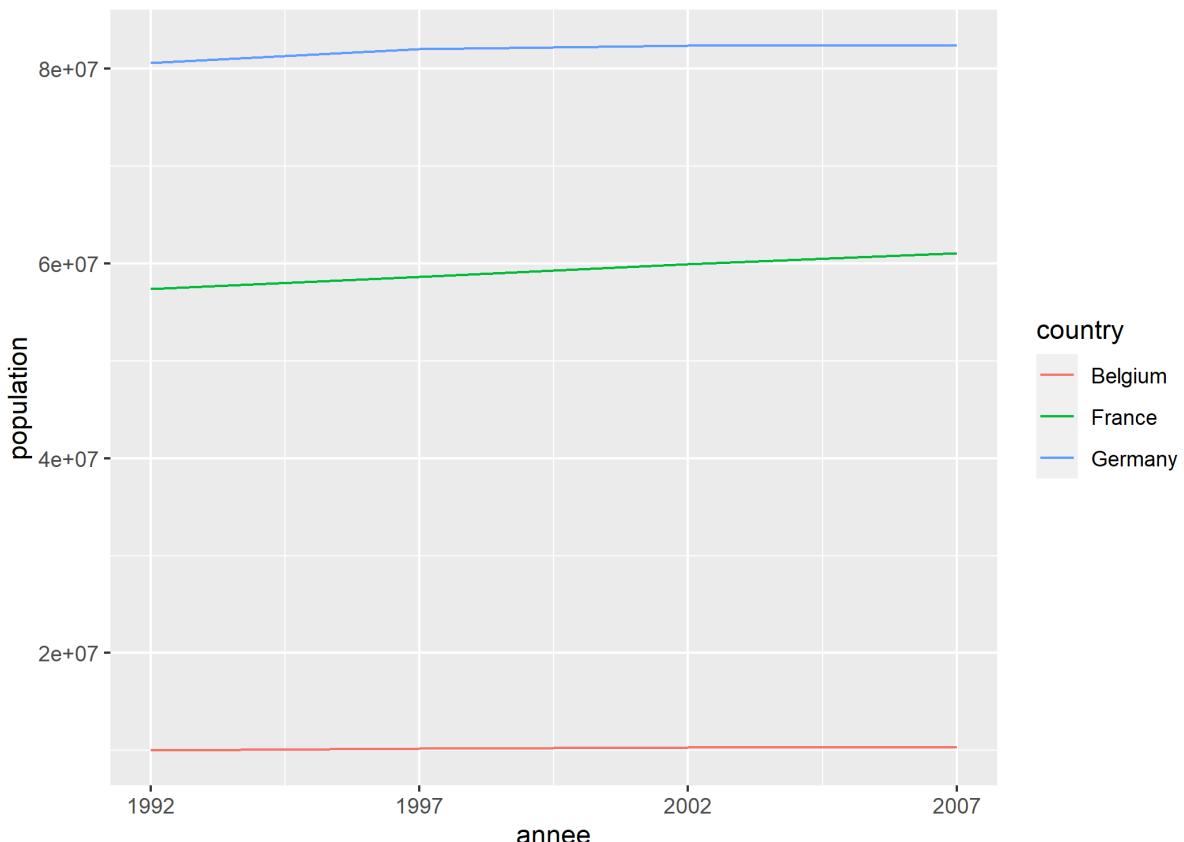
country	1992	1997	2002	2007
Belgium	10045622	10199787	10311970	10392226
France	57374179	58623428	59925035	61083916
Germany	80597764	82011073	82350671	82400996

Imaginons qu'on souhaite représenter avec `ggplot2` l'évolution de la population pour chaque pays sous forme de lignes : c'est impossible avec les données sous ce format. On a besoin d'arranger le tableau de la manière suivante :

country	annee	population
Belgium	1992	10045622
France	1992	57374179
Germany	1992	80597764
Belgium	1997	10199787
France	1997	58623428
Germany	1997	82011073
Belgium	2002	10311970
France	2002	59925035
Germany	2002	82350671
Belgium	2007	10392226
France	2007	61083916
Germany	2007	82400996

C'est seulement avec les données dans ce format qu'on peut réaliser le graphique :

```
R> ggplot(d) +
  geom_line(aes(x = annee, y = population, color = country)) +
  scale_x_continuous(breaks = unique(d$annee))
```



C'est la même chose pour `dplyr`, par exemple si on voulait calculer la population minimale pour chaque pays avec `summarise` :

```
R> d %>%
  group_by(country) %>%
  summarise(pop_min = min(population))
```

```
`summarise()` ungrouping output (override with `.` argument)
```

```
# A tibble: 3 x 2
  country pop_min
  <fct>    <int>
1 Belgium 10045622
2 France  57374179
3 Germany 80597764
```

## Trois règles pour des données bien rangées

Le concept de *tidy data* repose sur trois règles interdépendantes. Des données sont considérées comme *tidy* si :

1. chaque ligne correspond à une observation
2. chaque colonne correspond à une variable
3. chaque valeur est présente dans une unique case de la table ou, de manière équivalente, si des unités d’observations différentes sont présentes dans des tables différentes

Ces règles ne sont pas forcément très intuitives. De plus, il y a une infinité de manières pour un tableau de données de ne pas être *tidy*.

Prenons par exemple les règles 1 et 2 et le tableau de notre premier exemple :

country	1992	1997	2002	2007
Belgium	10045622	10199787	10311970	10392226
France	57374179	58623428	59925035	61083916
Germany	80597764	82011073	82350671	82400996

Pourquoi ce tableau n’est pas *tidy*? Parce que si on essaie d’identifier les variables mesurées dans le tableau, il y en a trois : le pays, l’année et la population. Or elles ne correspondent pas aux colonnes de la table. C'est le cas par contre pour la table transformée :

country	annee	population
Belgium	1992	10045622
France	1992	57374179
Germany	1992	80597764
Belgium	1997	10199787
France	1997	58623428
Germany	1997	82011073
Belgium	2002	10311970
France	2002	59925035
Germany	2002	82350671
Belgium	2007	10392226
France	2007	61083916
Germany	2007	82400996

On peut remarquer qu'en modifiant notre table pour satisfaire à la deuxième règle, on a aussi réglé la première : chaque ligne correspond désormais à une observation, en l'occurrence l'observation de trois pays à plusieurs moments dans le temps. Dans notre table d'origine, chaque ligne comportait en réalité quatre observations différentes.

Ce point permet d'illustrer le fait que les règles sont interdépendantes.

Autre exemple, généré depuis le jeu de données [nycflights13](#), permettant cette fois d'illustrer la troisième règle :

year	month	day	dep_time	carrier	name	flights_per_year
2013	1	1	517	UA	United Air Lines Inc.	58665
2013	1	1	533	UA	United Air Lines Inc.	58665
2013	1	1	542	AA	American Airlines Inc.	32729
2013	1	1	554	UA	United Air Lines Inc.	58665
2013	1	1	558	AA	American Airlines Inc.	32729
2013	1	1	558	UA	United Air Lines Inc.	58665
2013	1	1	558	UA	United Air Lines Inc.	58665
2013	1	1	559	AA	American Airlines Inc.	32729

Dans ce tableau on a bien une observation par ligne (un vol), et une variable par colonne. Mais on a une “infraction” à la troisième règle, qui est que chaque valeur doit être présente dans une unique case : si on regarde la colonne `name`, on a en effet une duplication de l’information concernant le nom des compagnies aériennes. Notre tableau mêle en fait deux types d’observations différents : des observations sur les vols, et des observations sur les compagnies aériennes.

Pour “arranger” ce tableau, il faut séparer les deux types d’observations en deux tables différentes :

year	month	day	dep_time	carrier
2013	1	1	517	UA
2013	1	1	533	UA
2013	1	1	542	AA
2013	1	1	554	UA
2013	1	1	558	AA
2013	1	1	558	UA
2013	1	1	558	UA
2013	1	1	559	AA

carrier	name	flights_per_year
UA	United Air Lines Inc.	58665
AA	American Airlines Inc.	32729

On a désormais deux tables distinctes, l'information n'est pas dupliquée, et on peut facilement faire une jointure si on a besoin de récupérer l'information d'une table dans une autre.

## Les verbes de `tidyR`

L'objectif de `tidyR` est de fournir des fonctions pour arranger ses données et les convertir dans un format `tidy`. Ces fonctions prennent la forme de verbes qui viennent compléter ceux de `dplyr` et s'intègrent parfaitement dans les séries de `pipes` (`%>%`), les *pipelines*, permettant d'enchaîner les opérations.

### `pivot_longer` : rassembler des colonnes

Prenons le tableau `d` suivant, qui liste la population de 6 pays en 2002 et 2007 :

country	2002	2007
Belgium	10311970	10392226
France	59925035	61083916
Germany	82350671	82400996
Italy	57926999	58147733
Spain	40152517	40448191
Switzerland	7361757	7554661

Dans ce tableau, une même variable (la population) est répartie sur plusieurs colonnes, chacune représentant une observation à un moment différent. On souhaite que la variable ne représente plus qu'une seule colonne, et que les observations soient réparties sur plusieurs lignes.

Pour cela on va utiliser la fonction `pivot_longer`<sup>1</sup>:

---

1. `pivot_longer` est une version plus récente et plus robuste de la fonction `gather`.

```
R> d %>% pivot_longer(cols = c(`2002`, `2007`), names_to = "annee", values_to = "population")
```

```
# A tibble: 12 x 3
  country     annee population
  <fct>      <chr>    <int>
  1 Belgium   2002     10311970
  2 Belgium   2007     10392226
  3 France    2002     59925035
  4 France    2007     61083916
  5 Germany   2002     82350671
  6 Germany   2007     82400996
  7 Italy     2002     57926999
  8 Italy     2007     58147733
  9 Spain     2002     40152517
 10 Spain    2007     40448191
 11 Switzerland 2002    7361757
 12 Switzerland 2007    7554661
```

La fonction `pivot_longer` prend comme arguments la liste des colonnes à rassembler (on peut également y utiliser les différentes fonctions de sélection de variables utilisables avec `select`, voir `select_helpers`), ainsi que deux arguments `names_to` et `values_to` :

- `names_to` est le nom de la colonne qui va contenir les “noms” des colonnes originelles, c'est-à-dire les identifiants des différentes observations
- `values_to` est le nom de la colonne qui va contenir la valeur des observations

Parfois il est plus rapide d’indiquer à `pivot_longer` les colonnes qu’on ne souhaite pas rassembler. On peut le faire avec la syntaxe suivante :

```
R> d %>% pivot_longer(-country, names_to = "annee", values_to = "population")
```

```
# A tibble: 12 x 3
  country     annee population
  <fct>      <chr>    <int>
  1 Belgium   2002     10311970
  2 Belgium   2007     10392226
  3 France    2002     59925035
  4 France    2007     61083916
  5 Germany   2002     82350671
  6 Germany   2007     82400996
  7 Italy     2002     57926999
  8 Italy     2007     58147733
  9 Spain     2002     40152517
```

10	Spain	2007	40448191
11	Switzerland	2002	7361757
12	Switzerland	2007	7554661

## pivot\_wider : disperser des lignes

La fonction `pivot_wider`<sup>2</sup> est l'inverse de `pivot_longer`.

Soit le tableau `d` suivant :

country	continent	year	variable	value
Belgium	Europe	2002	lifeExp	78.320
Belgium	Europe	2007	lifeExp	79.441
France	Europe	2002	lifeExp	79.590
France	Europe	2007	lifeExp	80.657
Germany	Europe	2002	lifeExp	78.670
Germany	Europe	2007	lifeExp	79.406
Belgium	Europe	2002	pop	10311970.000
Belgium	Europe	2007	pop	10392226.000
France	Europe	2002	pop	59925035.000
France	Europe	2007	pop	61083916.000
Germany	Europe	2002	pop	82350671.000
Germany	Europe	2007	pop	82400996.000

Ce tableau a le problème inverse du précédent : on a deux variables, `lifeExp` et `pop` qui, plutôt que d'être réparties en deux colonnes, sont réparties entre plusieurs lignes.

On va donc utiliser `pivot_wider` pour «disperser» ces lignes dans deux colonnes différentes :

---

2. `pivot_wider` est une version plus récente et plus robuste de la fonction `spread`.

```
R> d %>% pivot_wider(names_from = variable, values_from = value)
```

```
# A tibble: 6 x 5
  country continent year lifeExp      pop
  <fct>    <fct>   <int>    <dbl>    <dbl>
1 Belgium Europe     2002     78.3 10311970
2 Belgium Europe     2007     79.4 10392226
3 France  Europe     2002     79.6 59925035
4 France  Europe     2007     80.7 61083916
5 Germany Europe     2002     78.7 82350671
6 Germany Europe     2007     79.4 82400996
```

`pivot_wider` prend deux arguments principaux :

- `names_from` indique la colonne contenant les noms des nouvelles variables à créer
- `values_from` indique la colonne contenant les valeurs de ces variables

Il peut arriver que certaines variables soient absentes pour certaines observations. Dans ce cas l’argument `values_fill` permet de spécifier la valeur à utiliser pour ces données manquantes (par défaut, les valeurs manquantes sont indiquées avec `NA`).

Exemple avec le tableau `d` suivant :

country	continent	year	variable	value
Belgium	Europe	2002	lifeExp	78.320
Belgium	Europe	2007	lifeExp	79.441
France	Europe	2002	lifeExp	79.590
France	Europe	2007	lifeExp	80.657
Germany	Europe	2002	lifeExp	78.670
Germany	Europe	2007	lifeExp	79.406
Belgium	Europe	2002	pop	10311970.000
Belgium	Europe	2007	pop	10392226.000
France	Europe	2002	pop	59925035.000
France	Europe	2007	pop	61083916.000
Germany	Europe	2002	pop	82350671.000
Germany	Europe	2007	pop	82400996.000
France	Europe	2002	density	94.000

```
R> d %>%
  pivot_wider(names_from = variable, values_from = value)
```

```
# A tibble: 6 x 6
  country continent year lifeExp      pop density
  <chr>   <chr>    <dbl>   <dbl>     <dbl>   <dbl>
1 Belgium Europe     2002    78.3 10311970       NA
2 Belgium Europe     2007    79.4 10392226       NA
3 France  Europe     2002    79.6 59925035      94
4 France  Europe     2007    80.7 61083916       NA
5 Germany Europe     2002    78.7 82350671       NA
6 Germany Europe     2007    79.4 82400996       NA
```

```
R> d %>%
  pivot_wider(names_from = variable, values_from = value, values_fill = list(value = 0))
```

```
# A tibble: 6 x 6
  country continent year lifeExp      pop density
  <chr>    <chr>    <dbl>    <dbl>    <dbl>    <dbl>
1 Belgium Europe     2002    78.3 10311970      0
2 Belgium Europe     2007    79.4 10392226      0
3 France  Europe     2002    79.6 59925035     94
4 France  Europe     2007    80.7 61083916      0
5 Germany Europe     2002    78.7 82350671      0
6 Germany Europe     2007    79.4 82400996      0
```

## separate : séparer une colonne en plusieurs

Parfois on a plusieurs informations réunies en une seule colonne et on souhaite les séparer. Soit le tableau d’exemple caricatural suivant, nommé `df` :

eleve	note
Félicien Machin	5/20
Raymonde Bidule	6/10
Martial Truc	87/100

`separate` permet de séparer la colonne `note` en deux nouvelles colonnes `note` et `note_sur` :

```
R> d %>% separate(note, c("note", "note_sur"))
```

```
# A tibble: 3 x 3
  eleve          note note_sur
  <chr>        <chr>   <chr>
1 Félicien Machin 5      20
2 Raymonde Bidule 6      10
3 Martial Truc   87     100
```

`separate` prend deux arguments principaux, le nom de la colonne à séparer et un vecteur indiquant les noms des nouvelles variables à créer. Par défaut `separate` «sépare» au niveau des caractères non-alphanumérique (espace, symbole, etc.). On peut lui indiquer explicitement le caractère sur lequel séparer avec l’argument `sep` :

```
R> d %>% tidyverse::separate(eleve, c("prenom", "nom"), sep = " ")
```

```
# A tibble: 3 x 3
  prenom   nom    note
  <chr>    <chr>   <chr>
1 Félicien Machin 5/20
2 Raymonde Bidule 6/10
3 Martial Truc    87/100
```

## unite : regrouper plusieurs colonnes en une seule

`unite` est l'opération inverse de `separate`. Elle permet de regrouper plusieurs colonnes en une seule. Imaginons qu'on obtient le tableau `d` suivant :

code_departement	code_commune	commune	pop_tot
01	004	Ambérieu-en-Bugey	14233
01	007	Ambronay	2437
01	014	Arbent	3440
01	024	Attignat	3110
01	025	Bâgé-la-Ville	3130
01	027	Balan	2785

On souhaite reconstruire une colonne `code_insee` qui indique le code INSEE de la commune, et qui s'obtient en concaténant le code du département et celui de la commune. On peut utiliser `unite` pour cela :

```
R> d %>% unite(code_insee, code_departement, code_commune)
```

```
# A tibble: 6 x 3
  code_insee  commune          pop_tot
  <chr>       <chr>           <int>
1 01_004     Ambérieu-en-Bugey 14233
2 01_007     Ambronay         2437
3 01_014     Arbent           3440
4 01_024     Attignat         3110
5 01_025     Bâgé-la-Ville   3130
```

6 01_027	Balan	2785
----------	-------	------

Le résultat n'est pas idéal : par défaut `unite` ajoute un caractère `_` entre les deux valeurs concaténées, alors qu'on ne veut aucun séparateur. De plus, on souhaite conserver nos deux colonnes d'origine, qui peuvent nous être utiles. On peut résoudre ces deux problèmes à l'aide des arguments `sep` et `remove` :

```
R> d %>%
  unite(code_insee, code_departement, code_commune,
    sep = "", remove = FALSE
  )
```

	# A tibble: 6 x 5				
	code_insee	code_departement	code_commune	commune	pop_tot
1	01004	01	004	Ambérieu~	14233
2	01007	01	007	Ambronay	2437
3	01014	01	014	Arbent	3440
4	01024	01	024	Attignat	3110
5	01025	01	025	Bâgé-la-~	3130
6	01027	01	027	Balan	2785

## extract : créer de nouvelles colonnes à partir d'une colonne de texte

`extract` permet de créer de nouvelles colonnes à partir de sous-chaînes d'une colonne de texte existante, identifiées par des groupes dans une expression régulière.

Par exemple, à partir du tableau suivant :

eleve	note
Félicien Machin	5/20
Raymonde Bidule	6/10
Martial Truc	87/100

On peut extraire les noms et prénoms dans deux nouvelles colonnes avec :

```
R> df %>% extract(
  eleve,
  c("prenom", "nom"),
  "^(.*) (.*)$"
)
```

On passe donc à `extract` trois arguments : la colonne d'où on doit extraire les valeurs, un vecteur avec les noms des nouvelles colonnes à créer, et une expression régulière comportant autant de groupes (identifiés par des parenthèses) que de nouvelles colonnes.

Par défaut la colonne d'origine n'est pas conservée dans la table résultat. On peut modifier ce comportement avec l'argument `remove = FALSE`. Ainsi, le code suivant extrait les initiales du prénom et du nom mais conserve la colonne d'origine :

```
R> df %>% extract(eleve,
  c("initiale_prenom", "initiale_nom"),
  "^(.).*(.).*$",
  remove = FALSE
)
```

## complete : compléter des combinaisons de variables manquantes

Imaginons qu'on ait le tableau de résultats suivants :

eleve	matiere	note
Alain	Maths	16
Alain	Français	9
Barnabé	Maths	17
Chantal	Français	11

Les élèves Barnabé et Chantal n'ont pas de notes dans toutes les matières. Supposons que c'est parce qu'ils étaient absents et que leur note est en fait un 0. Si on veut calculer les moyennes des élèves, on doit compléter ces notes manquantes.

La fonction `complete` est prévue pour ce cas de figure : elle permet de compléter des combinaisons manquantes de valeurs de plusieurs colonnes.

On peut l'utiliser de cette manière :

```
R> df %>% complete(eleve, matiere)
```

On voit que les combinaisons manquante “Barnabé - Français” et “Chantal - Maths” ont bien été ajoutées par `complete`.

Par défaut les lignes insérées récupèrent des valeurs manquantes `NA` pour les colonnes restantes. On peut néanmoins choisir une autre valeur avec l'argument `fill`, qui prend la forme d'une liste nommée :

```
R> df %>% complete(eleve, matiere, fill = list(note = 0))
```

Parfois on ne souhaite pas inclure toutes les colonnes dans le calcul des combinaisons de valeurs. Par exemple, supposons qu'on rajoute dans notre tableau une colonne avec les identifiants de chaque élève :

<b>id</b>	<b>eleve</b>	<b>matiere</b>	<b>note</b>
1001001	Alain	Maths	16
1001001	Alain	Français	9
1001002	Barnabé	Maths	17
1001003	Chantal	Français	11

Si on applique `complete` comme précédemment, le résultat n'est pas bon car il contient toutes les combinaisons de `id`, `eleve` et `matiere`.

```
R> df %>% complete(id, eleve, matiere)
```

Dans ce cas, pour signifier à `complete` que `id` et `eleve` sont deux attributs d'un même individu et ne doivent pas être combinés entre eux, on doit les placer dans une fonction `nesting` :

```
R> df %>% complete(nesting(id, eleve), matiere)
```

## Ressources

Chaque jeu de données est différent, et le travail de remise en forme est souvent long et plus ou moins compliqué. On n'a donné ici que les exemples les plus simples, et c'est souvent en combinant différentes opérations qu'on finit par obtenir le résultat souhaité.

Le livre *R for data science*, librement accessible en ligne, contient [un chapitre complet](#) sur la remise en forme des données.

L'article [Tidy data](#), publié en 2014 dans le *Journal of Statistical Software*, présente de manière détaillée le concept éponyme (mais il utilise des extensions désormais obsolètes qui ont depuis été remplacées par `dplyr` et `tidyR`).

Le site de l'extension est accessible à l'adresse : <http://tidyR.tidyverse.org/> et contient une liste des fonctions et les pages d'aide associées.

En particulier, on pourra se référer à la vignette dédiée à `pivot_wider` et `pivot_longer` pour des exemples avancés de réorganisation des données.

Pour des usages avancés, il est possible avec `tidyR` de gérer des données nichées (*nested data*), c'est-à-dire des tableaux de données dans des tableaux de données. Ces fonctionnalités, réservées aux utilisateurs avancés, sont décrites dans une vignette spécifique.

## Fichiers volumineux

Si l'on a des tableaux de données particulièrement volumineux (plusieurs Go), les fonctions de `tidyR` ne sont pas forcément les plus performantes.

On aura alors intérêt à regarder du côté des fonctions `melt` et `dcast` de l'extension `data.table` développées pour optimiser la performance sur les grands tableaux de données.

Pour plus de détails, voir la vignette dédiée : <https://rdatatable.gitlab.io/data.table/articles/datatable-reshape.html>



# Scraping

Les sources de l'exemple . . . . .	317
Les blogs . . . . .	318
Les mots-clés . . . . .	319
Récupération des données . . . . .	319
Récupération d'éléments HTML . . . . .	320
Récupération de plusieurs pages . . . . .	322
Utilisation de la syntaxe XPath . . . . .	323
Combinaison des résultats . . . . .	324

Une grande partie des données que l'on trouve sur Internet n'y sont pas présentées sous la forme d'un jeu de données : dans de très nombreux cas de figure, ces données peuvent être présentées, par exemple, sous la forme d'un tableau, ou d'une série de pages Web. Ce chapitre explique comment récupérer ces données, de manière à en permettre la manipulation dans R.

La récupération de données numériques, que l'on va illustrer à partir de trois sites Internet consacrés aux théories du complot circulant en France, est plus connue sous le nom de *scraping* ou de *Web scraping*. Il s'agit d'un ensemble de techniques, dont on présentera ici que les principaux aspects, appliqués à un cas d'étude précis.

## Les sources de l'exemple

Ce chapitre s'intéresse à trois sites Internet consacrés aux théories du complot et à leurs diffuseurs, les « conspirationnistes ». Le site de Rudy Reichstadt, [Conspiracy Watch](#), qui va devenir notre principale source de données, propose une [définition de ce terme](#). La seconde source utilisée, le site [Confusionnisme](#) d'Ornella Guyet, utilise une [définition différente](#), qui recoupe largement la première du point de vue des individus et des groupes qu'elle identifie. Notre troisième source, le site anonyme [Conspis hors de nos vi\[ll\]es](#), ne propose pas de définition précise pour sa part, mais fournit [quelques éléments supplémentaires de description](#).

Les termes de « théorie du complot » et de « conspirationnisme » étant difficiles à saisir en seulement quelques phrases, on renverra le lecteur à la [note publiée par Rudy Reichstadt](#) pour l'Observatoire des

radicalités politiques de la Fondation Jean Jaurès. Cette note donne un bon aperçu des différents groupes impliqués dans la diffusion de ces « théories » en France, que l’on retrouve dans une [cartographie en réseau](#) de leurs sites Internet, réalisée par Joël Gombin en juillet 2014. Les données récupérées dans ce chapitre recoupent les informations fournies dans ces deux sources.

## Les blogs

Les sites Internet auxquels on s’intéresse sont tous les trois publiés sous la forme de blogs. Ce détail est important, car pour en récupérer les informations publiées par ces sites, il va falloir comprendre la structure sous-jacente de ces blogs, c’est-à-dire la syntaxe HTML de leurs pages. Les sites [Confusionnisme](#) et [Conspis hors de nos vi\[ll\]es](#) sont les plus simples à comprendre. En effet, ils sont tous les deux publiés grâce au moteur de blog [WordPress](#), qui permet de parcourir les différentes pages d’un blog en rajoutant le suffixe `/page/n` à l’adresse-racine du site, de la manière suivante :

```
http://confusionnisme.info/page/1  
http://confusionnisme.info/page/2  
http://confusionnisme.info/page/3  
...  
http://conspishorsdenosvies.noblogs.org/page/1  
http://conspishorsdenosvies.noblogs.org/page/2  
http://conspishorsdenosvies.noblogs.org/page/3  
...
```

En navigant ces liens, on s’aperçoit que les deux sites en question n’ont publié qu’un nombre limité de billets : il n’y a que 4 pages de billets sur le premier, et 5 pages sur le second. Le site [Conspiracy Watch](#) est, en comparaison, beaucoup plus riche : en effet, comme l’indique le compteur visible en bas de chaque page, le site compte 60 pages de billets, auxquelles le lecteur peut accéder en utilisant un suffixe différent, lié à l’utilisation d’un moteur de blog différent de WordPress. Dans ce cas de figure, le suffixe ne renvoie pas à une « page », mais à un « compteur » de billets, où le dernier billet publié est numéroté `0` :

```
http://www.conspiracywatch.info/?start=0  
http://www.conspiracywatch.info/?start=20  
http://www.conspiracywatch.info/?start=40  
...
```

Suivant ce schéma de pagination, qui commence à `0` puis augmente de 20 billets par page, la page `60` va correspondre au suffixe `?start=1180`. On connaît donc désormais le nombres de pages à récupérer sur chacun des blogs étudiés, en notant bien que c’est le site [Conspiracy Watch](#) qui va fournir la très grande majorité des pages. On aurait pu « découvrir » ces informations de manière programmatique, en écrivant un peu de code pour ce faire, mais un repérage manuel du nombre de pages sur chacun des blogs est ici tout aussi rapide, même s’il faudra le mettre à jour lorsque les blogs auront publié de nouvelles pages de billets.

## Les mots-clés

Sur chacun des blogs auxquels on s'intéresse, on trouve des billets très détaillés sur tel ou tel groupe diffusant une ou plusieurs « théories du complot ». Sur les blogs [Confusionnisme](#) et [Conspiracy Watch](#), on trouve par exemple [deux articles](#) sur un groupuscule ayant appelé à un « Mouvement du 14 juillet » 2015. Sur le blog [Conspis hors de nos vi\[ll\]es](#), qui a cessé de publier en mars 2012, le [dernier billet](#) évoque un autre exemple de ces groupes. Ces différents billets sont tous soigneusement catégorisés par de très nombreux mots-clés, qui incluent notamment les noms propres des individus cités ; [ce billet](#), par exemple, se termine par les mots-clés suivants :

```
11 septembre, antiaméricanisme, apollo 11, etat islamique, etats-unis, laurent louis, lune
```

Ces mots-clés sont destinés à permettre aux lecteurs de naviguer plus facilement à travers les différents billets du site, ainsi qu'à faciliter l'indexation du blog par les moteurs de recherche. Ce que l'on se propose de faire ici consiste à récupérer, pour chacun des billets publiés par chacun des trois blogs, l'ensemble de ces mots-clés, ainsi que les titres, les dates de publication et les adresses Internet – les [URL](#) – des billets auxquels ils correspondent. Ces données permettront par la suite de construire un [réseau de co-occurrences](#) de ces mots-clés, c'est-à-dire une représentation graphique des associations entre ces mots-clés sur la base des trois sources utilisées.

## Récupération des données

Pour récupérer les données des trois blogs, on va commencer par charger quelques extensions utilisées dans plusieurs autres chapitres : l'extension [dplyr](#) va servir à manipuler les données au fur et à mesure de leur récupération ; l'extension [readr](#) va servir à sauvegarder le résultat final au format CSV ; l'extension [lubridate](#) va servir à convertir les dates de publication des billets vers un même format générique ; et l'extension [stringr](#) va servir à nettoyer le texte récupéré.

```
R> library(dplyr)
library(readr)
library(lubridate)
library(stringr)
```

Chargeons à présent l'extension [rvest](#), qui va fournir les fonctions essentielles à la récupération des données de chacun des blogs. Comme [l'explique](#) l'auteur de l'extension, celle-ci est inspirée d'extensions équivalentes disponibles pour le langage Python. Sa fonctionnalité principale est de permettre à l'utilisateur, à l'aide d'une syntaxe simplifiée ou à l'aide de la syntaxe [XPath](#), de sélectionner les différents éléments d'une page Web, à partir des balises [HTML](#) et [CSS](#) de cette page.<sup>1</sup>

```
R> library(rvest)
```

## Récupération d’éléments HTML

Commençons par le blog [Confusionnisme](#). Un rapide coup d’œil au [code source de sa page d’accueil](#) montre que les billets publiés sur ce blog se trouvent dans une suite de structures : l’une d’entre elles, `<div id="scraping_content">`, qui se lit « diviseur à identifiant `content` », contient tous les billets, et à l’intérieur de cette structure, tous les titres de billets se trouvent dans un lien `<a>` à l’intérieur d’une balise `<h1 class="entry-title">`, qui se lit « titre de niveau 1 de classe « `entry-title` ».

Récupérons désormais le code source de la page d’accueil du blog grâce à la fonction `html`. Une fois exécuté le code ci-dessous, affichez le contenu de l’objet `h` pour réaliser que vous venez de récupérer le code source HTML de la page d’accueil du blog :

```
R> h <- html("http://confusionnisme.info/")
```

Selectionnons, à présent, toutes les balises correspondant aux identifiants notés ci-dessus, grâce à la fonction `html_nodes`. Pour gagner de la place, on n’affichera ici que les deux premiers titres de billets que renvoie cette dernière fonction :

```
R> html_nodes(h, "#content .entry-title a") %>%
  head(2)
```

Le code ci-dessus signifie : « sélectionner tous les hyperliens `<a>`, à l’intérieur des éléments identifiés par la classe `entry-title`, à l’intérieur de l’élément portant l’identifiant `content` ». Comme l’on peut le voir, les identifiants des éléments HTML (`id`), qui sont censés être uniques, sont codés par un dièse (`#`), et les classes de ces mêmes éléments (`class`), qui peuvent se répéter, sont codées par un point (`.`). Ces codes sont identiques à ceux que l’on utilise pour attribuer des styles particuliers à ces éléments en langage CSS.

Les éléments HTML que l’on a sélectionnés contiennent aussi bien des balises HTML (telles que `<a>` et `<i>`) que du texte. Pour ne sélectionner que le texte, on rajoute la fonction `html_text` au code montré ci-dessus. Toujours par économie de place, on ne montre que les deux premiers résultats de ce nouvel enchaînement de fonctions :

---

1. Si vous ne connaissez rien aux langages HTML et CSS, c’est le moment ou jamais d’en apprendre les bases ! Un excellent site de référence pour ce faire est [W3 Schools](#).

```
R> html_nodes(h, "#content .entry-title a") %>%
  html_text %>%
  head(2)
```

Voilà qui permet donc de récupérer les titres des billets ! Pour récupérer les hyperliens vers ces billets, rien de plus simple : au lieu de récupérer le texte des titres, il suffit de demander à récupérer l'attribut `href` de chaque lien, en utilisant la fonction `html_attr`. On obtient cette fois-ci les hyperliens complets vers chaque billet :

```
R> html_nodes(h, "#content .entry-title a") %>%
  html_attr("href") %>%
  head(2)
```

Présentons encore un exemple de sélection d'éléments sur la page d'accueil de ce blog, cette fois-ci en montrant l'intégralité des éléments récupérés, car ils prennent peu de place à l'écran. Ici, on récupère les dates de publications des billets, qui se trouvent, toujours selon le [code source de la page](#), dans une balise `<time>` qui se trouve dans une balise `<header class="entry-meta">`. Le code que l'on donne à la fonction `html_nodes` est donc :

```
R> html_nodes(h, "#content header.entry-header time") %>%
  html_text
```

On voit bien ici que les deux premières dates sont identiques aux dates qui figurent dans les hyperliens des deux premiers billets, tels que vus plus haut.

```
R> html_nodes(h, "#content header.entry-header time") %>%
  html_text
```

Terminons, enfin, par un exemple plus compliqué. Comme on l'a déjà écrit, chacun des billets du blog est accompagné de plusieurs mots-clés. Après inspection du code source, on voit que ces mots-clés se trouvent regroupés dans un élément appelé `<span class="tag-links">`. Visionnons les deux premiers éléments en question, toujours à l'aide de la même syntaxe de sélection :

```
R> html_nodes(h, ".tag-links") %>%
  head(2)
```

Pour pouvoir stocker tous les mots-clés d'un billet sur la même ligne d'un fichier CSV, qui contiendra aussi le titre du billet, son hyperlien et sa date de publication, il va falloir regrouper ces mots-clés. On va donc, à l'intérieur de chacun des éléments de la liste d'éléments `<span>`, extraire le texte des mots-clés, contenus dans les éléments `<a>`, et les “coller” ensemble grâce à la fonction `paste0` et à son argument `collapse` :

```
R> html_nodes(h, ".tag-links") %>%
  sapply(function(x) html_nodes(x, "a")) %>%
  html_text %>%
  paste0(collapse = ";"))
head(2)
```

L'astuce se trouve ici dans l'utilisation de la fonction `sapply`, qui permet de travailler sur chacun des éléments `<span class="tag-links">` de manière séparée. L'utilisation de la fonction `pipe %>%` a par ailleurs permis de travailler de manière cumulative, par essai-erreur, tout en produisant un code final plutôt lisible.

## Récupération de plusieurs pages

On sait désormais comment récupérer les informations que l'on veut collecter. Le blog [Confusionnisme](#) n'ayant que 4 pages, il va être très simple de les récupérer à l'aide d'une petite boucle qui récupère chaque page, en extrait les données inspectées ci-dessus, et les rajoute à un jeu de données initialement vide, nommé `d1`, grâce à la fonction `rbind` :

```
R> d1 = data_frame()

for(i in 1:4) {

  h = html(paste0("http://confusionnisme.info/page/", i))

  d1 = rbind(d1, data_frame(
    url = html_nodes(h, "#content .entry-title a") %>% html_attr("href"),
    title = html_nodes(h, "#content .entry-title a") %>% html_text,
    date = html_nodes(h, "#content header.entry-header time") %>% html_text,
    tags = html_nodes(h, ".tag-links") %>%
      sapply(function(x) html_nodes(x, "a")) %>%
      html_text %>%
      paste0(collapse = ";"))
  ))
}
```

À la date de publication de ce blog, ce petit bout de code récupère les 36 billets étalés sur les 4 pages du site [Confusionnisme](#). Comme le montre l'inspection du résultat, le jeu de données que l'on vient de constituer contient l'adresse, le titre, la date de publication et les mots-clés de ces billets :

```
R> View(d1)
```

Il ne reste plus qu'à convertir la variable `date` vers le format générique `yyyy-mm-dd` que propose R

à travers la fonction `as.Date`. Pour convertir la variable, on utilise l'extension `lubridate`, qui peut facilement interpréter les mois écrits en langue française grâce à l'argument `locale` spécifié ci-dessous :

```
R> d1$date = parse_date_time(d1$date, "%d %m %Y", locale = "fr_FR") %>%  
  as.Date
```

## Utilisation de la syntaxe XPath

L'exemple que l'on vient de voir permet de récupérer les données du blog [Confusionnisme](#). Il se trouve que ce code fonctionne presque aussi bien pour le blog [Conspis hors de nos vies](#) : en effet, celui-ci utilisant aussi le moteur de blog WordPress, la structure de ses pages est quasiment identique à celle que l'on vient de voir. Voici le code complet pour récupérer les 5 pages de ce blog :

```
R> d2 = data_frame()  
  
for(i in 1:5) {  
  
  h = html(paste0("http://conspishorsdenosvies.noblogs.org/page/", i))  
  
  d2 = rbind(d2, data_frame(  
    url = html_nodes(h, "#content .entry-title a") %>% html_attr("href"),  
    title = html_nodes(h, "#content .entry-title a") %>% html_text,  
    date = html_nodes(h, "#content .entry-date") %>% html_text,  
    tags = html_nodes(h, ".tag-links") %>%  
      sapply(function(x) html_nodes(x, xpath = "a[@rel='tag']")) %>%  
        html_text %>%  
        paste0(collapse = ";"))  
  ))  
  
}  
  
d2$date = parse_date_time(d2$date, "%d/%m/%Y") %>% as.Date
```

On remarquera que plusieurs petites choses ont changé : par exemple, sur le blog [Conspis hors de nos vies](#), les dates sont affichées dans un format `dd/mm/yyyy` qui ne nécessite pas de conversion, car chaque élément de la date est donné sous la forme d'un chiffre. On remarquera aussi que l'emplacement de la date a changé, car le gabarit graphique du blog diffère de celui de [Confusionnisme](#) et place cette information dans un élément différent du [code source de la page](#).

Le changement le plus important ici concerne l'utilisation de la syntaxe **XPath** : en effet, pour récupérer les mots-clés, il nous a fallu limiter ceux-ci à ceux se trouvant dans des hyperliens (`<a>`) dont la propriété `rel` est égale à `tag`, pour ne pas également récupérer les mots-clés correspondant à des catégories du blog. La syntaxe XPath est un peu plus alambiquée : ici, c'est l'expression `a[@rel='tag']` qui accomplit l'opération souhaitée, à condition d'être bien passée à l'argument `xpath` de la fonction `html_nodes`.

## Combinaison des résultats

Il nous reste un blog à couvrir : [Conspiracy Watch](#). Le code pour celui-ci diffère assez fondamentalement des blogs précédents du point de vue de la syntaxe de ses pages, qui utilisent un moteur de blog complètement différent de WordPress. Après lecture de la source, on arrive au code suivant, qui récupère les mêmes variables que récupérées pour les deux autres blogs :

```
R> d3 = data_frame()

for(i in seq(0, 1180, 20)) {

  h = html(paste0("http://www.conspiracywatch.info/?start=", i))

  d3 = rbind(d3, data_frame(
    url = html_nodes(h, "#mod_1260437 .titre a") %>% html_attr("href"),
    title = html_nodes(h, "#mod_1260437 .titre") %>% html_text %>% str_trim,
    date = html_nodes(h, "#mod_1260437 .cel_pied .date") %>% html_text,
    tags = html_nodes(h, "#mod_1260437 .cel_pied") %>%
      sapply(function(x) html_nodes(x, ".objet-tag a")) %>%
      html_text %>%
      paste0(collapse = ";"))
  ))

}

d3$url = paste0("http://www.conspiracywatch.info", d3$url)
d3$date = parse_date_time(d3$date, "%d %m %Y", locale = "fr_FR") %>% as.Date
```

Il ne reste plus qu’à combiner les différents résultats de nos récupérations, de les ordonner par date de publication, puis d’harmoniser les mots-clés à minima, en supprimant les traits d’union et en s’assurant qu’ils ne contiennent pas de lettres majuscules :

```
R> d <- rbind(d1, d2, d3) %>% arrange(date)
d$tags <- gsub("-", " ", d$tags) %>% tolower()
```

L’inspection du résultat montre que l’on dispose à présent d’un jeu de données contenant les métadonnées de 1,268 billets de blogs, dont l’immense majorité proviennent de [Conspiracy Watch](#) :

```
R> # nombre de billets récupérés  
nrow(d)  
# sources des billets  
table(substr(d$url, 1, 25))
```

Il ne reste plus qu'à sauvegarder ce résultat, pour réutilisation future :

```
R> write_csv(d, "data/conspi.csv")
```



# Export de données

Export de tableaux de données .....	327
Exporter des objets spatiaux .....	328
Sauvegarder des objets .....	328

## Export de tableaux de données

On peut avoir besoin d'exporter un tableau de données dans R vers un fichier dans différents formats. La plupart des fonctions d'import disposent d'un équivalent permettant l'export de données. On citera notamment :

- `write_csv`, `write_delim`, `write_tsv` (`readr`) permettent d'enregistrer un *data frame* ou un *tibble* dans un fichier au format texte délimité
- `write_sas` (`haven`) permet d'exporter au format **SAS**
- `write_sav` (`haven`) permet d'exporter au format **SPSS**
- `write_dta` (`haven`) permet d'exporter au format **Stata**

L'extension `readxl` ne fournit pas de fonction pour exporter au format **Excel**. Par contre, on pourra passer par la fonction `write.xlsx` de l'extension `openxlsx` ou la fonction `write.xlsx` de l'extension `xlsx`. L'intérêt de l'extension `openxlsx` est de ne pas dépendre de **Java** à la différence de l'extension `xlsx`.

Pour le format **dBase**, on peut utiliser `write.dbf` (`foreign`).

Ces fonctions sont utiles si on souhaite diffuser des données à quelqu'un d'autre, ou entre deux logiciels.

Si vous travaillez sur des données de grandes dimensions, les formats texte peuvent être lents à exporter et importer. Dans ce cas, l'extension `feather` peut être utile : elle permet d'enregistrer un *data frame* au format feather, qui n'est pas le plus compact mais qui est extrêmement rapide à lire et écrire<sup>1</sup>.

Les fonctions `read_feather` et `write_feather` permettent d'importer et exporter des tableaux de données dans ce format.

---

1. `feather` est un format compatible avec **Python**, **R** et **Julia**. Pour plus d'informations voir <https://github.com/wesm/feather>

## Exporter des objets spatiaux

On aura recours à l’extension `maptools` qui fournit les fonctions `writePointsShape` | `maptools`, `writeLinesShape` et `writePolyShape` pour exporter des données respectivement de type points, lignes et polygones au format **Shapefile**, et la fonction `writeAsciiGrid` pour exporter un objet raster au format **ASCII grid**.

## Sauvegarder des objets

Une autre manière de sauvegarder des données est de les enregistrer au format `RData`. Ce format propre à **R** est compact, rapide, et permet d’enregistrer plusieurs objets **R**, quel que soit leur type, dans un même fichier.

Pour enregistrer des objets, il suffit d’utiliser la fonction `save` et de lui fournir la liste des objets à sauvegarder et le nom du fichier :

```
R> save(d, rp2012, tab, file = "fichier.RData")
```

Pour charger des objets préalablement enregistrés, utiliser `load` :

```
R> load("fichier.RData")
```

Les objets `d`, `rp2012` et `tab` devraient alors apparaître dans votre environnement.

### IMPORTANT

Attention, quand on utilise `load`, les objets chargés sont importés directement dans l’environnement en cours avec leur nom d’origine. Si d’autres objets du même nom existaient déjà, ils sont écrasés sans avertissement.

**R** propose différentes fonctions permettant d’exporter des données vers des formats variés.

Type de fichier souhaité	Fonction	Extension
texte	<code>write.table</code>	<code>utils</code>
CSV	<code>write.csv</code>	<code>utils</code>
CSV	<code>write_csv</code>	<code>readr</code>
Excel	<code>write.xlsx</code>	<code>xlsx</code>
dBase	<code>write.dbf</code>	<code>foreign</code>
SPSS	<code>write_sav</code>	<code>haven</code>
SPSS	<code>write.foreign</code>	<code>foreign</code>
Stata	<code>write.dta</code>	<code>foreign</code>
Stata	<code>write_dta</code>	<code>haven</code>
SAS	<code>write.foreign</code>	<code>foreign</code>
SPSS	<code>write.foreign</code>	<code>foreign</code>

À nouveau, pour plus de détails on se référera aux pages d'aide de ces fonctions et au manuel *R Data Import/Export* accessible à l'adresse suivante : <http://cran.r-project.org/manuals.html>.



# Export de graphiques

Via l'interface de RStudio . . . . .	331
Sauvegarder le fichier en tant qu'image . . . . .	332
Sauvegarder le graphique en PDF . . . . .	334
Copier le graphique dans le presse-papier . . . . .	335
Export avec les commandes de R . . . . .	336
Export avec ggplot2 . . . . .	337

## Via l'interface de RStudio

L'export de graphiques est très facile avec **RStudio**. Lorsque l'on créé un graphique, ce dernier est affiché sous l'onglet *Plots* dans le quadrant inférieur droit. Il suffit de cliquer sur *Export* pour avoir accès à trois options différentes :

- *Save as image* pour sauvegarder le graphique en tant que fichier image ;
- *Save as PDF* pour sauvegarder le graphique dans un fichier **PDF** ;
- *Copy to Clipboard* pour copier le graphique dans le presse-papier (et pouvoir ainsi le coller ensuite dans un document **Word** par exemple).

## Sauvegarder le fichier en tant qu’image

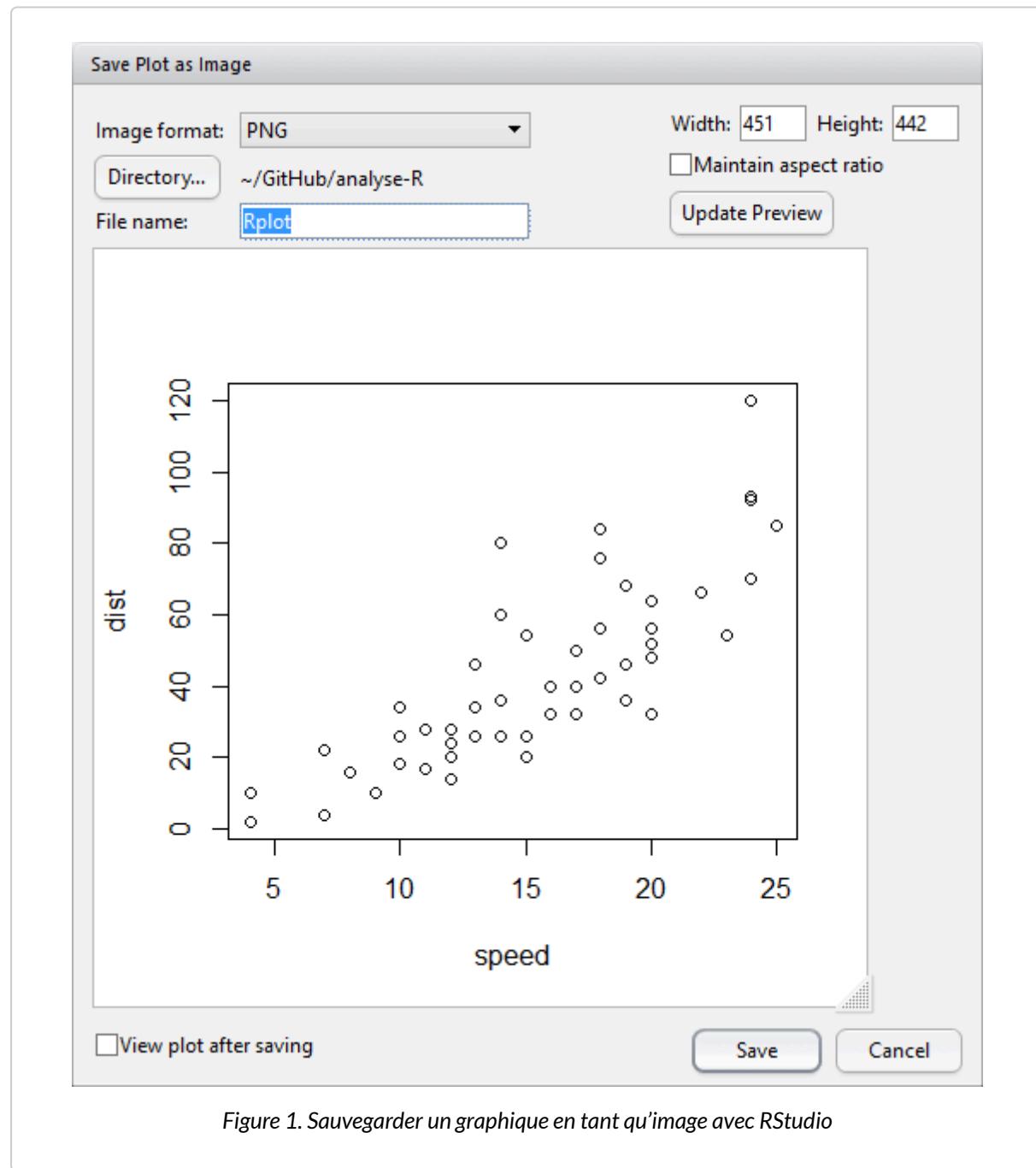


Figure 1. Sauvegarder un graphique en tant qu’image avec RStudio

La boîte de dialogue qui s’ouvre propose différentes options d’export :

- le type de fichier désiré;

- le nom du fichier ;
- le répertoire où le fichier doit être créé (par défaut, il s'agit du répertoire de travail) ;
- la taille de l'image.

R peut exporter un graphique dans une grande variété de formats. Nous n'aborderons ici que les principaux. Les formats **PNG**, **JPEG** et **TIFF** sont des formats de type bitmap (on parle aussi d'images matricielles<sup>1</sup>). L'image est stockée sous forme de points, sa qualité dépendant de sa résolution, c'est-à-dire du nombre total de points qui la composent. L'intérêt des images matricielles est d'être toujours interprétées de manière identique quelque soit l'outil utilisé. Par contre, elles ne sont pas adaptées lorsque l'on souhaite effectuer des retouches avec un logiciel de dessin.

Pour une utilisation sur un site web, on privilégiera une résolution d'image modérée (entre 400 et 800 pixels de largeur) et les formats **PNG** ou **JPEG**. Pour un document destiné à être imprimé, on privilierera une résolution plus élevée, pour éviter un phénomène dit de **pixellisation**.

Les images vectorielles<sup>2</sup> ont l'avantage de pouvoir être redimensionnées à volonté sans perte de qualité et produisent des fichiers en général de plus petite taille<sup>3</sup>. Elles sont donc tout à fait adaptées pour l'impression. Si l'on souhaite importer l'image dans **Word**, on choisira le format **Metafile** (le seul compris par ce logiciel). Pour **Libre Office** ou **Open Office**, on choisira le format **SVG**.

**SVG** (*scalable vector graphic*<sup>4</sup>) est un format libre permettant de décrire une image vectorielle. Les fichiers **SVG** peuvent être directement lus par la majorité des navigateurs récents (**Firefox**, **Chrome**, ...). De plus, le logiciel libre de dessins **Inkscape**<sup>5</sup> permet d'éditer et de modifier des fichiers **SVG**. Ce format est donc tout à fait adapté pour les graphiques que l'on souhaite retoucher avant publication. Depuis **Inkscape**, il sera possible de faire un export **PNG** en haute résolution pour intégration dans un fichier **Word**.

On pourra modifier la taille de l'image avec les paramètres *Height* (hauteur) et *Width* (largeur). En cliquant sur *Update Preview* la prévisualisation du rendu final sera mise à jour.

1. Voir [http://fr.wikipedia.org/wiki/Image\\_matricielle](http://fr.wikipedia.org/wiki/Image_matricielle).

2. Voir [http://fr.wikipedia.org/wiki/Image\\_vectorielle](http://fr.wikipedia.org/wiki/Image_vectorielle).

3. Sauf dans le cas des graphiques complexes reposant sur des dégradés de couleurs, comme les cartes produites à partir de rasters. Auquel cas, il sera parfois préférable de privilier un export dans un format *bitmap*.

4. Voir [https://www.wikiwand.com/fr/Scalable\\_Vector\\_Graphics](https://www.wikiwand.com/fr/Scalable_Vector_Graphics).

5. téléchargeable gratuitement sur <https://inkscape.org/fr/>.

## Sauvegarder le graphique en PDF

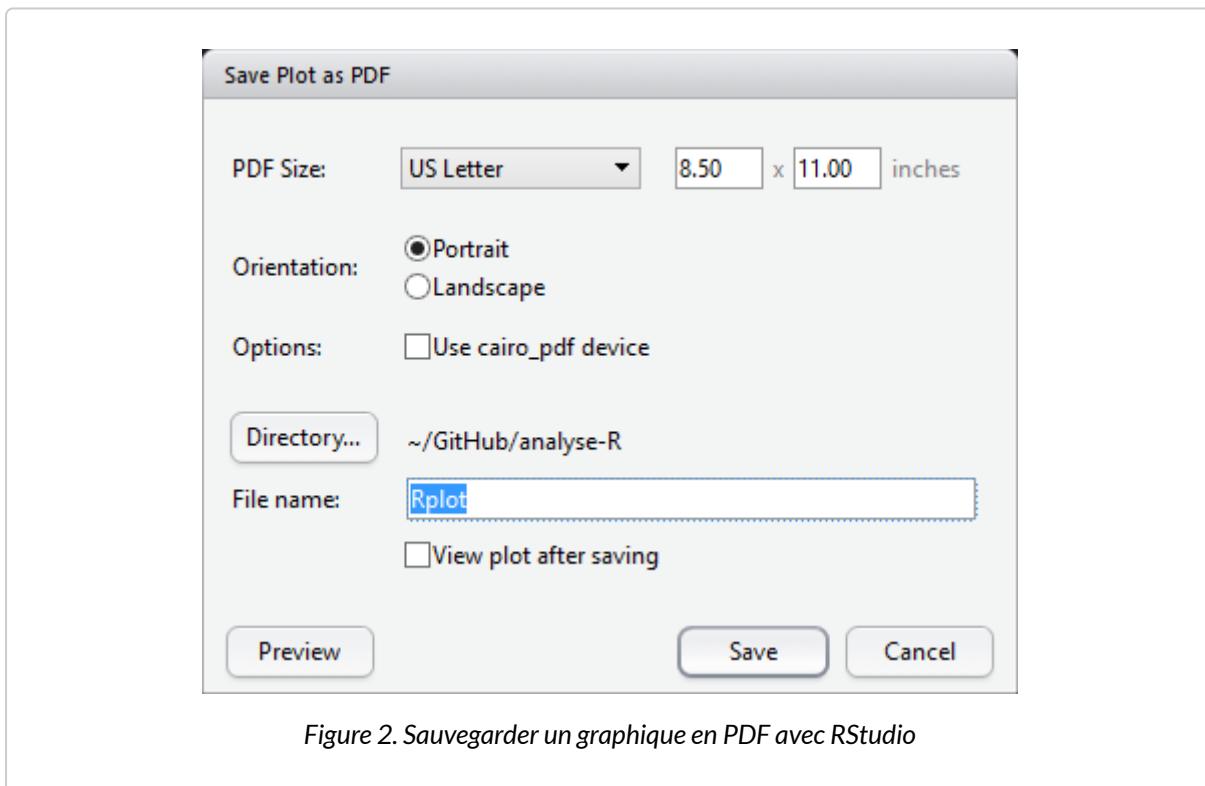


Figure 2. Sauvegarder un graphique en PDF avec RStudio

Les options de la boîte de dialogue permettent de modifier la taille du fichier **PDF** et, bien entendu, d’indiquer le nom et le répertoire du fichier à créer.

En cliquant sur **Preview**, **RStudio** générera un fichier temporaire afin de visualiser le rendu final.

## Copier le graphique dans le presse-papier

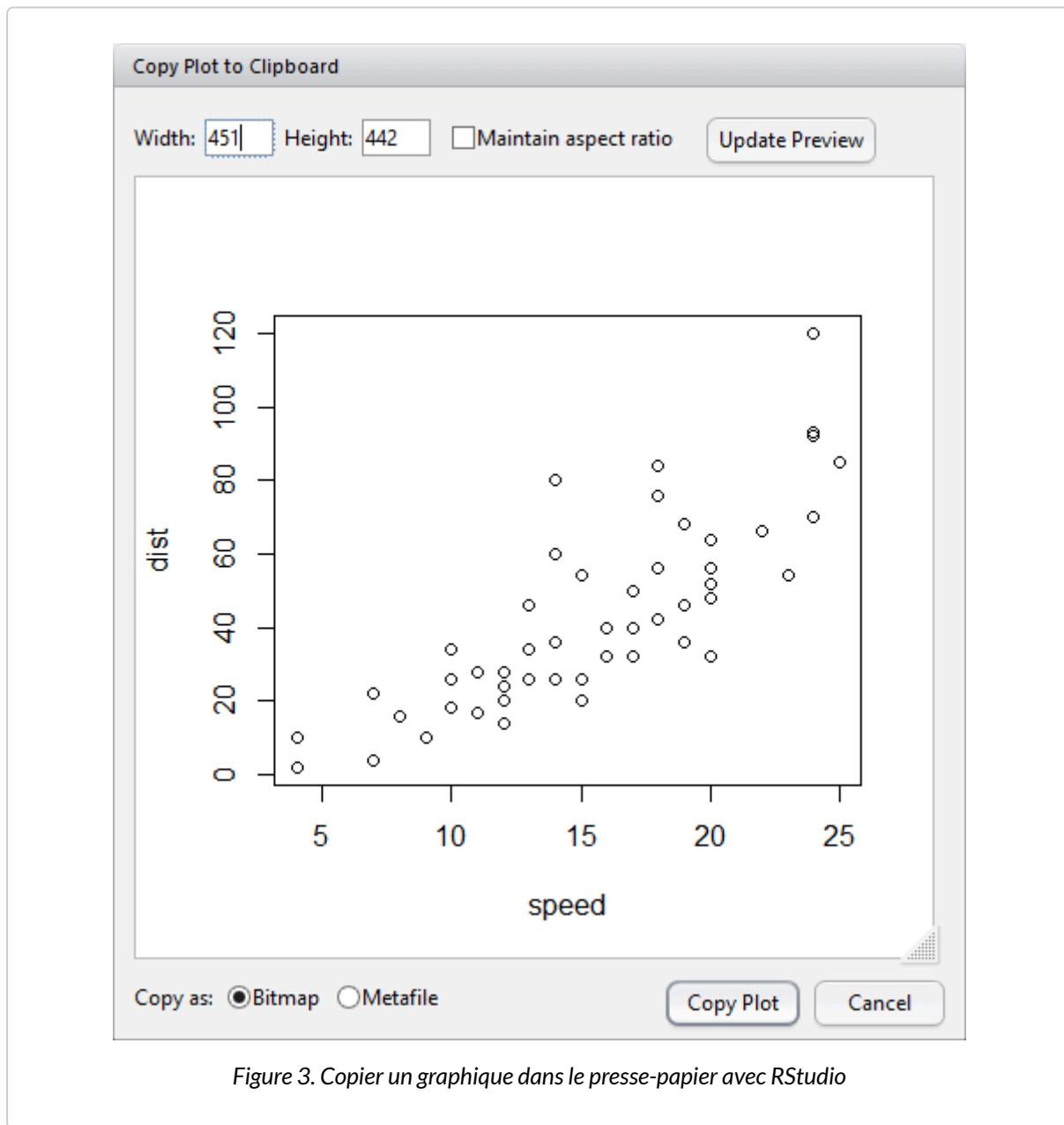


Figure 3. Copier un graphique dans le presse-papier avec RStudio

Il est possible de redimensionner le graphique. De plus, on précisera si l'on souhaite copier une version matricielle (bitmap) ou vectorielle (metafile) du graphique.

## Export avec les commandes de R

On peut également exporter les graphiques dans des fichiers de différents formats directement avec des commandes R. Ceci a l'avantage de fonctionner sur toutes les plateformes et de faciliter la mise à jour du graphique exporté (on n'a qu'à relancer les commandes concernées pour que le fichier externe soit mis à jour).

La première possibilité est d'exporter le contenu d'une fenêtre déjà existante à l'aide de la fonction `dev.print`. On doit fournir à celle-ci le format de l'export (option `device`) et le nom du fichier (option `file`).

Par exemple :

```
R> boxplot(rnorm(100))
  dev.print(device = png, file = "export.png", width = 600)
```

Les formats de sortie possibles varient selon les plateformes, mais on retrouve partout les formats `bitmap`, `png`, `jpeg`, `tiff` et les formats vectoriels `svg`, `postscript` ou `pdf`.

L'autre possibilité est de rediriger directement la sortie graphique dans un fichier, avant d'exécuter la commande générant la figure. On doit pour cela faire appel à l'une des commandes permettant cette redirection. Les plus courantes sont `png`, `jpeg` et `tiff` pour les formats `bitmap`, `svg`, `pdf`, `postscript` et `win.metafile` pour les formats vectoriels.

Ces fonctions prennent différentes options permettant de personnaliser la sortie graphique. Les plus courantes sont `width` et `height` qui donnent la largeur et la hauteur de l'image générée (en pixels pour les images bitmap, en pouces pour les images vectorielles) et `pointsize` qui donne la taille de base des polices de caractère utilisées.

```
R> png(file = "out.png", width = 800, height = 700)
  plot(rnorm(100))
  dev.off()
  pdf(file = "out.pdf", width = 9, height = 9, pointsize = 10)
  plot(rnorm(150))
  dev.off()
```

Il est nécessaire de faire un appel à la fonction `dev.off` après génération du graphique pour que le résultat soit bien écrit dans le fichier de sortie (dans le cas contraire on se retrouve avec un fichier vide).

## Export avec ggplot2

Les graphiques produits par `ggplot2` peuvent être sauvegardés manuellement, comme vu précédemment, ou programmatiquement. Pour sauvegarder le dernier graphique affiché par `ggplot2` au format PNG, il suffit d'utiliser la fonction `ggsave`, qui permet d'en régler la taille (en pouces) et la résolution (en pixels par pouce ; 72 par défaut) :

```
R> ggsave("mon_graphique.png", width = 11, height = 8)
```

De la même manière, pour sauvegarder n'importe quel graphique construit avec `ggplot2` et stocké dans un objet, il suffit de préciser le nom de cet objet, comme ci-dessous, où l'on sauvegarde le graphique contenu dans l'objet `p` au format vectoriel PDF, qui préserve la netteté du texte et des autres éléments du graphique à n'importe quelle résolution d'affichage :

```
R> ggsave("mon_graphique.pdf",
  plot = p,
  width = 11, height = 8
)
```



# Statistique univariée

Variable quantitative .....	340
Principaux indicateurs .....	340
Histogramme .....	341
Densité et répartition cumulée .....	344
Boîtes à moustaches .....	346
Variable qualitative .....	350
Tris à plat .....	350
Représentation graphique .....	355
Tableaux faciles avec <code>gtsummary</code> .....	361
Exporter les graphiques obtenus .....	364

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #03 (statistiques descriptives avec `gtsummary` et esquisse) sur [YouTube](#).

On entend par statistique univariée l'étude d'une seule variable, que celle-ci soit quantitative ou qualitative. La statistique univariée fait partie de la statistique descriptive.

Nous utiliserons dans ce chapitre les données de l'enquête *Histoire de vie 2003* fournies avec l'extension [questionr](#).

```
R> library(questionr)
  data("hdv2003")
  d <- hdv2003
```

## Variable quantitative

### Principaux indicateurs

Comme la fonction `str` nous l’a indiqué, notre tableau `d` contient plusieurs variables numériques ou variables quantitatives, dont la variable `heures.tv` qui représente le nombre moyen passé par les enquêtés à regarder la télévision quotidiennement. On peut essayer de déterminer quelques caractéristiques de cette variable, en utilisant les fonctions `mean` (moyenne), `sd` (écart-type), `min` (minimum), `max` (maximum) et `range` (étendue) :

```
R> mean(d$heures.tv)
```

```
[1] NA
```

```
R> mean(d$heures.tv, na.rm = TRUE)
```

```
[1] 2.246566
```

```
R> sd(d$heures.tv, na.rm = TRUE)
```

```
[1] 1.775853
```

```
R> min(d$heures.tv, na.rm = TRUE)
```

```
[1] 0
```

```
R> max(d$heures.tv, na.rm = TRUE)
```

```
[1] 12
```

```
R> range(d$heures.tv, na.rm = TRUE)
```

```
[1] 0 12
```

On peut lui ajouter la fonction `median` qui donne la valeur médiane, `quantile` qui calcule plus généralement tout type de quantiles, et le très utile `summary` qui donne toutes ces informations ou presque en une seule fois, avec en prime le nombre de valeurs manquantes ( NA ) :

```
R> median(d$heures.tv, na.rm = TRUE)
```

```
[1] 2
```

```
R> quantile(d$heures.tv, na.rm = TRUE)
```

0%	25%	50%	75%	100%
0	1	2	3	12

```
R> summary(d$heures.tv)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
0	1	2	2.246566	3	12	5

La fonction `summary` est une fonction générique qui peut être utilisée sur tout type d'objet, y compris un tableau de données. Essayez donc `summary(d)` .

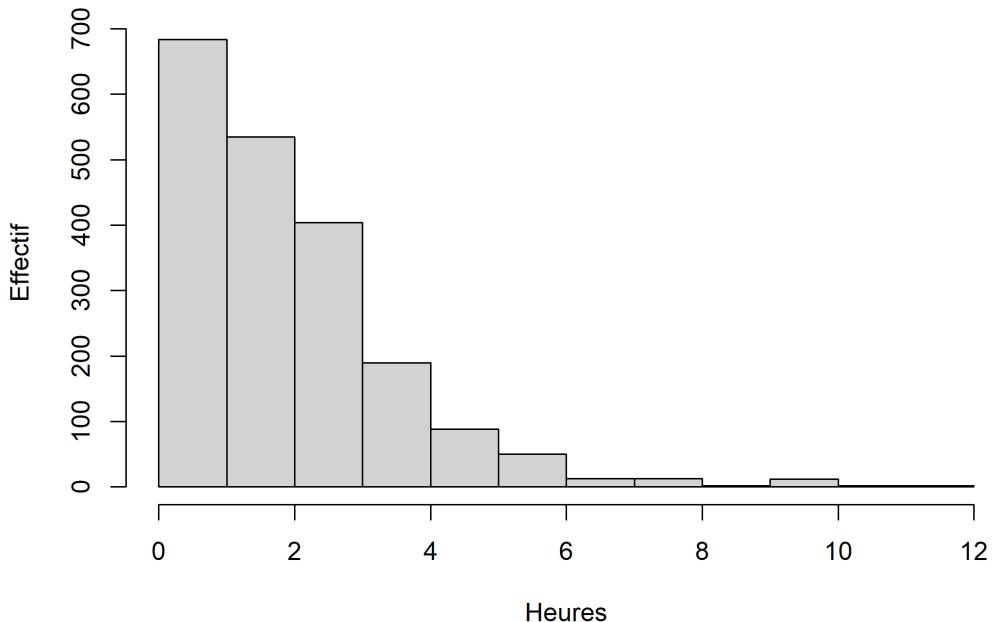
## Histogramme

Tout cela est bien pratique, mais pour pouvoir observer la distribution des valeurs d'une variable quantitative, il n'y a quand même rien de mieux qu'un bon graphique.

On peut commencer par un histogramme de la répartition des valeurs. Celui-ci peut être généré très facilement avec la fonction `hist` :

```
R> hist(d$heures.tv, main = "Nombre d'heures passées devant la télé par jour", xlab = "Heures", ylab = "Effectif")
```

**Nombre d'heures passées devant la télé par jour**



*Figure 1. Exemple d'histogramme*

Sous **RStudio**, les graphiques s'affichent dans l'onglet *Plots* du quadrant inférieur droit. Il est possible d'afficher une version plus grande de votre graphique en cliquant sur *Zoom*.

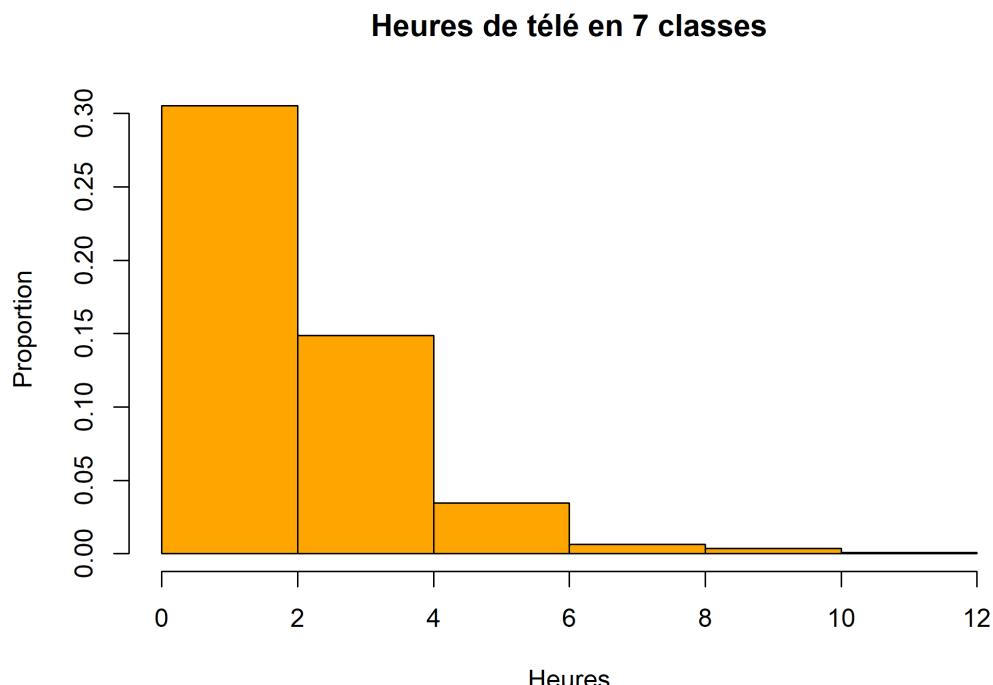
Ici, les options `main`, `xlab` et `ylab` permettent de personnaliser le titre du graphique, ainsi que les étiquettes des axes. De nombreuses autres options existent pour personnaliser l'histogramme, parmi celles-ci on notera :

- `probability` si elle vaut `TRUE`, l'histogramme indique la proportion des classes de valeurs au lieu des effectifs.
- `breaks` permet de contrôler les classes de valeurs. On peut lui passer un chiffre, qui indiquera alors le nombre de classes, un vecteur, qui indique alors les limites des différentes classes, ou encore une chaîne de caractère ou une fonction indiquant comment les classes doivent être calculées.
- `col` la couleur de l'histogramme<sup>1</sup>.

1. Il existe un grand nombre de couleurs prédéfinies dans R. On peut récupérer leur liste en utilisant la fonction `colors`

Voir la page d'aide de la fonction `hist` pour plus de détails sur les différentes options. Les deux figures ci-après sont deux autres exemples d'histogramme.

```
R> hist(d$heures.tv, main = "Heures de télé en 7 classes", breaks = 7, xlab = "Heures", ylab = "Proportion", probability = TRUE, col = "orange")
```



*Figure 2. Un autre exemple d'histogramme*

---

en tapant simplement `colors()` dans la console, ou en consultant le document suivant : <http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf>.

```
R> hist(d$heures.tv, main = "Heures de télé avec classes spécifiées", breaks = c(0, 1, 4, 9, 12), xlab = "Heures", ylab = "Proportion", col = "red")
```

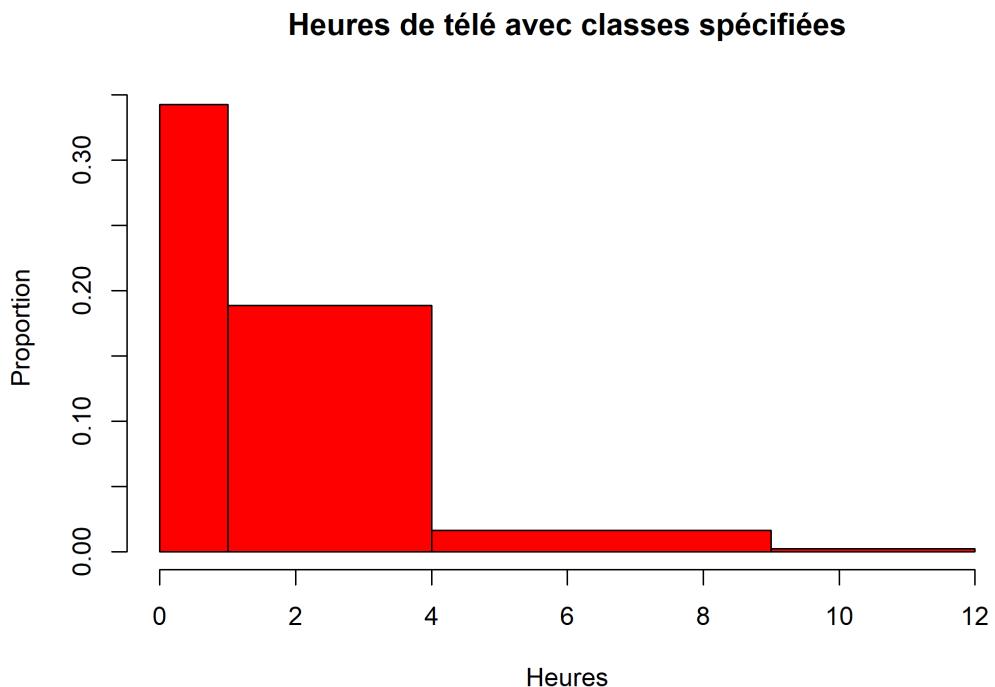


Figure 3. Encore un autre exemple d’histogramme

## Densité et répartition cumulée

La fonction `density` permet d’obtenir une estimation par noyau<sup>2</sup> de la distribution du nombre d’heures consacrées à regarder la télévision. Le paramètre `na.rm = TRUE` indique que l’on souhaite retirer les valeurs manquantes avant de calculer cette courbe de densité.

Le résultat de cette estimation est ensuite représenté graphiquement à l’aide de `plot`. L’argument `main` permet de spécifier le titre du graphique.

---

2. Voir [https://fr.wikipedia.org/wiki/Estimation\\_par\\_noyau](https://fr.wikipedia.org/wiki/Estimation_par_noyau)

```
R> plot(density(d$heures.tv, na.rm = TRUE), main = "Heures consacrées à la télévision")
```

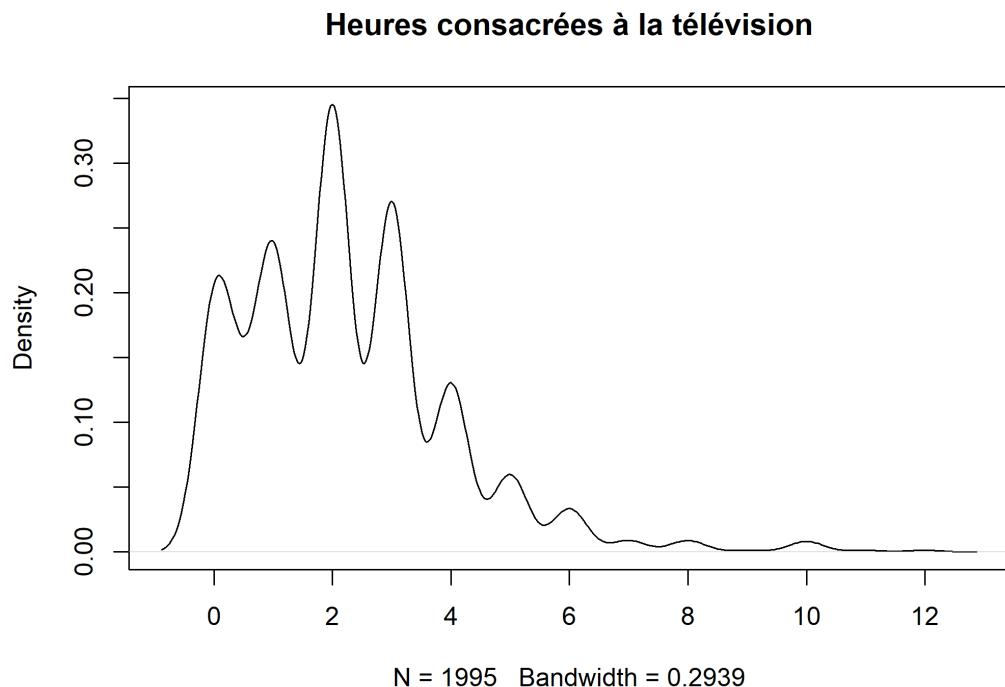


Figure 4. Courbe de densité

De manière similaire, on peut calculer la fonction de répartition empirique ou *empirical cumulative distribution function* en anglais avec la fonction `ecdf`. Le résultat obtenu peut, une fois encore, être représenté sur un graphique à l'aide de la fonction `plot`.

```
R> plot(ecdf(d$heures.tv))
```

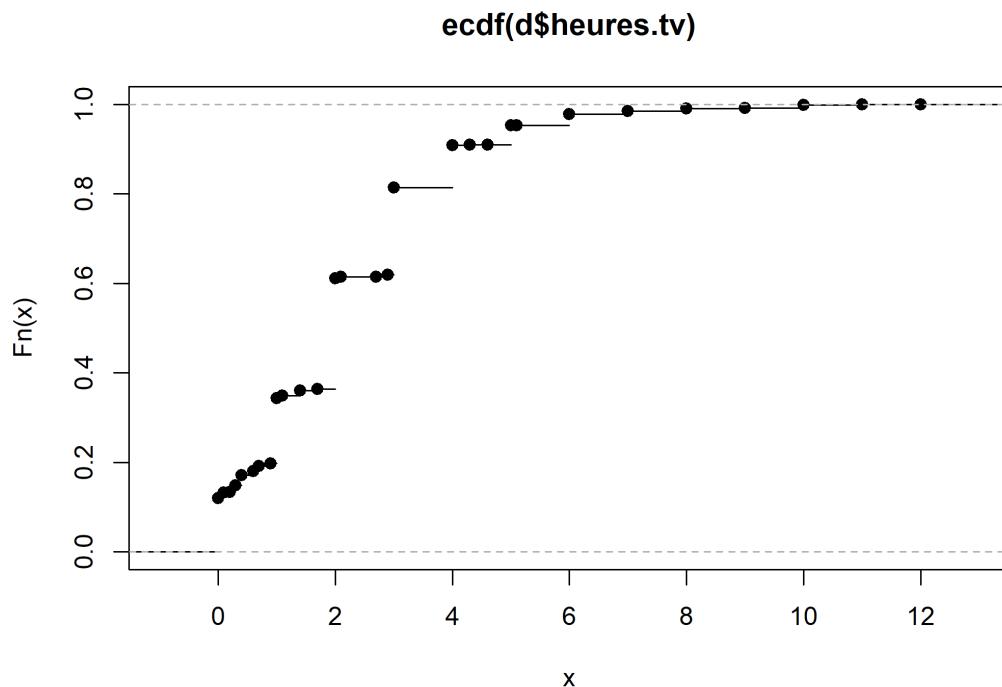


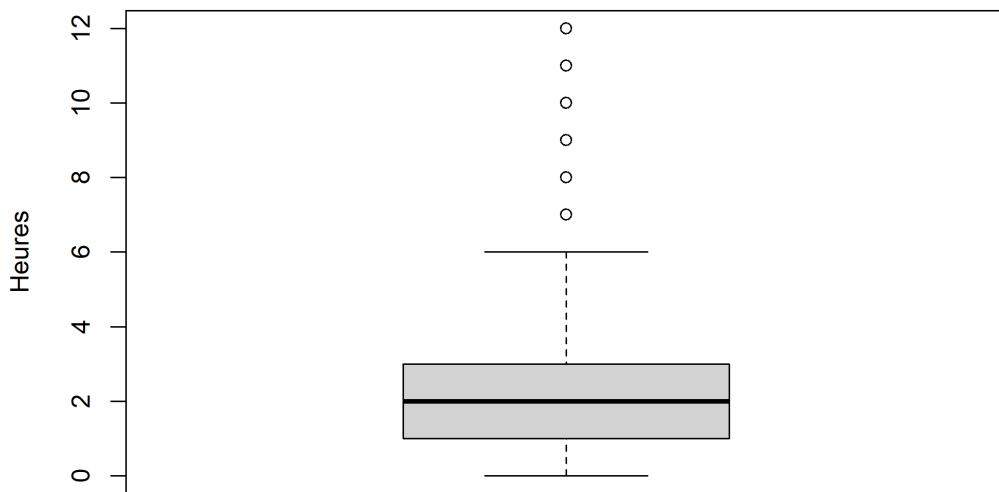
Figure 5. Fonction de répartition empirique cumulée

## Boîtes à moustaches

Les boîtes à moustaches, ou *boxplots* en anglais, sont une autre représentation graphique de la répartition des valeurs d'une variable quantitative. Elles sont particulièrement utiles pour comparer les distributions de plusieurs variables ou d'une même variable entre différents groupes, mais peuvent aussi être utilisées pour représenter la dispersion d'une unique variable. La fonction qui produit ces graphiques est la fonction `boxplot`.

```
R> boxplot(d$heures.tv, main = "Nombre d'heures passées devant la télé par jour", ylab = "Heures")
```

**Nombre d'heures passées devant la télé par jour**

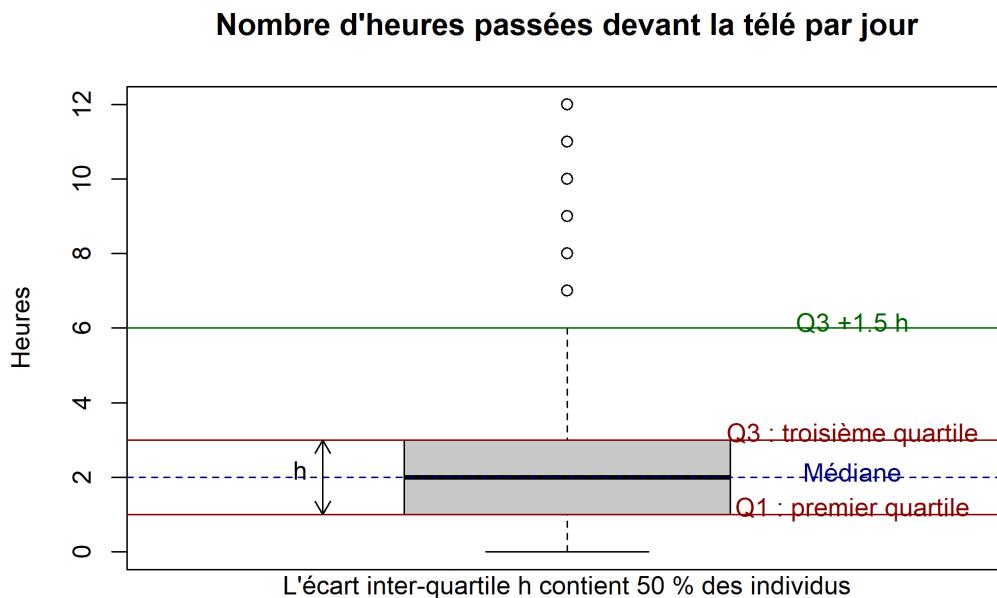


*Figure 6. Exemple de boîte à moustaches*

Comment interpréter ce graphique ? On le comprendra mieux à partir de la figure ci-après<sup>3</sup>.

3. Le code ayant servi à générer cette figure est une copie quasi conforme de celui présenté dans l'excellent document de Jean Lobry sur les graphiques de base avec R, téléchargeable sur le site du Pôle bioinformatique lyonnais : <http://pbil.univ-lyon1.fr/R/pdf/lang04.pdf>.

```
R> boxplot(d$heures.tv, col = grey(0.8), main = "Nombre d'heures passées devant la télé par jour", ylab = "Heures")
abline(h = median(d$heures.tv, na.rm = TRUE), col = "navy", lty = 2)
text(1.35, median(d$heures.tv, na.rm = TRUE) + 0.15, "Médiane", col = "navy")
Q1 <- quantile(d$heures.tv, probs = 0.25, na.rm = TRUE)
abline(h = Q1, col = "darkred")
text(1.35, Q1 + 0.15, "Q1 : premier quartile", col = "darkred", lty = 2)
Q3 <- quantile(d$heures.tv, probs = 0.75, na.rm = TRUE)
abline(h = Q3, col = "darkred")
text(1.35, Q3 + 0.15, "Q3 : troisième quartile", col = "darkred", lty = 2)
arrows(x0 = 0.7, y0 = quantile(d$heures.tv, probs = 0.75, na.rm = TRUE),
x1 = 0.7, y1 = quantile(d$heures.tv, probs = 0.25, na.rm = TRUE), length = 0.1, code = 3)
text(0.7, Q1 + (Q3 - Q1) / 2 + 0.15, "h", pos = 2)
mtext("L'écart inter-quartile h contient 50 % des individus", side = 1)
abline(h = Q1 - 1.5 * (Q3 - Q1), col = "darkgreen")
text(1.35, Q1 - 1.5 * (Q3 - Q1) + 0.15, "Q1 -1.5 h", col = "darkgreen", lty = 2)
abline(h = Q3 + 1.5 * (Q3 - Q1), col = "darkgreen")
text(1.35, Q3 + 1.5 * (Q3 - Q1) + 0.15, "Q3 +1.5 h", col = "darkgreen", lty = 2)
```



*Figure 7. Interprétation d'une boîte à moustaches*

Le carré au centre du graphique est délimité par les premiers et troisième quartiles, avec la médiane représentée par une ligne plus sombre au milieu. Les « fourchettes » s'étendant de part et d'autre vont soit jusqu'à la valeur minimale ou maximale, soit jusqu'à une valeur approximativement égale au quartile le plus proche plus 1,5 fois l'écart interquartile. Les points se situant en-dehors de cette fourchette sont représentés par des petits ronds et sont généralement considérés comme des valeurs extrêmes, potentiellement aberrantes.

On peut ajouter la représentation des valeurs sur le graphique pour en faciliter la lecture avec des petits traits dessinés sur l'axe vertical (fonction `rug`) :

```
R> boxplot(d$heures.tv, main = "Nombre d'heures passées devant la télé par\njour", ylab = "Heures")\nrug(d$heures.tv, side = 2)
```

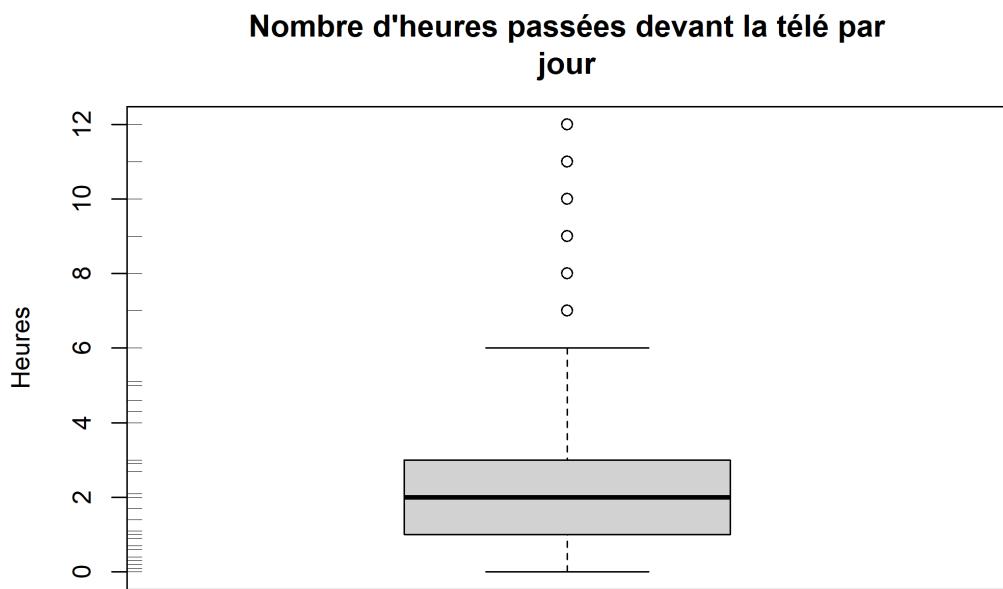


Figure 8. Boîte à moustaches avec représentation des valeurs

## Variable qualitative

### Tris à plat

La fonction la plus utilisée pour le traitement et l'analyse des variables qualitatives (variable prenant ses valeurs dans un ensemble de modalités) est sans aucun doute la fonction `table`, qui donne les effectifs de chaque modalité de la variable, ce qu'on appelle un tri à plat ou tableau de fréquences.

```
R> table(d$sexe)
```

Homme	Femme
899	1101

La tableau précédent nous indique que parmi nos enquêtés on trouve 899 hommes et 1101 femmes.

Quand le nombre de modalités est élevé, on peut ordonner le tri à plat selon les effectifs à l'aide de la fonction `sort`.

```
R> table(d$occup)
```

Exerce une profession	Chomeur	Etudiant, eleve	Retraite	Retire des affaires	Au foyer	Autre inactif
1049	134	94	392	77	171	83

```
R> sort(table(d$occup))
```

Retire des affaires	Autre inactif	Etudiant, eleve	Chomeur	Au foyer	Retraite	Exerce une profession
77	83	94	134	171	392	1049

```
R> sort(table(d$occup), decreasing = TRUE)
```

Exerce une profession	Retraite	Au foyer	Chomeur	Etudiant, eleve	Autre inactif	Retire des affaires
1049	392	171	134	94	83	77

À noter que la fonction `table` exclut par défaut les non-réponses du tableau résultat. L'argument `useNA` de cette fonction permet de modifier ce comportement :

- avec `useNA="no"` (valeur par défaut), les valeurs manquantes ne sont jamais incluses dans le tri à plat ;
- avec `useNA="ifany"`, une colonne `NA` est ajoutée si des valeurs manquantes sont présentes dans les données ;
- avec `useNA="always"`, une colonne `NA` est toujours ajoutée, même s'il n'y a pas de valeurs

manquantes dans les données.

On peut donc utiliser :

```
R> table(d$trav.satisf, useNA = "ifany")
```

Satisfaction	Insatisfaction	Equilibre	NA
480	117	451	952

L'utilisation de `summary` permet également l'affichage du tri à plat et du nombre de non-réponses :

```
R> summary(d$trav.satisf)
```

Satisfaction	Insatisfaction	Equilibre	NA 's
480	117	451	952

Pour obtenir un tableau avec la répartition en pourcentages, on peut utiliser la fonction `freq` de l'extension `questionr`<sup>4</sup>.

```
R> freq(d$qualif)
```

	n	%	val%
Ouvrier specialise	203	10.2	12.3
Ouvrier qualifie	292	14.6	17.7
Technicien	86	4.3	5.2
Profession intermediaire	160	8.0	9.7
Cadre	260	13.0	15.7
Employe	594	29.7	35.9
Autre	58	2.9	3.5
NA	347	17.3	NA

La colonne `n` donne les effectifs bruts, la colonne `%` la répartition en pourcentages et `val%` la

---

4. En l'absence de l'extension `questionr`, on pourra se rabattre sur la fonction `prop.table` avec la commande suivante : `prop.table(table(d$qualif))`.

répartition en pourcentages, données manquantes exclues. La fonction accepte plusieurs paramètres permettant d'afficher les totaux, les pourcentages cumulés, de trier selon les effectifs ou de contrôler l'affichage. Par exemple :

```
R> freq(d$qualif, cum = TRUE, total = TRUE, sort = "inc", digits = 2, exclude = N
      A)
```

	n	%	%cum
Autre	58	3.51	3.51
Technicien	86	5.20	8.71
Profession intermédiaire	160	9.68	18.39
Ouvrier spécialisé	203	12.28	30.67
Cadre	260	15.73	46.40
Ouvrier qualifié	292	17.66	64.07
Employé	594	35.93	100.00
Total	1653	100.00	100.00

La colonne `%cum` indique ici le pourcentage cumulé, ce qui est ici une très mauvaise idée puisque pour ce type de variable cela n'a aucun sens. Les lignes du tableau résultat ont été triées par effectifs croissants, les totaux ont été ajoutés, les non-réponses exclues et les pourcentages arrondis à deux décimales.

La fonction `freq` est également en mesure de tenir compte des étiquettes de valeurs lorsqu'on utilise des données labellisées, page 117. Ainsi :

```
R> data(fecondite)
  describe(femmes$region)
```

```
[2000 obs.] Région de résidence
labelled double: 4 4 4 4 4 3 3 3 3 3 ...
min: 1 - max: 4 - NAs: 0 (0%) - 4 unique values
4 value labels: [1] Nord [2] Est [3] Sud [4] Ouest
```

	n	%
[1] Nord	707	35.4
[2] Est	324	16.2
[3] Sud	407	20.3
[4] Ouest	562	28.1

```
Total      2000 100.0
```

```
R> freq(femmes$region)
```

	n	%	val%
[1] Nord	707	35.4	35.4
[2] Est	324	16.2	16.2
[3] Sud	407	20.3	20.3
[4] Ouest	562	28.1	28.1

```
R> freq(femmes$region, levels = "labels")
```

	n	%	val%
Nord	707	35.4	35.4
Est	324	16.2	16.2
Sud	407	20.3	20.3
Ouest	562	28.1	28.1

```
R> freq(femmes$region, levels = "values")
```

	n	%	val%
	707	35.4	35.4
	324	16.2	16.2
	407	20.3	20.3
	562	28.1	28.1

Pour plus d'informations sur la fonction `freq`, consultez sa page d'aide en ligne avec `?freq` ou `help("freq")`.

## Représentation graphique

Pour représenter la répartition des effectifs parmi les modalités d'une variable qualitative, on a souvent tendance à utiliser des diagrammes en secteurs (camemberts). Ceci est possible sous R avec la fonction `pie`, mais la page d'aide de la dite fonction nous le déconseille assez vivement : les diagrammes en secteur sont en effet une mauvaise manière de présenter ce type d'information, car l'oeil humain préfère comparer des longueurs plutôt que des surfaces<sup>5</sup>.

On privilégiera donc d'autres formes de représentations, à savoir les diagrammes en bâtons et les diagrammes de Cleveland.

Les diagrammes en bâtons sont utilisés automatiquement par R lorsqu'on applique la fonction générique `plot` à un tri à plat obtenu avec `table`. On privilégiera cependant ce type de représentations pour les variables de type numérique comportant un nombre fini de valeurs. Le nombre de frères, soeurs, demi-frères et demi-soeurs est un bon exemple :

---

5. Voir en particulier <https://www.data-to-viz.com/caveat/pie.html> pour un exemple concret.

```
R> plot(table(d$freres.soeurs), main = "Nombre de frères, soeurs, demi-frères et demi-soeurs", ylab = "Effectif")
```

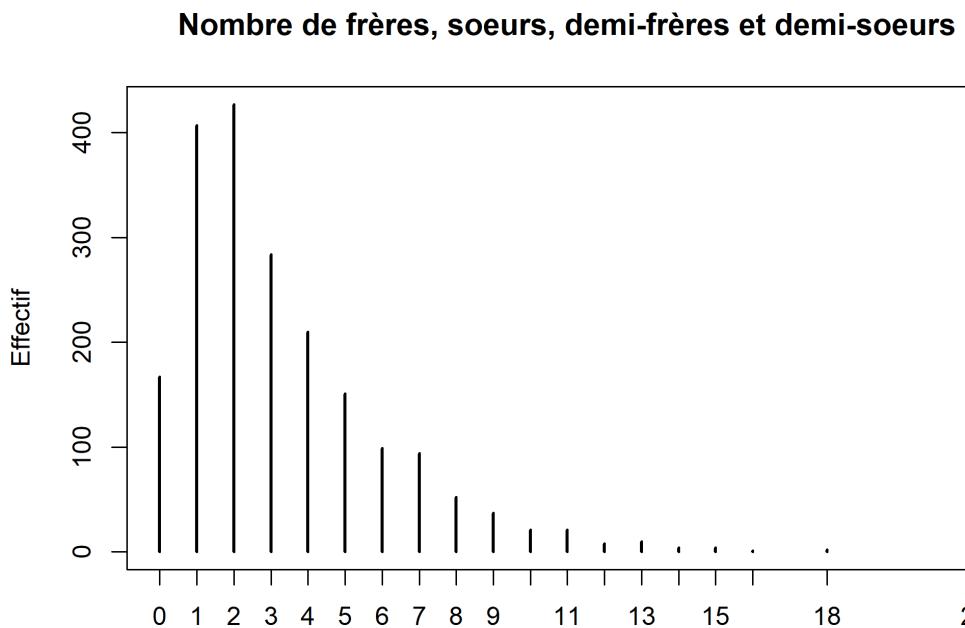


Figure 9. Exemple de diagramme en bâtons

Pour les autres types de variables qualitatives, on privilégiera les diagrammes de Cleveland, obtenus avec la fonction `dotchart`. On doit appliquer cette fonction au tri à plat de la variable, obtenu avec `table`<sup>6</sup>:

6. Pour des raisons liées au fonctionnement interne de la fonction `dotchart`, on doit transformer le tri à plat en matrice, d'où l'appel à la fonction `as.matrix`.

```
R> dotchart(as.matrix(table(d$clso))[, 1], main = "Sentiment d'appartenance  
à une classe sociale", pch = 19)
```

### Sentiment d'appartenance à une classe sociale

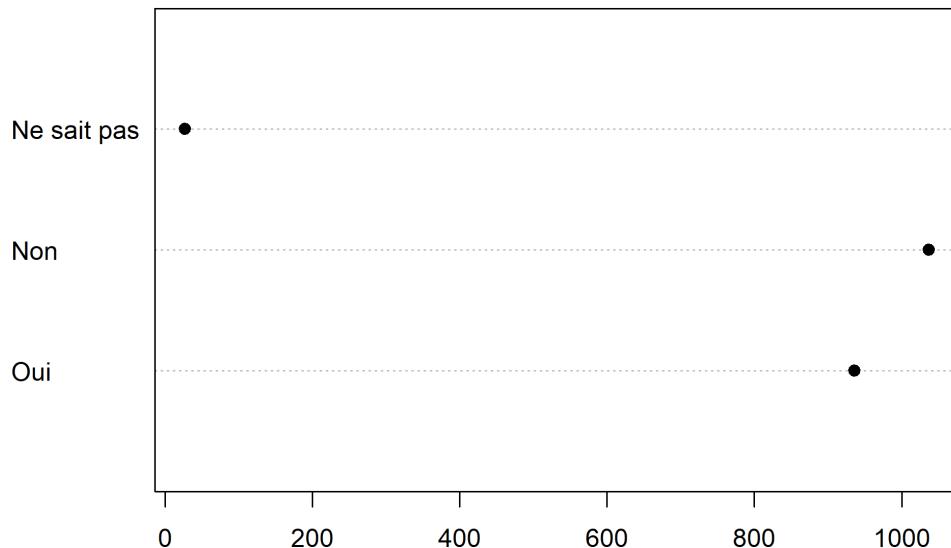


Figure 10. Exemple de diagramme de Cleveland

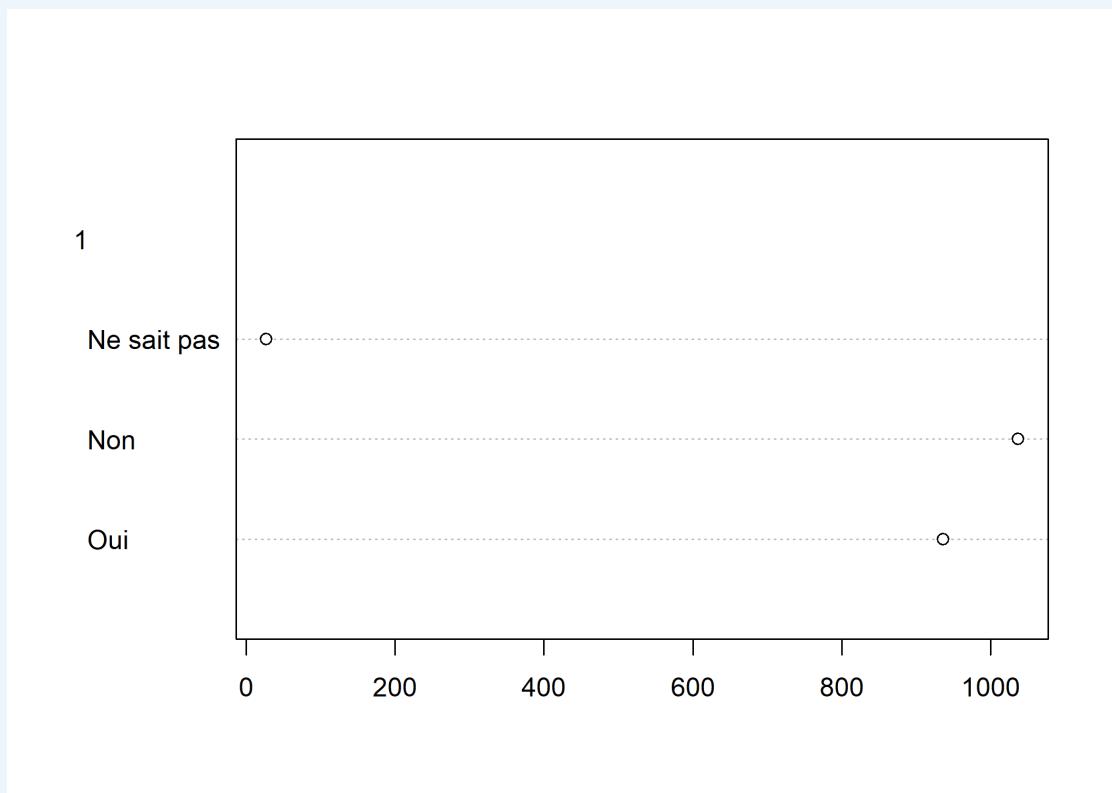
#### NOTE

Il est possible d’entrer directement la commande suivante dans la console :

```
R> dotchart(table(d$c1so))
```

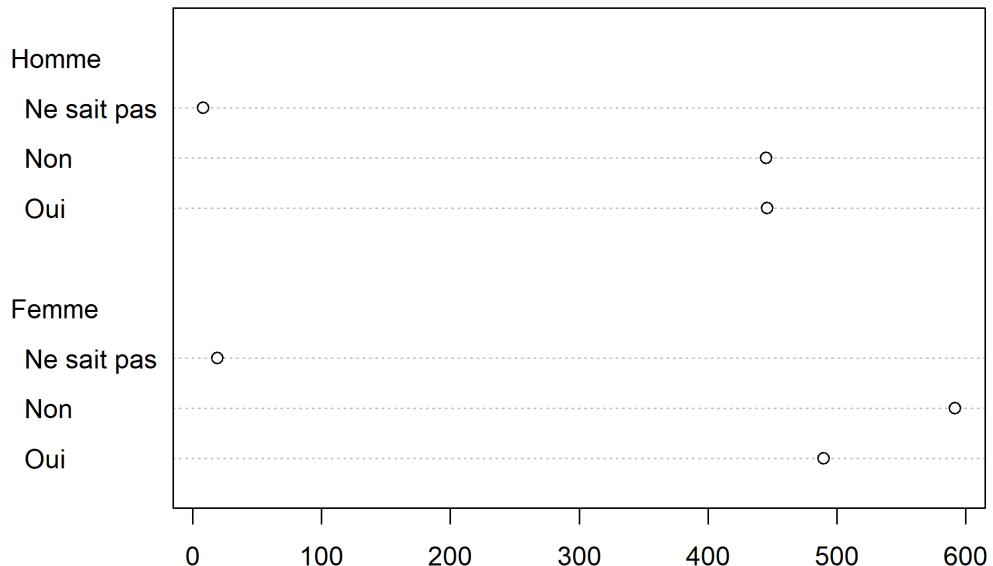
R produira bien le diagramme de Cleveland désiré mais affichera un message d’avertissement (*Warning*) car pour des raisons liées au fonctionnement interne de la fonction `dotchart`, il est attendu une matrice ou un vecteur, non un objet de type table. Pour éviter cet avertissement, il est nécessaire de faire appel à la fonction `as.matrix`.

```
R> dotchart(as.matrix(table(d$c1so)))
```



Dans le cas présent, on voit apparaître un chiffre 1 au-dessus des modalités. En fait, `dotchart` peut être appliqué au résultat d’un tableau croisé à deux entrées, auquel cas il présentera les résultats pour chaque colonne. Comme dans l’exemple ci-après.

```
R> dotchart(as.matrix(table(d$cloo, d$sex)))
```



Cela ne résoud pas le problème pour notre diagramme de Cleveland issu d'un tri à plat simple. Pour bien comprendre, la fonction `as.matrix` a produit un objet à deux dimensions ayant une colonne et plusieurs lignes. On indiquera à R que l'on ne souhaite extraire la première colonne avec `[, 1]` (juste après l'appel à `as.matrix`). C'est ce qu'on appelle l'*indexation*, abordée plus en détail dans le chapitre Listes et tableaux de données, page 87.

Quand la variable comprend un grand nombre de modalités, il est préférable d'ordonner le tri à plat obtenu à l'aide de la fonction `sort` :

```
R> dotchart(as.matrix(sort(table(d$qualif)))[, 1], main = "Niveau de qualification")
```

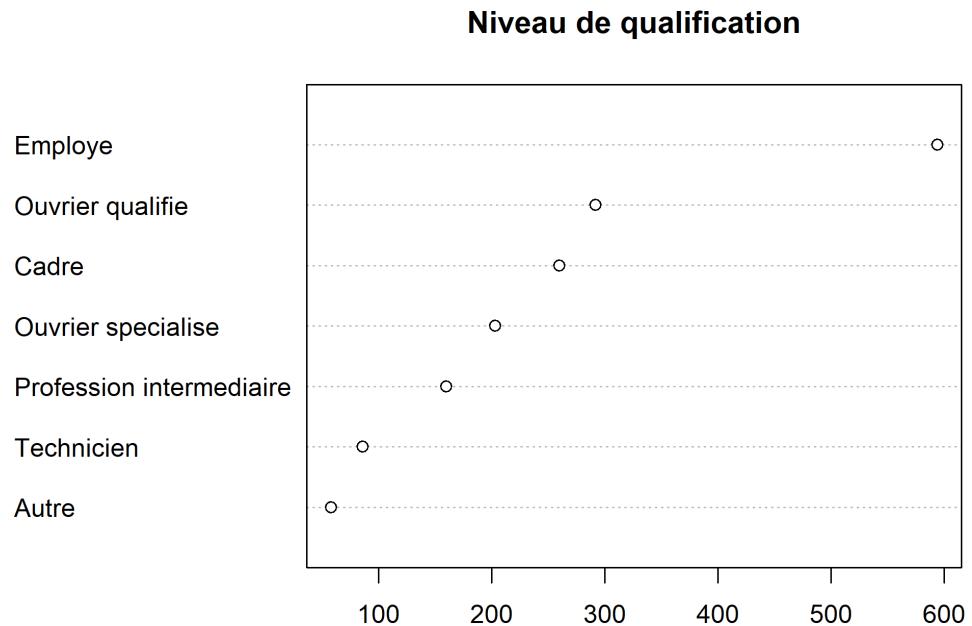
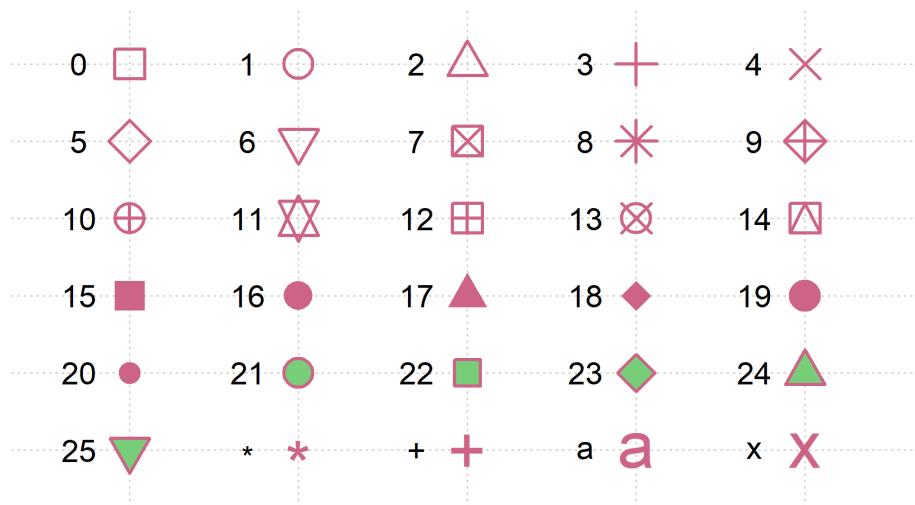


Figure 11. Exemple de diagramme de Cleveland ordonné

**NOTE**

L'argument `pch`, qui est utilisé par la plupart des graphiques de type points, permet de spécifier le symbole à utiliser. Il peut prendre soit un nombre entier compris entre 0 et 25, soit un caractère textuel (voir ci-dessous).

**Différentes valeurs possibles pour l'argument pch**



## Tableaux faciles avec `gtsummary`

L'extension `gtsummary` permet de réaliser facilement des tableaux univariés grâce à la fonction `tbl_summary`. Le résultat produit des tableaux très propres, notamment dans le cadre de rapports automatisés au format Rmarkdown (voir le chapitre dédié, page 979).

On peut indiquer à la fois des variables quantitatives et qualitatives.

```
R> library(gtsummary)
d %>%tbl_summary(include = c("heures.tv", "occup", "qualif"))
```

Caractéristique	N = 2 000 <sup>1</sup>
heures.tv	2,00 (1,00 – 3,00)
Manquant	5
occup	
Exerce une profession	1 049 (52%)
Chomeur	134 (6,7%)
Etudiant, eleve	94 (4,7%)
Retraite	392 (20%)
Retire des affaires	77 (3,9%)
Au foyer	171 (8,6%)
Autre inactif	83 (4,2%)
qualif	
Ouvrier specialise	203 (12%)
Ouvrier qualifie	292 (18%)
Technicien	86 (5,2%)
Profession intermediaire	160 (9,7%)
Cadre	260 (16%)
Employe	594 (36%)
Autre	58 (3,5%)
Manquant	347

<sup>1</sup> Statistique présentée: Médiane (El); n (%)

Il est possible de personnaliser les statistiques présentées (par exemple moyenne et écart-type à la place de la médiane et l'intervale inter-quartile). De plus, `tbl_summary` prend en compte les étiquettes de variables si elles ont été définies (voir le [chapitre dédié](#)). Il est aussi possible de franciser la présentation des résultats.

```
R> library(labelled)
var_label(d$heures.tv) <- "Heures quotidiennes devant la télévision"
var_label(d$occup) <- "Activité"
theme_gtsummary_language("fr", decimal.mark = ",",
big.mark = " ")
```

Setting `language: fr` theme

```
R> d %>%tbl_summary(
  include = c("heures.tv", "occup"),
  statistic = list(all_continuous() ~ "{mean} ({sd})")
)
```

<b>Caractéristique</b>	<b>N = 2 000<sup>1</sup></b>
Heures quotidiennes devant la télévision	2,25 (1,78)
Manquant	5
<b>Activité</b>	
Exerce une profession	1 049 (52%)
Chomeur	134 (6,7%)
Etudiant, élève	94 (4,7%)
Retraite	392 (20%)
Retire des affaires	77 (3,9%)
Au foyer	171 (8,6%)
Autre inactif	83 (4,2%)

<sup>1</sup> Statistique présentée: Moyenne (ET); n (%)

Pour une présentation de toutes les possibilités offertes, voir la vignette dédiée sur [http://www.danielsjoberg.com/gtsummary/articles/tbl\\_summary.html](http://www.danielsjoberg.com/gtsummary/articles/tbl_summary.html).

## Exporter les graphiques obtenus

L’export de graphiques est très facile avec **RStudio**. Lorsque l’on créé un graphique, ce dernier est affiché sous l’onglet *Plots* dans le quadrant inférieur droit. Il suffit de cliquer sur *Export* pour avoir accès à trois options différentes :

- *Save as image* pour sauvegarder le graphique en tant que fichier image ;
- *Save as PDF* pour sauvegarder le graphique dans un fichier **PDF** ;
- *Copy to Clipboard* pour copier le graphique dans le presse-papier (et pouvoir ainsi le coller ensuite dans un document **Word** par exemple).

Pour une présentation détaillée de l’export de graphiques avec **RStudio**, ainsi que pour connaître les commandes **R** permettant d’exporter des graphiques via un script, on pourra se référer au chapitre dédié, page 331.

# Statistique bivariée

Deux variables quantitatives . . . . .	366
Trois variables ou plus . . . . .	374
Une variable quantitative et une variable qualitative . . . . .	376
Représentations graphiques . . . . .	376
Tests statistiques . . . . .	380
Deux variables qualitatives . . . . .	381
Tableau croisé . . . . .	381
Pourcentages en ligne et en colonne . . . . .	384
Représentation graphique . . . . .	386
Tests statistiques . . . . .	389
Tableaux faciles avec <code>gtsummary</code> . . . . .	389

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #03 (statistiques descriptives avec `gtsummary` et esquisse) sur [YouTube](#).

On entend par **statistique bivariée** l'étude des relations entre deux variables, celles-ci pouvant être **quantitatives ou qualitatives**. La statistique bivariée fait partie de la **statistique descriptive**.

La statistique univariée a quant à elle déjà été abordée dans un chapitre dédié, page 339.

Comme dans la partie précédente, on travaillera sur les jeux de données fournis avec l'extension `questionr` et tiré de l'enquête *Histoire de vie* et du recensement 1999 :

```
R> library(questionr)
data(hdv2003)
d <- hdv2003
data(rp99)
```

## Deux variables quantitatives

La comparaison de deux variables quantitatives se fait en premier lieu graphiquement, en représentant l’ensemble des couples de valeurs. On peut ainsi représenter les valeurs du nombre d’heures passées devant la télévision selon l’âge.

```
R> plot(d$age, d$heures.tv)
```

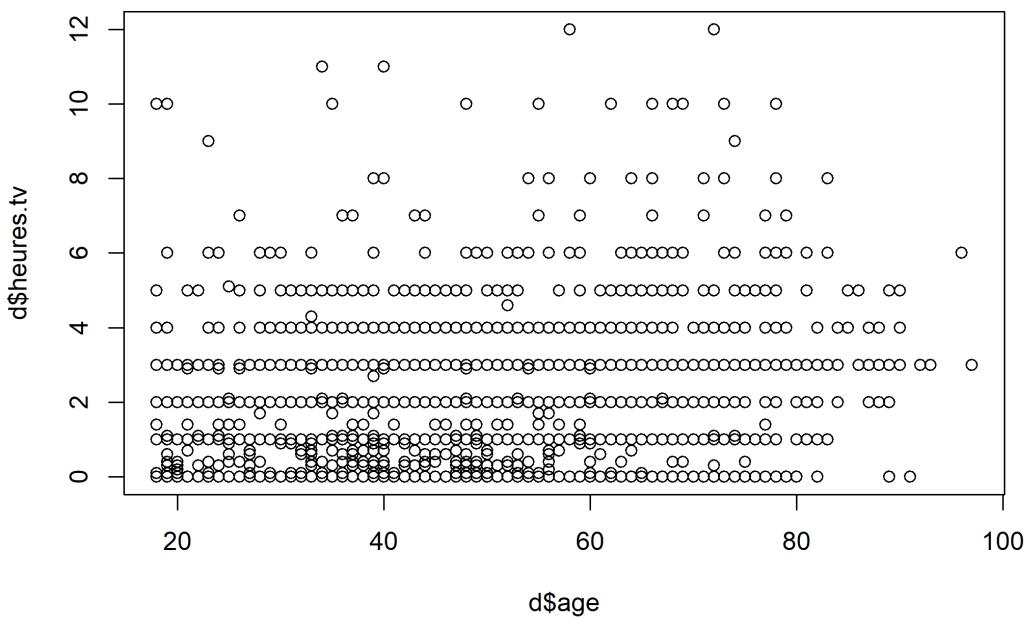


Figure 1. Nombre d'heures de télévision selon l'âge

Le fait que des points sont superposés ne facilite pas la lecture du graphique. On peut utiliser une représentation avec des points semi-transparents.

```
R> plot(d$age, d$heures.tv, pch = 19, col = rgb(1, 0, 0, 0.1))
```

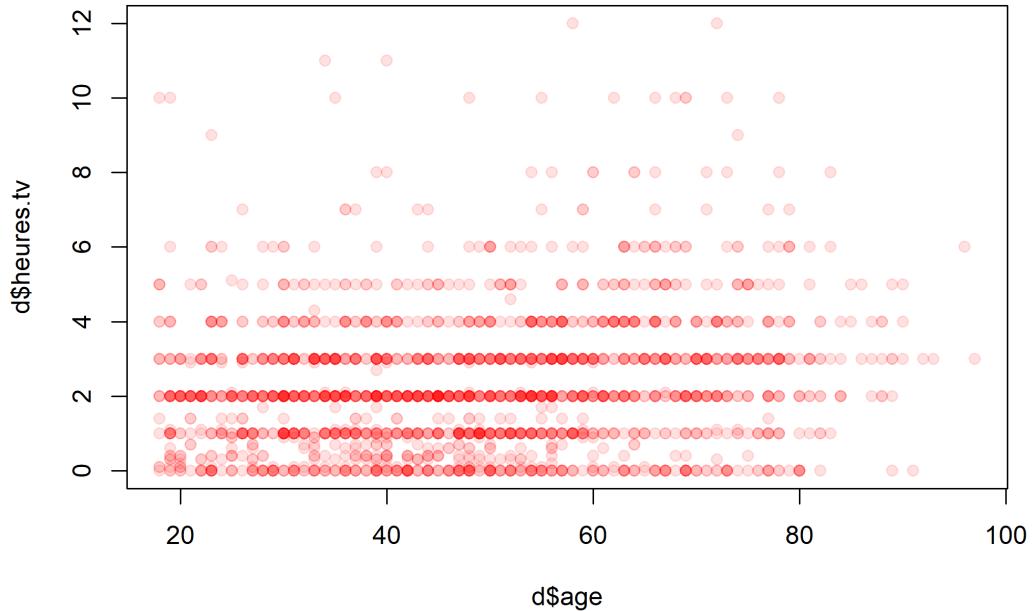


Figure 2. Nombre d'heures de télévision selon l'âge avec semi-transparence

Plus sophistiqué, on peut faire une estimation locale de densité et représenter le résultat sous forme de « carte ». Pour cela on commence par isoler les deux variables, supprimer les observations ayant au moins une valeur manquante à l'aide de la fonction `complete.cases`, estimer la densité locale à l'aide de la fonction `kde2d` de l'extension **MASS**<sup>1</sup> et représenter le tout à l'aide d'une des fonctions `image`, `contour` ou `filled.contour` ...

1. **MASS** est installée par défaut avec la version de base de R.

```
R> library(MASS)
```

```
Attaching package: 'MASS'
```

```
The following object is masked from 'package:formattable':
```

```
area
```

```
The following object is masked from 'package:gtsummary':
```

```
select
```

```
The following object is masked from 'package:dplyr':
```

```
select
```

```
R> tmp <- d[, c("age", "heures.tv")]
  tmp <- tmp[complete.cases(tmp), ]
  filled.contour(kde2d(tmp$age, tmp$heures.tv), color = terrain.colors)
```

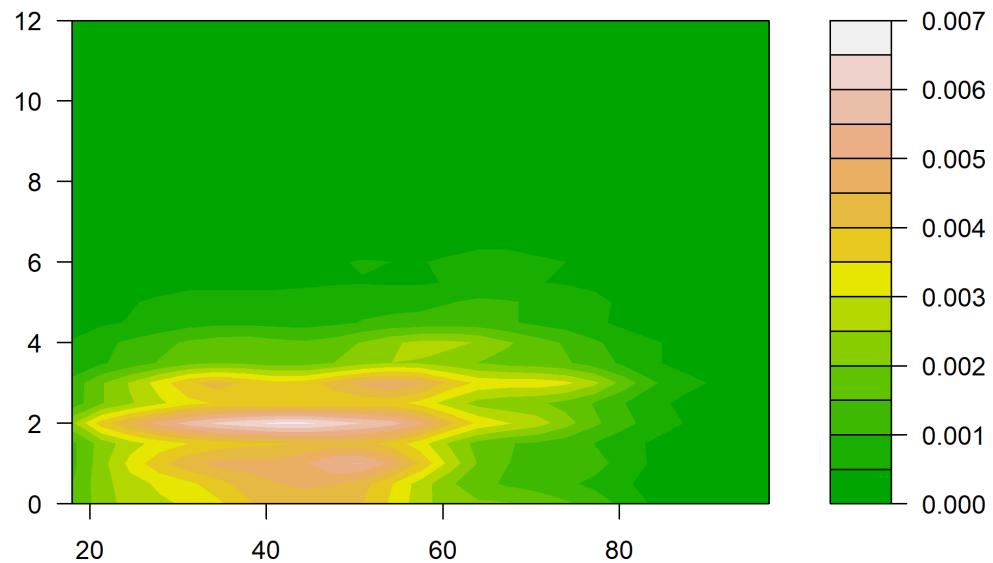


Figure 3. Représentation de l'estimation de densité locale

Une représentation alternative de la densité locale peut être obtenue avec la fonction `smoothScatter`.

```
R> smoothScatter(d[, c("age", "heures.tv")])
```

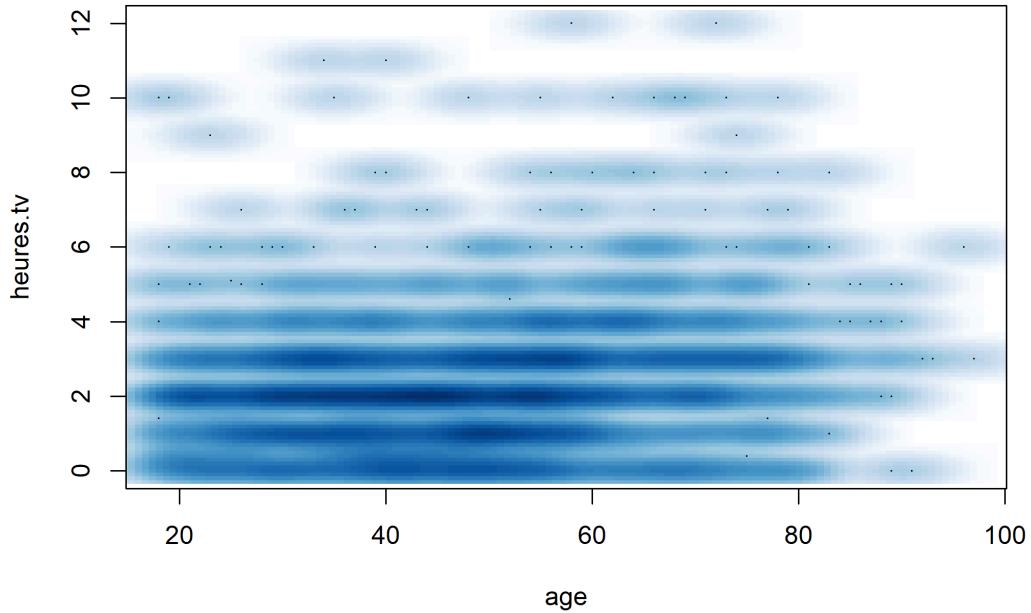


Figure 4. Représentation alternative de l'estimation de densité locale

Dans tous les cas, il n'y a pas de structure très nette qui semble se dégager. On peut tester ceci mathématiquement en calculant le coefficient de corrélation entre les deux variables à l'aide de la fonction `cor` :

```
R> cor(d$age, d$heures.tv, use = "complete.obs")
```

```
[1] 0.1776249
```

L'option `use` permet d'éliminer les observations pour lesquelles l'une des deux valeurs est manquante. Le coefficient de corrélation est très faible.

On va donc s'intéresser plutôt à deux variables présentes dans le jeu de données `rp99`, la part de diplômés du supérieur et la proportion de cadres dans les communes du Rhône en 1999.

À nouveau, commençons par représenter les deux variables.

```
R> plot(rp99$dipl.sup, rp99$cadres, ylab = "Part des cadres", xlab = "Part des diplômés du supérieur")
```

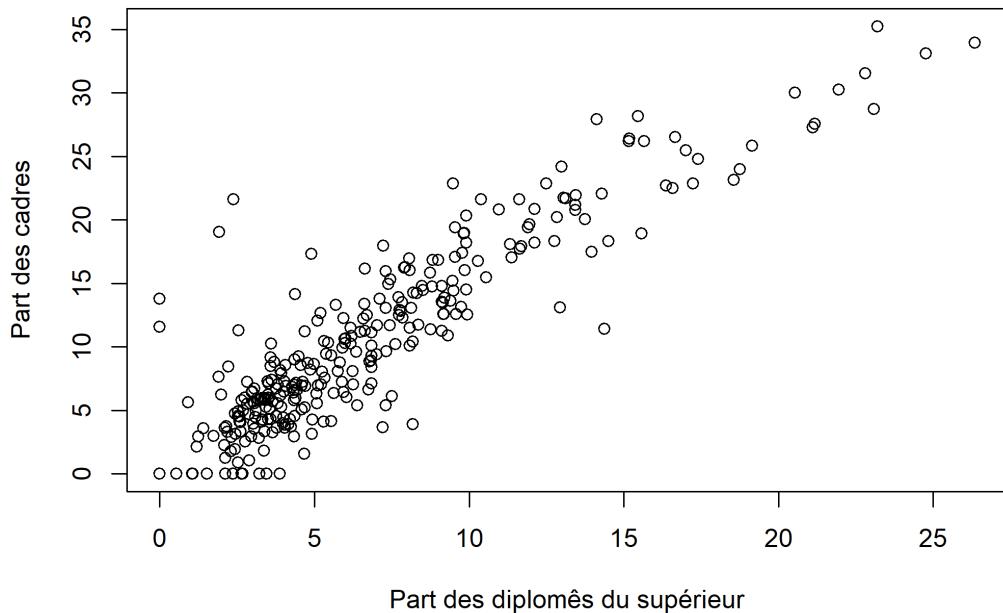


Figure 5. Proportion de cadres et proportion de diplômés du supérieur

Ça ressemble déjà beaucoup plus à une relation de type linéaire.

Calculons le coefficient de corrélation :

```
R> cor(rp99$dipl.sup, rp99$cadres)
```

```
[1] 0.8975282
```

C'est beaucoup plus proche de 1. On peut alors effectuer une régression linéaire complète en utilisant la fonction `lm` :

```
R> reg <- lm(cadres ~ dipl.sup, data = rp99)
summary(reg)
```

```
Call:
lm(formula = cadres ~ dipl.sup, data = rp99)

Residuals:
    Min      1Q  Median      3Q     Max 
-9.6905 -1.9010 -0.1823  1.4913 17.0866 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 1.24088   0.32988   3.762 0.000203 ***
dipl.sup     1.38352   0.03931  35.196 < 2e-16 ***  
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 

Residual standard error: 3.281 on 299 degrees of freedom
Multiple R-squared:  0.8056,    Adjusted R-squared:  0.8049 
F-statistic: 1239 on 1 and 299 DF,  p-value: < 2.2e-16
```

Le résultat montre que les coefficients sont significativement différents de 0. La part de cadres augmente donc avec celle de diplômés du supérieur (ô surprise). On peut très facilement représenter la droite de régression à l'aide de la fonction `abline`.

```
R> plot(rp99$dipl.sup, rp99$cadres, ylab = "Part des cadres", xlab = "Part des diplômés du supérieur")
abline(reg, col = "red")
```

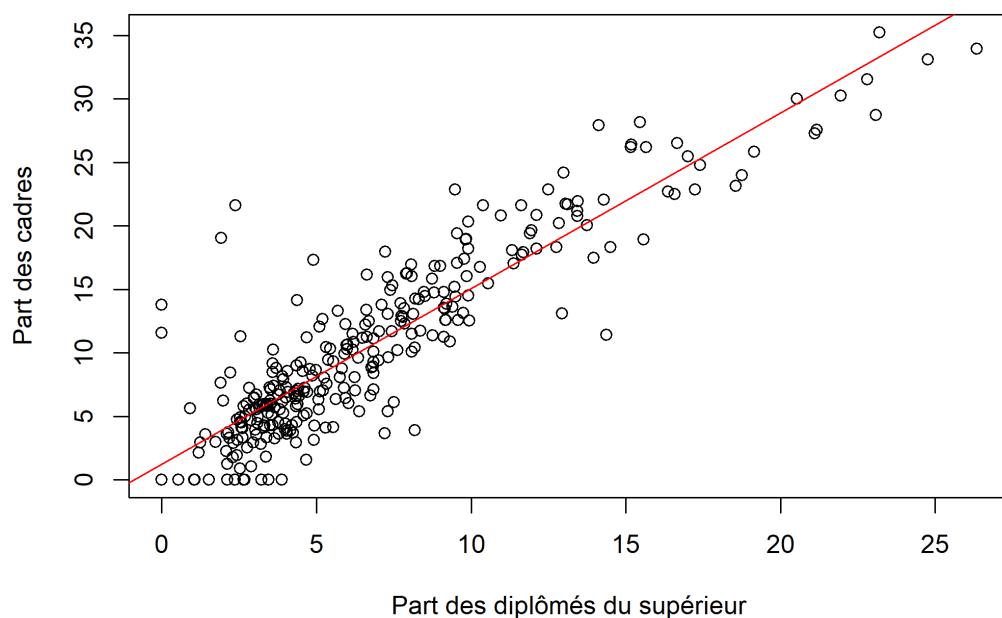


Figure 6. Régression de la proportion de cadres par celle de diplômés du supérieur

#### NOTE

On remarquera que le premier argument passé à la fonction `lm` a une syntaxe un peu particulière. Il s’agit d’une formule, utilisée de manière générale dans les modèles statistiques. On indique la variable d’intérêt à gauche et la variable explicative à droite, les deux étant séparées par un tilde `~` (obtenu sous **Windows** en appuyant simultanément sur les touches `Alt Gr` et `2`). On remarquera que les noms des colonnes de notre tableau de données ont été écrites sans guillemets.

Dans le cas présent, nous avons calculé une régression linéaire simple entre deux variables, d'où l'écriture `cadres ~ dipl.sup`. Si nous avions voulu expliquer une variable `z` par deux variables `x` et `y`, nous aurions écrit `z ~ x + y`. Il est possible de spécifier des modèles encore plus complexes.

Pour un aperçu de la syntaxe des formules sous R, voir <http://ww2.coastal.edu/kingw/statistics/R-tutorials/formulae.html>.

## Trois variables ou plus

Lorsque l’on souhaite représenter trois variables quantitatives simultanément, il est possible de réaliser un nuage de points représentant les deux premières variables sur l’axe horizontal et l’axe vertical et en faisant varier la taille des points selon la troisième variable, en utilisant l’argument `cex` de la fonction `plot`.

```
R> plot(rp99$dipl.aucun, rp99$tx.chom, cex = rp99$pop.tot / 10^4)
```

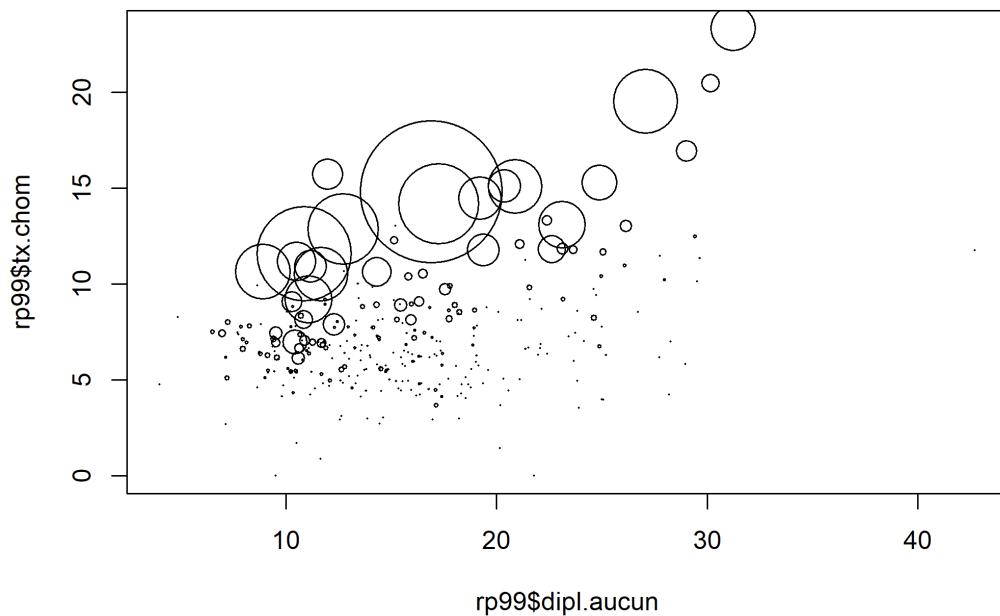


Figure 7. Nuage de points avec taille des points proportionnels à une troisième variable

Lorsque l'on étudie un plus grand nombres de variables quantitatives, il est peut être utile de réaliser une matrice de nuages de points, qui compare chaque variable deux à deux et qui s'obtient facilement avec la fonction `pairs`.

```
R> pairs(rp99[, c("proprio", "hlm", "locataire", "maison")])
```

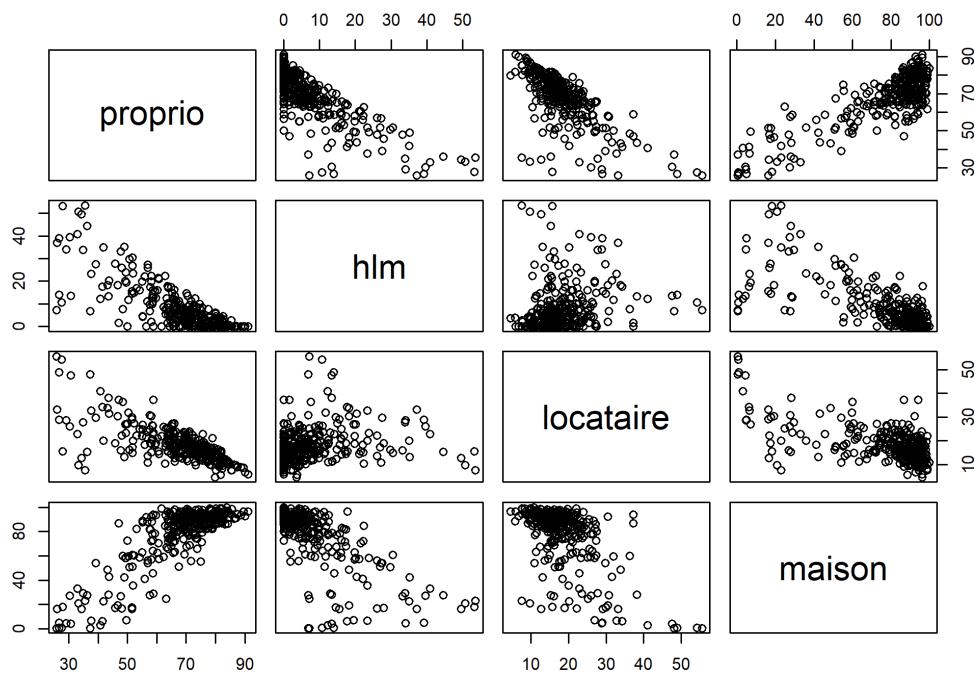


Figure 8. Matrice de nuages de points

## Une variable quantitative et une variable qualitative

### Représentations graphiques

Quand on parle de comparaison entre une variable quantitative et une variable qualitative, on veut en général savoir si la distribution des valeurs de la variable quantitative est la même selon les modalités de la variable qualitative. En clair : est ce que l’âge de ceux qui écoutent du hard rock est différent de l’âge de ceux qui n’en écoutent pas ?

Là encore, l’idéal est de commencer par une représentation graphique. Les boîtes à moustaches (*boxplot* en anglais) sont parfaitement adaptées pour cela.

Si on a construit des sous-populations d'individus écoutant ou non du hard rock, on peut utiliser la fonction `boxplot`.

```
R> d.hard <- subset(d, hard.rock == "Oui")
d.non.hard <- subset(d, hard.rock == "Non")
boxplot(d.hard$age, d.non.hard$age)
```

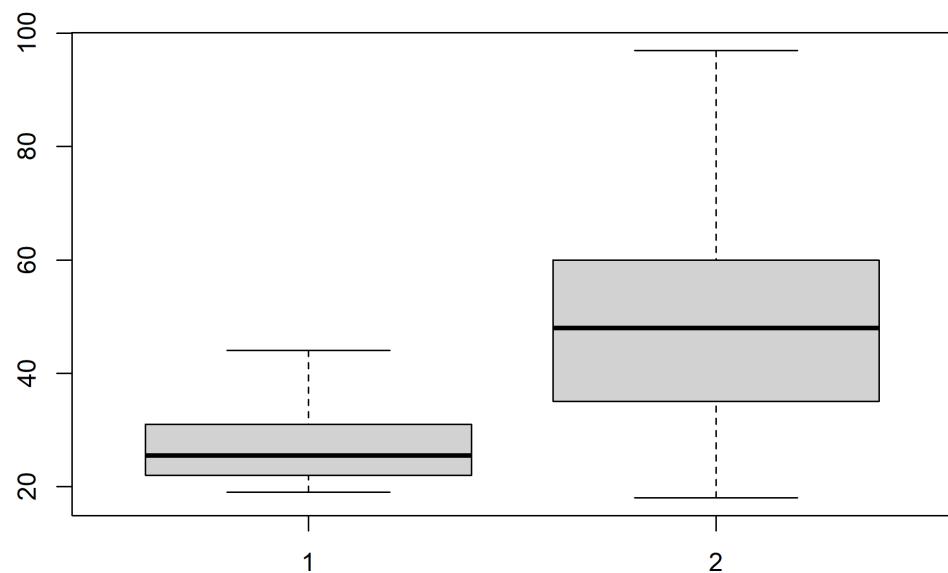


Figure 9. Boxplot de la répartition des âges (sous-populations)

Mais construire les sous-populations n'est pas nécessaire. On peut utiliser directement la version de `boxplot` prenant une formule en argument.

```
R> boxplot(age ~ hard.rock, data = d)
```

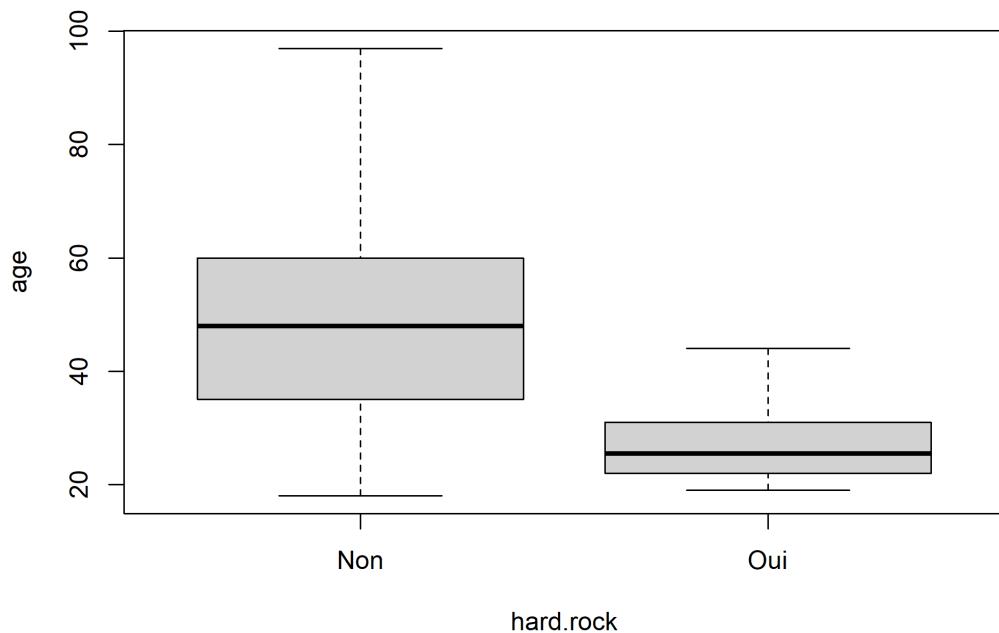


Figure 10. Boxplot de la répartition des âges (formule)

À première vue, ô surprise, la population écoutant du hard rock a l’air sensiblement plus jeune. Peut-on le tester mathématiquement ?

**NOTE**

Les boîtes à moustache peuvent parfois être trompeuses car ne représentant qu'imparfaitement la distribution d'une variable quantitative<sup>2</sup>.

Les graphiques de pirates ou *pirateplot* sont une visualisation alternative qui combinent :

- un nuage de points représentant les données brutes ;
- une barre verticale représentant la moyenne ;
- un rectangle traduisant une inférence sur cette moyenne ;
- une forme en «haricot» ou «violon» indiquant la distribution.

De tels graphiques peuvent être réalisés avec la fonction `pirateplot` de l'extension `yarr`. Par défaut, les rectangles représentent un intervalle bayésien crédible ou *Bayesian Highest Density Intervals* ou *HDI* de la moyenne. On peut représenter à la place des intervalles de confiance avec `inf.method = "ci"`.

---

2. Voir par exemple [The boxplot and its pitfalls](https://www.data-to-viz.com) sur <https://www.data-to-viz.com>.

```
R> library(yarrr)
pirateplot(
  age ~ hard.rock,
  data = d,
  theme = 1, inf.method = "ci",
  bar.f.o = .1, bar.f.col = "grey10"
)
```

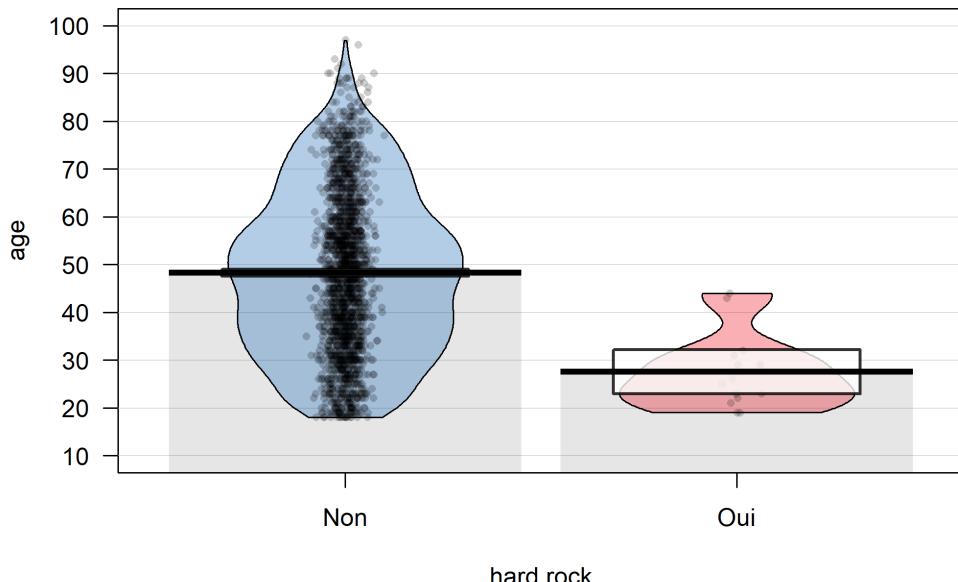


Figure 11. Graphique de «pirates»

## Tests statistiques

On peut calculer la moyenne d’âge des deux groupes en utilisant la fonction `tapply`<sup>3</sup>:

3. La fonction `tapply` est présentée plus en détails dans le chapitre [Manipulation de données](#).

```
R> tapply(d$age, d$hard.rock, mean)
```

	Non	Oui
48.30211	27.57143	

Pour un test de comparaison de deux moyennes (test *t* de Student), on pourra se référer au chapitre dédié aux test statistiques de comparaison, page 485.

## Deux variables qualitatives

La comparaison de deux variables qualitatives s'appelle en général un tableau croisé. C'est sans doute l'une des analyses les plus fréquentes lors du traitement d'enquêtes en sciences sociales.

### Tableau croisé

La manière la plus simple d'obtenir un tableau croisé est d'utiliser la fonction `table` en lui donnant en paramètres les deux variables à croiser. En l'occurrence nous allons croiser un recodage du niveau de qualification regroupé avec le fait de pratiquer un sport.

On commence par calculer la variable recodée et par afficher le tri à plat des deux variables :

```
R> d$qualif2 <- as.character(d$qualif)
d$qualif2[d$qualif %in% c("Ouvrier specialise", "Ouvrier qualifie")] <- "Ouvrier"
d$qualif2[d$qualif %in% c("Profession intermediaire", "Technicien")] <- "Intermediaire"
table(d$qualif2)
```

Autre	Cadre	Employe	Intermediaire	Ouvrier
58	260	594	246	495

Le tableau croisé des deux variables s'obtient de la manière suivante :

```
R> table(d$sport, d$qualif2)
```

/	Autre	Cadre	Employé	Intermédiaire	Ouvrier
Non	38	117	401	127	381
Oui	20	143	193	119	114

#### NOTE

Il est tout à fait possible de croiser trois variables ou plus. Par exemple :

```
R> table(d$sport, d$cuisine, d$sex)
```

```
Var1 Var2 Var3 Freq
Non Non Femme 358
Homme 401
Oui Femme 389
Homme 129
Oui Non Femme 132
Homme 228
Oui Femme 222
Homme 141
```

Une alternative à la fonction `table` est la fonction `xtabs`. On indiquera à cette dernière le croisement à effectuer à l'aide d'une formule puis l'objet contenant nos données. Comme il ne s'agit pas d'un modèle avec une variable à expliquer, toutes les variables seront indiquées à la droite du symbole `~` et séparées par `+`.

```
R> xtabs(~sport, d)
```

Non	Oui
1277	723

```
R> xtabs(~ sport + cuisine, d)
```

sport/cuisine	Non	Oui
Non	759	518
Oui	360	363

```
R> xtabs(~ sport + cuisine + sexe, d)
```

sport	cuisine	sexe	Freq
Non	Non	Femme	358
		Homme	401
Oui	Femme	Femme	389
		Homme	129
Oui	Non	Femme	132
		Homme	228
	Oui	Femme	222
		Homme	141

On remarquera que le rendu par défaut est en général plus lisible car le nom des variables est indiqué, permettant de savoir quelle variable est affichée en colonnes et laquelle en lignes.

Si l'on utilise des données labellisées, page 117, la fonction `xtabs` ne prendra pas en compte les étiquettes de valeur.

```
R> data(fecondite)
> xtabs(~ educ + region, femmes)
```

educ/region	1	2	3	4
0	387	213	282	256
1	179	53	86	142
2	123	57	37	131
3	18	1	2	33

On pourra alors utiliser la fonction `ltabs` de l'extension `question`, qui fonctionne exactement comme `xtabs`, à ceci près qu'elle prendra en compte les étiquettes de variable et de valeur quand elles existent.

```
R> ltabs(~ educ + region, femmes)
```

educ: Niveau d'éducation/region: Région de résidence	[1] Nord	[2] Est	[3] Sud	[4] Ouest
[0] aucun	387	213	282	256
[1] primaire	179	53	86	142
[2] secondaire	123	57	37	131
[3] supérieur	18	1	2	33

## Pourcentages en ligne et en colonne

On n'a cependant que les effectifs, ce qui rend difficile les comparaisons. L'extension `questionr` fournit des fonctions permettant de calculer facilement les pourcentages lignes, colonnes et totaux d'un tableau croisé.

Les pourcentages lignes s'obtiennent avec la fonction `lprop`<sup>4</sup>. Celle-ci s'applique au tableau croisé généré par `table` ou `xtabs` :

```
R> tab <- table(d$sport, d$qualif2)
lprop(tab)
```

/	Autre	Cadre	Employé	Intermédiaire	Ouvrier	Total
Non	3.571429	10.99624	37.68797	11.93609	35.80827	100
Oui	3.395586	24.27844	32.76740	20.20374	19.35484	100
Ensemble	3.508772	15.72898	35.93466	14.88203	29.94555	100

4. Il s'agit en fait d'un alias pour les francophones de la fonction `rprop`.

```
R> tab <- xtabs(~ sport + qualif2, d)
lprop(tab)
```

sport/qualif2	Autre	Cadre	Employe	Intermediaire	Ouvrier	Total
Non	3.571429	10.99624	37.68797	11.93609	35.80827	100
Oui	3.395586	24.27844	32.76740	20.20374	19.35484	100
Ensemble	3.508772	15.72898	35.93466	14.88203	29.94555	100

Les pourcentages ligne ne nous intéressent guère ici. On ne cherche pas à voir quelle est la proportion de cadres parmi ceux qui pratiquent un sport, mais plutôt quelle est la proportion de sportifs chez les cadres. Il nous faut donc des pourcentages colonnes, que l'on obtient avec la fonction `cprop` :

```
R> cprop(tab)
```

sport/ qualif2	Autre	Cadre	Employe	Intermediaire	Ouvrier	Ensemble
Non	65.51724	45	67.50842	51.62602	76.9697	64.36782
Oui	34.48276	55	32.49158	48.37398	23.0303	35.63218
Total	100.00000	100	100.00000	100.00000	100.0000	100.00000

Dans l'ensemble, le pourcentage de personnes ayant pratiqué un sport est de 35,6 %. Mais cette proportion varie fortement d'une catégorie professionnelle à l'autre : 55,0 % chez les cadres contre 23,0 % chez les ouvriers.

Enfin, les pourcentages totaux s'obtiennent avec la fonction `prop` :

```
R> prop(tab)
```

sport/ qualif2	Autre	Cadre	Employe	Intermediaire	Ouvrier	Total
Non	2.298851	7.078040	24.25892	7.683001	23.049002	64.36782
Oui	1.209921	8.650938	11.67574	7.199032	6.896552	35.63218
Total	3.508772	15.728978	35.93466	14.882033	29.945553	100.00000

À noter qu'on peut personnaliser l'affichage de ces tableaux de pourcentages à l'aide de différentes options, dont `digits` qui règle le nombre de décimales à afficher et `percent` qui indique si on souhaite ou non rajouter un symbole `%` dans chaque case du tableau. Cette personnalisation peut se faire directement au moment de la génération du tableau et dans ce cas elle sera utilisée par défaut :

```
R> ctab <- cprop(tab, digits = 2, percent = TRUE)
      ctab
```

<b>sport/ qualif2</b>	<b>Autre</b>	<b>Cadre</b>	<b>Employe</b>	<b>Intermediaire</b>	<b>Ouvrier</b>	<b>Ensemble</b>
Non	65.51724	45	67.50842	51.62602	76.9697	64.36782
Oui	34.48276	55	32.49158	48.37398	23.0303	35.63218
Total	100.00000	100	100.00000	100.00000	100.0000	100.00000

ou bien ponctuellement en passant les mêmes arguments à la fonction `print` :

```
R> ctab <- cprop(tab)
      print(ctab, percent = TRUE)
```

```
      qualif2
sport  Autre Cadre Employe Intermediaire Ouvrier
  Non   65.5% 45.0% 67.5% 51.6%        77.0%
  Oui   34.5% 55.0% 32.5% 48.4%        23.0%
  Total 100.0% 100.0% 100.0% 100.0%       100.0%
      qualif2
sport  Ensemble
  Non   64.4%
  Oui   35.6%
  Total 100.0%
```

## Représentation graphique

On peut obtenir une représentation graphique synthétisant l'ensemble des résultats obtenus sous la forme d'un graphique en mosaïque grâce à la fonction `mosaicplot`.

```
R> mosaicplot(qualif2 ~ sport, data = d, shade = TRUE, main = "Graphe en mosaïque")
```

### Graphe en mosaïque

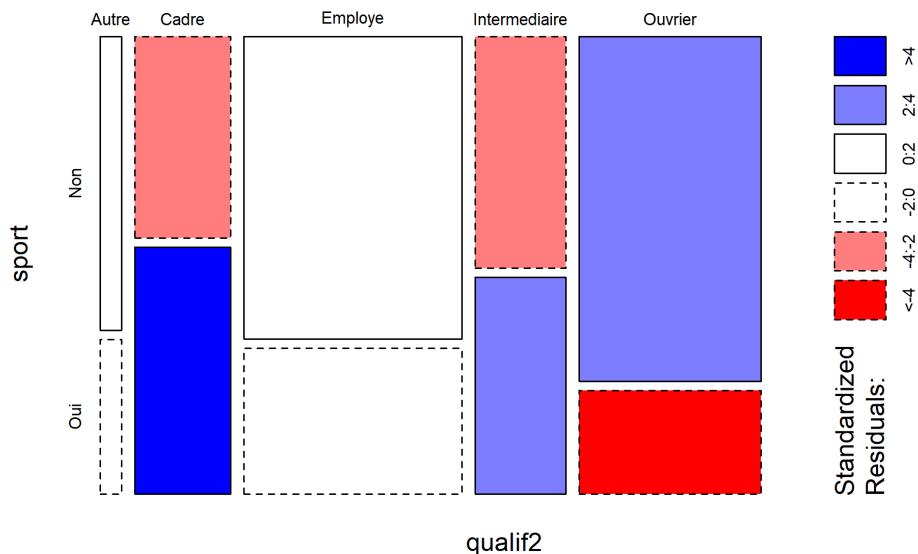


Figure 12. Exemple de graphe en mosaïque

Comment interpréter ce graphique haut en couleurs<sup>5</sup>? Chaque rectangle représente une case de tableau. Sa largeur correspond aux pourcentages en colonnes (il y a beaucoup d'employés et d'ouvriers et très peu d'« Autre »). Sa hauteur correspond aux pourcentages en lignes : la proportion de sportifs chez les cadres est plus élevée que chez les employés. Enfin, la couleur de la case correspond au résidu du test du  $\chi^2$  correspondant : les cases en rouge sont sous-représentées, les cases en bleu sur-représentées, et les cases blanches sont statistiquement proches de l'hypothèse d'indépendance.

5. Sauf s'il est imprimé en noir et blanc...

### NOTE

Les graphiques en mosaïque permettent notamment de représenter des tableaux croisés à 3 ou 4 dimensions, voire plus.

L'extension `vcde` fournit une fonction `mosaic` fournissant plus d'options pour la création d'un graphique en mosaïque, permettant par exemple d'indiquer quelles variables doivent être affichées horizontalement ou verticalement, ou encore de colorier le contenu des rectangles en fonction d'une variable donnée, ...

```
R> library(vcd)
```

```
Loading required package: grid
```

```
R> mosaic(~ sport + cuisine + sexe, d, highlighting = "sexe", main = "Exemple de graphique en mosaïque à 3 dimensions")
```

### Exemple de graphique en mosaïque à 3 dimensions



Lorsque l'on s'intéresse principalement aux variations d'une variable selon une autre, par exemple ici à la

pratique du sport selon le niveau de qualification, il peut être intéressant de présenter les pourcentages en colonne sous la forme de barres cumulées.

```
R> barplot(cprop(tab, total = FALSE), main = "Pratique du sport selon le niveau de qualification")
```

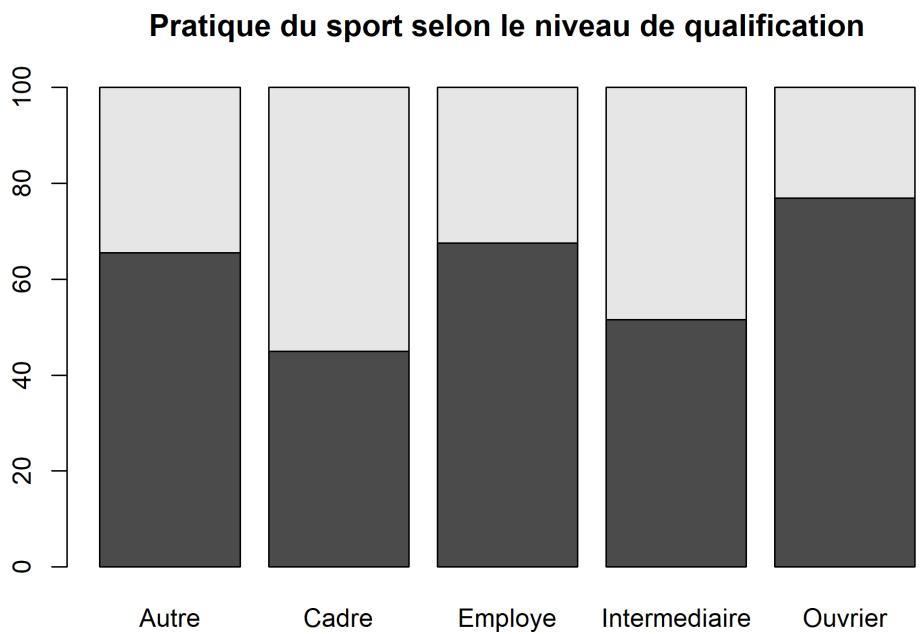


Figure 13. Exemple de barres cumulées

## Tests statistiques

Pour un test de comparaison de proportions, un test du Chi<sup>2</sup> ou encore un test exact de Fisher, on pourra se référer au chapitre dédié aux tests statistiques de comparaison, page 490.

## Tableaux faciles avec `gtsummary`

La fonction `tbl_summary` de l'extension `gtsummary` accepte un argument `by` permettant de réaliser des croisements selon une variable catégorielle.

```
R> library(gtsummary)
  theme_gtsummary_language("fr", decimal.mark = ",",
                           big.mark = " ")
```

Setting `language: fr` theme

```
R> d %>%
 tbl_summary(
  include = c("hard.rock", "heures.tv", "sport", "qualif2"),
  by = "hard.rock"
)
```

<b>Caractéristique</b>	<b>Non, N = 1 986<sup>1</sup></b>	<b>Oui, N = 14<sup>1</sup></b>
heures.tv	2,00 (1,00 – 3,00)	2,00 (2,00 – 3,75)
Manquant	5	0
<b>sport</b>		
Non	1 268 (64%)	9 (64%)
Oui	718 (36%)	5 (36%)
<b>qualif2</b>		
Autre	57 (3,5%)	1 (9,1%)
Cadre	260 (16%)	0 (0%)
Employe	587 (36%)	7 (64%)
Intermediaire	244 (15%)	2 (18%)
Ouvrier	494 (30%)	1 (9,1%)
Manquant	344	3

<sup>1</sup> Statistique présentée: Médiane (El); n (%)

Il est possible d'ajouter une colonne total avec `add_overall` et des tests statistiques avec `add_p`.

```
R> d %>%
 tbl_summary(
  include = c("hard.rock", "heures.tv", "sport", "qualif2"),
  by = "hard.rock"
) %>%
add_overall(last = TRUE) %>%
add_p()
```

Caractéristique	Non, N = 1 986 <sup>1</sup>	Oui, N = 14 <sup>1</sup>	Total, N = 2 000 <sup>1</sup>	p-value <sup>2</sup>
heures.tv	2,00 (1,00 – 3,00)	2,00 (2,00 – 3,75)	2,00 (1,00 – 3,00)	0,4
Manquant	5	0	5	
sport				>0,9
Non	1 268 (64%)	9 (64%)	1 277 (64%)	
Oui	718 (36%)	5 (36%)	723 (36%)	
qualif2				0,088
Autre	57 (3,5%)	1 (9,1%)	58 (3,5%)	
Cadre	260 (16%)	0 (0%)	260 (16%)	
Employe	587 (36%)	7 (64%)	594 (36%)	
Intermediaire	244 (15%)	2 (18%)	246 (15%)	
Ouvrier	494 (30%)	1 (9,1%)	495 (30%)	
Manquant	344	3	347	

<sup>1</sup> Statistique présentée: Médiane (EI); n (%)

<sup>2</sup> Test statistique réalisé: test de Wilcoxon-Mann-Whitney; test du khi-deux d'indépendance; test exact de Fisher

Par défaut, les pourcentages affichés sont les pourcentages en colonne. Mais il est possible de les remplacer par les pourcentages en ligne avec l'argument `percent`.

```
R> d %>%
  dplyr::select(hard.rock, sport, qualif2) %>%
 tbl_summary(
  by = "hard.rock",
  percent = "row"
) %>%
add_overall(last = TRUE)
```

<b>Caractéristique</b>	<b>Non, N = 1 986<sup>1</sup></b>	<b>Oui, N = 14<sup>1</sup></b>	<b>Total, N = 2 000<sup>1</sup></b>
<b>sport</b>			
Non	1 268 (99%)	9 (0,7%)	1 277 (100%)
Oui	718 (99%)	5 (0,7%)	723 (100%)
<b>qualif2</b>			
Autre	57 (98%)	1 (1,7%)	58 (100%)
Cadre	260 (100%)	0 (0%)	260 (100%)
Employe	587 (99%)	7 (1,2%)	594 (100%)
Intermediaire	244 (99%)	2 (0,8%)	246 (100%)
Ouvrier	494 (100%)	1 (0,2%)	495 (100%)
Manquant	344	3	347

<sup>1</sup> Statistique présentée: n (%)

Citons également la variante `tbl_cross` utilisable pour un tableau croisé de seulement deux variables (une en ligne et une en colonne).

```
R> d %>%  
 tbl_cross(sport, hard.rock)
```

Caractéristique	hard.rock		Total
	Non	Oui	
<b>sport</b>			
Non	1 268	9	1 277
Oui	718	5	723
<b>Total</b>	<b>1 986</b>	<b>14</b>	<b>2 000</b>

Pour une présentation de toutes les possibilités offertes, voir la vignette dédiée sur [http://www.danielsjoberg.com/gtsummary/articles/tbl\\_summary.html](http://www.danielsjoberg.com/gtsummary/articles/tbl_summary.html).



# Introduction à ggplot2, la grammaire des graphiques

Les données de l'exemple .....	396
Nettoyage des données .....	397
Recodage d'une variable .....	398
Visualisation des données .....	399
Visualisation par «petits multiples» .....	401
Visualisation en séries temporelles .....	403
Combinaisons d'éléments graphiques .....	404
Composition graphique avec ggplot2 .....	405
Couleurs et échelles .....	407
Utilisation des thèmes .....	411
Export des graphiques .....	413
Pour aller plus loin .....	413
Ressources essentielles .....	415
Extensions de ggplot2 .....	415

## NOTE

Ce chapitre est tiré d'une [séance de cours](#) de François Briatte et destinée à des étudiants de L2 sans aucune connaissance de R. Cette séance de cours est elle-même inspirée d'un [exercice](#) tiré d'un cours de [Cosma Shalizi](#).

R possède un puissant moteur graphique interne, qui permet de «dessiner» dans un graphique en y rajoutant des segments, des points, du texte, ou toutes sortes d'autres symboles. Toutefois, pour produire un graphique complet avec les fonctions basiques de R, il faut un peu bricoler : d'abord, ouvrir une fenêtre ; puis rajouter des points ; puis rajouter des lignes ; tout en configurant les couleurs au fur-et-à-mesure ; puis finir par fermer la fenêtre graphique.

L'extension [ggplot2](#)<sup>1</sup>, développée par Hadley Wickham et mettant en œuvre la «grammaire graphique» théorisée par [Leland Wilkinson](#), devient vite indispensable lorsque l'on souhaite réaliser des graphiques

plus complexes<sup>2</sup>. On renvoie le lecteur intéressé à l’ouvrage de Winston Chang, *R Graphics Cookbook*, disponible gratuitement dans sa deuxième édition à l’adresse suivante : <https://r-graphics.org>.

Ce chapitre, articulé autour d’une étude de cas, présente **ggplot2** à partir d’un exemple simple de visualisation de séries temporelles, puis rentre dans le détail de sa syntaxe. Pour une présentation plus formelle, on pourra se référer au [chapitre dédié](#) de la section [Approfondir](#).

## Les données de l’exemple

Il y a quelques années, les chercheurs Carmen M. Reinhart et Kenneth S. Rogoff publiaient un article intitulé *Growth in a Time of Debt*, dans lequel ils faisaient la démonstration qu’un niveau élevé de dette publique nuisait à la croissance économique. Plus exactement, les deux chercheurs y défendaient l’idée que, lorsque la dette publique dépasse 90 % du produit intérieur brut, ce produit cesse de croître.

Cette conclusion, proche du discours porté par des institutions comme le Fonds Monétaire International, a alimenté plusieurs argumentaires politiques. Des parlementaires américains s’en ainsi sont servi pour exiger une diminution du budget fédéral, et surtout, la Commission européenne s’est appuyée sur cet argumentaire pour exiger que des pays comme la Grèce, durement frappés par la crise financière globale de 2008, adoptent des plans d’austérité drastiques.

Or, en tentant de reproduire les résultats de Reinhart et Rogoff, les chercheurs Thomas Herndon, Michael Ash et Robert Pollin y ont trouvé de nombreuses erreurs, ainsi qu’une bête erreur de calcul due à une utilisation peu attentive du logiciel **Microsoft Excel**. La révélation de ces erreurs donna lieu à un débat très vif entre adversaires et partisans des politiques économiques d’austérité, débat toujours autant d’actualité aujourd’hui.

Dans ce chapitre, on va se servir des données (corrigées) de Reinhart et Rogoff pour évaluer, de manière indépendante, la cohérence de leur argument sur le rapport entre endettement et croissance économique. Commençons par récupérer ces données au format CSV sur le site du chercheur américain [Cosma Shalizi](#), qui utilise ces données dans [l’un de ses exercices de cours](#) :

- 
1. Voir l’excellente [documentation de l’extension](#) et les autres ressources citées en fin de chapitre.
  2. Bien que l’on ait fait le choix de présenter l’extension **ggplot2** plutôt que l’extension **lattice**, celle-ci reste un excellent choix pour la visualisation, notamment, de [panels](#) et de [séries temporelles](#). On trouve de très beaux exemples d’utilisation de **lattice** en ligne, mais un peu moins de documentation, et beaucoup moins d’extensions, que pour **ggplot2**.

```
R> # charger l'extension lisant le format CSV
library(readr)

# emplacement souhaité pour le jeu de données
file <- "data/debt.csv"

# télécharger le jeu de données s'il n'existe pas
if(!file.exists(file))
  download.file("http://www.stat.cmu.edu/~cshalizi/uADA/13/hw/11/debt.csv",
                file, mode = "wb")

# charger les données dans l'objet 'debt'
debt <- read_csv(file)
```

Warning: Missing column names filled in: 'X1' [1]

```
-- Column specification -----
cols(
  X1 = col_double(),
  Country = col_character(),
  Year = col_double(),
  growth = col_double(),
  ratio = col_double()
)
```

#### NOTE

Le code ci-dessus utilise la fonction `read_csv` de l'extension `readr`, dont on a recommandé l'utilisation dans un précédent chapitre, page 145. En l'absence de cette extension, on aurait pu utiliser la fonction de base `read.csv`.

## Nettoyage des données

Les données de Reinhart et Rogoff contiennent, pour un échantillon de 20 pays occidentaux membres de la zone OCDE, la croissance de leur **produit intérieur brut** (PIB)<sup>3</sup>, et le ratio entre leur **dette publique** et ce produit, exprimé sous la forme d'un pourcentage «Dette / PIB». Les données vont du milieu des années 1940 à la fin des années 2000. La première colonne du jeu de données ne contenant que les numéros des lignes, on va la supprimer d'entrée de jeu :

---

3. Ce produit est mesuré en **termes réels**, de manière à ce que le calcul de sa croissance ne soit pas affecté par l'inflation.

```
R> # inspection du jeu de données
str(debt)
```

```
tibble [1,171 x 5] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
$ X1      : num [1:1171] 147 148 149 150 151 ...
$ Country: chr [1:1171] "Australia" "Australia" "Australia" ...
$ Year    : num [1:1171] 1946 1947 1948 1949 1950 ...
$ growth  : num [1:1171] -3.56 2.46 6.44 6.61 6.92 ...
$ ratio   : num [1:1171] 190 177 149 126 110 ...
- attr(*, "spec")=
.. cols(
..   X1 = col_double(),
..   Country = col_character(),
..   Year = col_double(),
..   growth = col_double(),
..   ratio = col_double()
.. )
```

```
R> # suppression de la première colonne
debt <- debt[, -1]
```

Il faut aussi noter d'emblée que certaines mesures sont manquantes : pour certains pays, on ne dispose pas d'une mesure fiable du PIB et/ou de la dette publique. En conséquence, le nombre d'observations par pays est différent, et va de 40 observations «pays-année» pour la Grèce à 64 observations «pays-année» pour plusieurs pays comme l'Australie ou les États-Unis :

```
R> table(debt$Country)
```

Australia	Austria	Belgium	Canada	Denmark
64	59	63	64	56
Finland	France	Germany	Greece	Ireland
64	54	59	40	63
Italy	Japan	Netherlands	New Zealand	Norway
59	54	53	64	64
Portugal	Spain	Sweden	UK	US
58	42	64	63	64

## Recodage d'une variable

Dernière manipulation préalable avant l'analyse : on va calculer la décennie de chaque observation, en divisant l'année de mesure par 10, et en multipliant la partie entière de ce résultat par 10. Cette

manipulation très simple donne «1940» pour les mesures des années 1940 à 1949, «1950» pour les années 1950-1959, et ainsi de suite.

```
R> debt$Decade <- factor(10 * debt$Year %% 10)
```

Voici, pour terminer, les premières lignes du jeu de données sur lequel on travaille :

```
R> head(debt)
```

## Visualisation des données

Chargeons à présent l'extension graphique **ggplot2** :

```
R> library(ggplot2)
```

Procédons désormais à quelques visualisations très simples de ces données. On dispose de trois variables continues : l'année, le taux de croissance du PIB, et le ratio «Dette publique / PIB». Si l'on souhaite visualiser la croissance du PIB au cours du temps, la solution basique dans R s'écrit de la manière suivante :

```
R> with(debt, plot(Year, growth))
```

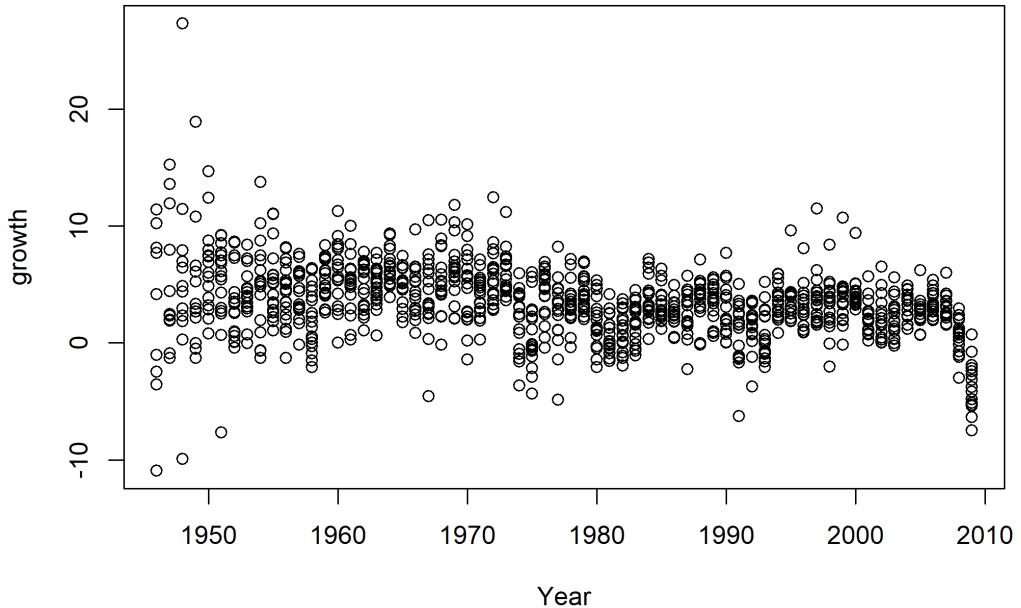


Figure 1

Le code de la visualisation est très simple et se lit : «avec l'objet `debt`, construire le graphique montrant l'année d'observation `Year` en abscisse et le taux de croissance du PIB `growth` en ordonnée». Le code est compris de cette manière par R car la fonction `plot` comprend le premier argument comme étant la variable à représenter sur l'axe horizontal `x`, et le second comme la variable à représenter sur l'axe vertical `y`.

Le même graphique s'écrit de la manière suivante avec l'extension `ggplot2` :

```
R> with(debt, qplot(Year, growth))
```

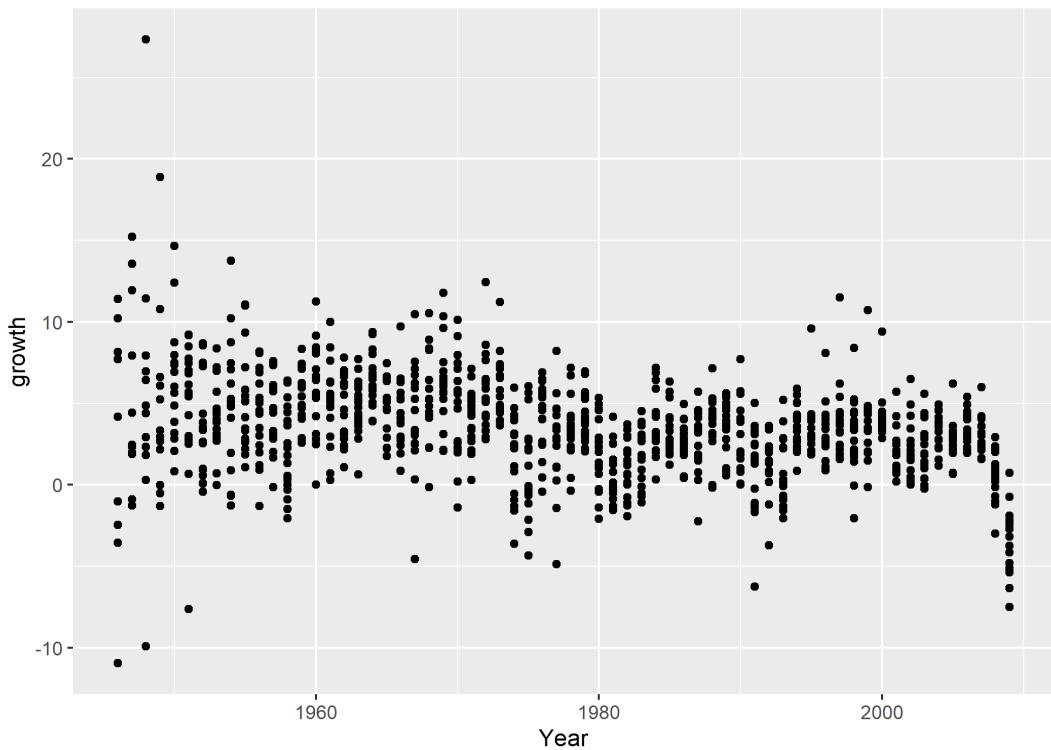


Figure 2

Comme on peut le voir, le code est très proche du code utilisé dans «R base», la syntaxe signifiant toujours : «avec le jeu de données `debt`, visualiser les variables `Year` sur l'axe `x` et `growth` sur l'axe `y`». Le résultat est similaire, bien que plusieurs paramètres graphiques aient changé : le fond gris clair, en particulier, est caractéristique du thème graphique par défaut de `ggplot2`, que l'on apprendra à modifier plus loin.

Par ailleurs, dans les deux exemples précédents, on a écrit `with(debt, ...)` pour indiquer que l'on travaillait avec l'objet `debt`. Lorsque l'on travaille avec l'extension `ggplot2`, il est toutefois plus commun d'utiliser l'argument `data` dans l'appel de `qplot` pour indiquer ce choix :

```
R> qplot(Year, growth, data = debt)
```

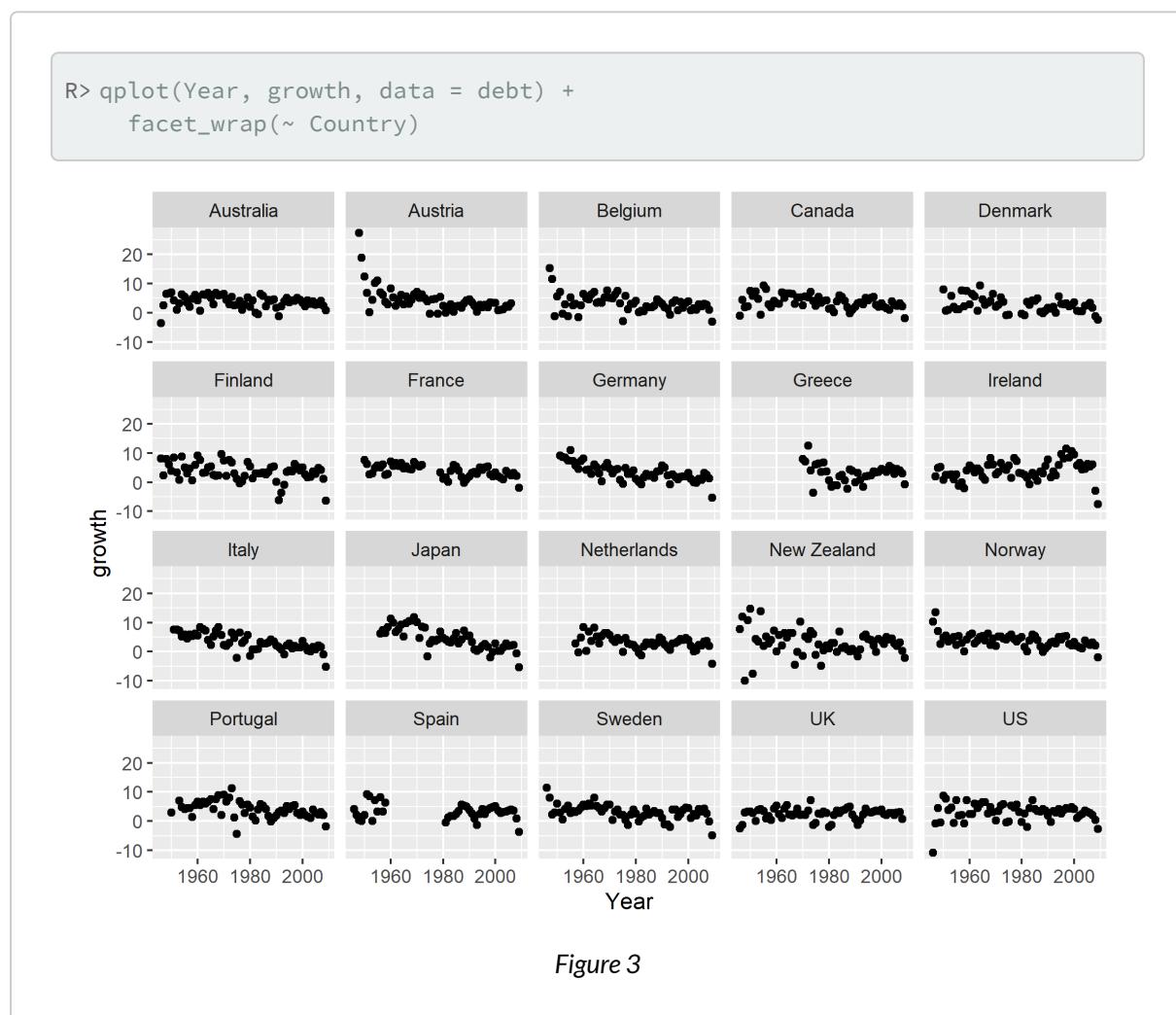
## Visualisation par «petits multiples»

Cherchons désormais à mieux comprendre les variations du taux de croissance du PIB au fil des années.

Dans les graphiques précédents, on voit clairement que ce taux est très variable dans l'immédiat après-guerre, puis qu'il oscille entre environ -5 % et +15 %, puis qu'il semble chuter dramatiquement à la fin des années 2000, marquées par la crise financière globale. Mais comment visualiser ces variations pour chacun des vingt pays de l'échantillon ?

On va ici utiliser le principe de la visualisation par «[petits multiples](#)», c'est-à-dire que l'on va reproduire le même graphique pour chacun des pays, et visualiser l'ensemble de ces graphiques dans une même fenêtre. Concrètement, il va donc s'agir de montrer la croissance annuelle du PIB en faisant apparaître chaque pays dans une facette différente du graphique.

[ggplot2](#) permet d'effectuer cette opération en rajoutant au graphique précédent, au moyen de l'opérateur `+`, l'élément `facet_wrap(~ Country)` au graphique et qui signifie «construire le graphique pour chaque valeur différente de la variable `Country`». On notera que la fonction `facet_wrap` utilise la syntaxe équation, page 929 de **R**. Par défaut, ces «facettes» sont classées par ordre alphabétique :



Voilà qui est beaucoup plus clair ! On aperçoit bien, dans ce graphique, les variations très importantes de croissance du PIB dans un pays comme l'Autriche, ruinée après la Seconde guerre mondiale, ou l'Irlande,

très durement frappée par la crise financière globale en 2008 et 2009. On aperçoit aussi où se trouvent les données manquantes : voir le graphique de l'Espagne, par exemple.

Il faut noter ici un élément essentiel de la grammaire graphique de **ggplot2**, qui utilise une syntaxe additive, où différents éléments et paramètres graphiques peuvent être combinés en les additionnant, ce qui permet de construire et de modifier des graphiques de manière cumulative, pas à pas. Cette caractéristique permet de tâtonner, et de construire progressivement des graphiques très complets.

## Visualisation en séries temporelles

Enfin, pour produire le même graphique que ci-dessus en utilisant des lignes plutôt que des points, il suffit d'utiliser l'argument `geom = "line"`, ce qui peut être considéré comme une meilleure manière de visualiser des séries temporelles, mais qui tend aussi à rendre plus difficile la détection des périodes pour lesquelles il manque des données (voir, à nouveau, le graphique pour l'Espagne) :

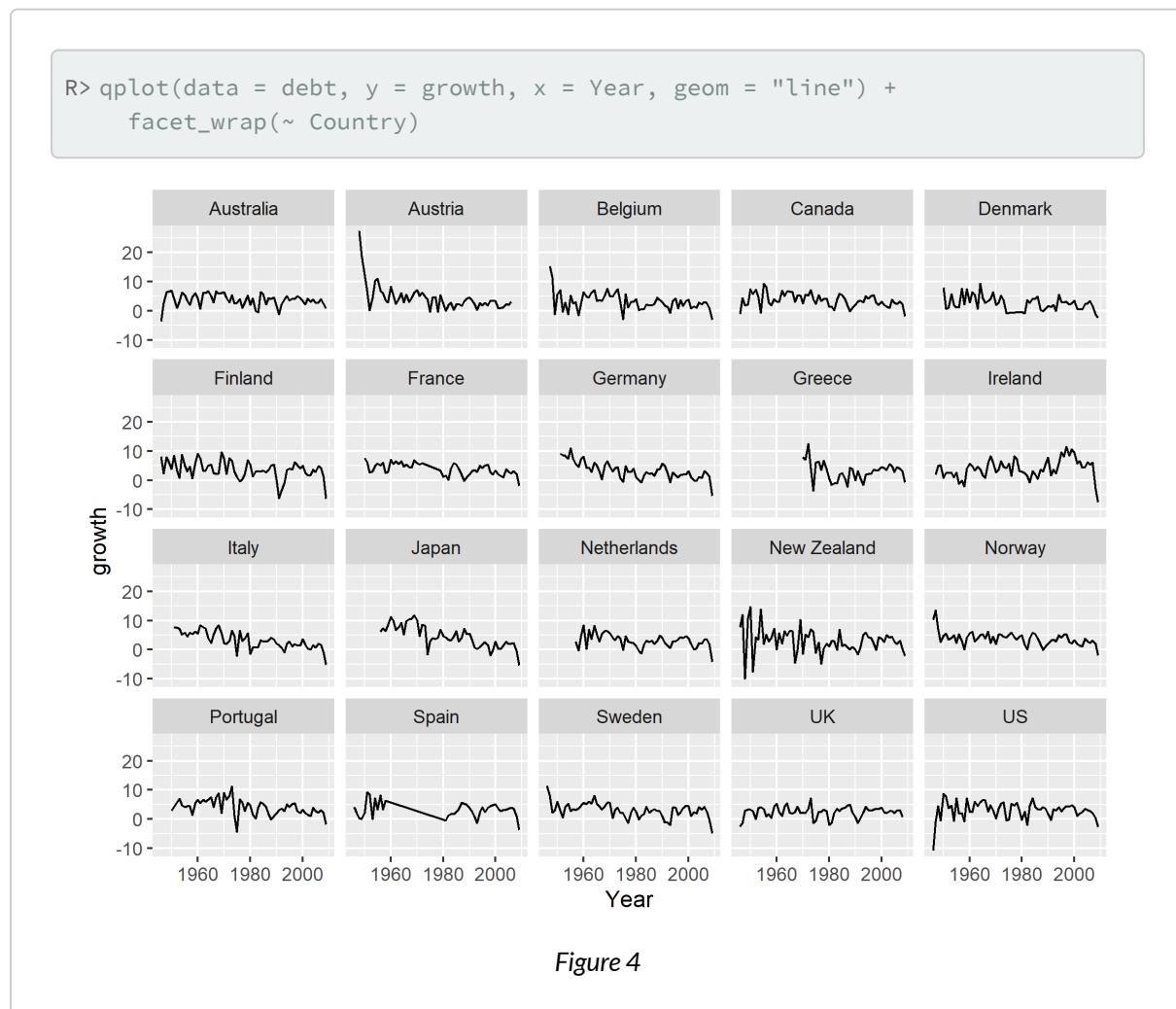


Figure 4

Dans ce dernier exemple, on a défini l'axe `y` avant de définir l'axe `x`, en écrivant ces arguments de manière explicite ; de même, on a commencé par spécifier l'argument `data`, et l'on a terminé par l'argument `geom`. Cet ordre d'écriture permet de conserver une forme de cohérence dans l'écriture des fonctions graphiques.

## Combinaisons d'éléments graphiques

On n'a pas encore visualisé le ratio «Dette publique / PIB», l'autre variable du raisonnement de Reinhart et Rogoff. C'est l'occasion de voir comment rajouter des titres aux axes des graphiques, et d'utiliser les lignes en même temps que des points, toujours grâce à l'argument `geom`, qui peut prendre plusieurs valeurs (ici, "`point`" produit les points et "`line`" produit les lignes) :

```
R> qplot(data = debt, y = ratio, x = Year, geom = c("line", "point")) +
  facet_wrap(~ Country) +
  labs(x = NULL,
       y = "Ratio dette publique / produit intérieur brut (%)\\n")
```

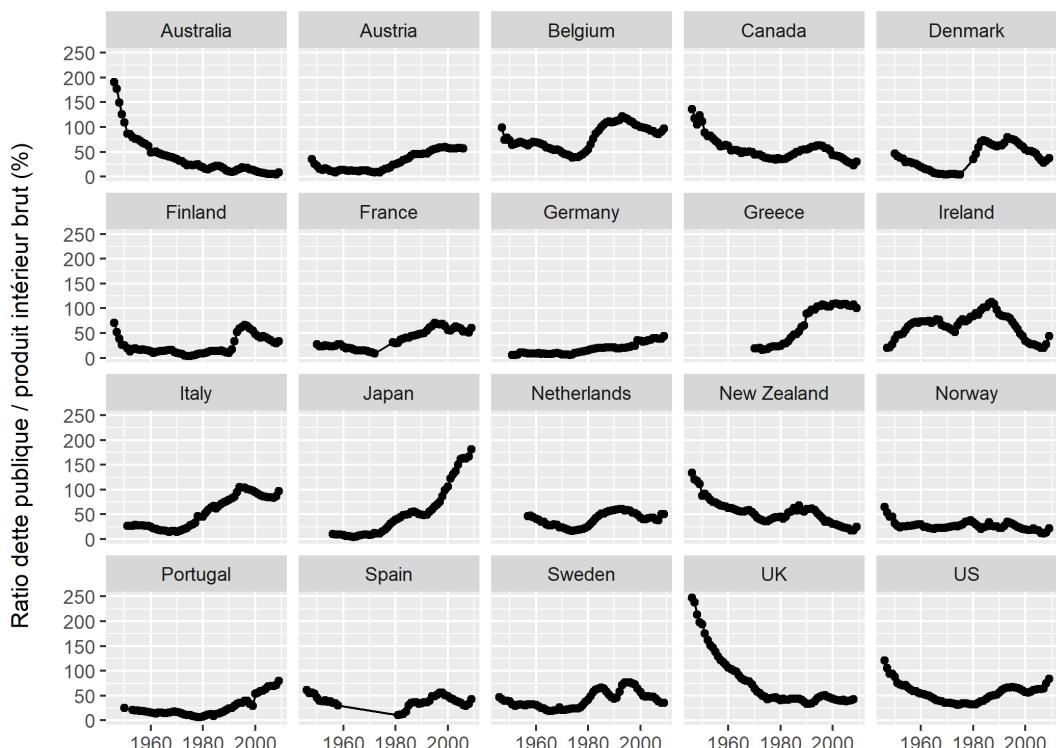


Figure 5

Dans ce graphique, on a combiné deux «objets géométriques» (`geom`) pour afficher à la fois des points et

des lignes. On a ensuite défini les titres des axes, en supprimant celui de l'axe `x`, et en rajoutant un peu d'espace entre le titre de l'axe `y` et l'axe lui-même grâce à la chaîne de caractères finale `\n`, qui rajoute une ligne vide entre ces deux éléments<sup>4</sup>.

Les différents exemples vus dans cette section montrent qu'il va falloir apprendre un minimum de syntaxe graphique pour parvenir à produire des graphiques avec `ggplot2`. Ce petit investissement permet de savoir très vite produire de très nombreux types de graphiques, assez élégants de surcroît, et très facilement modifiables à l'aide de toutes sortes de paramètres optionnels.

#### IMPORTANT

Aussi élégants que soient vos graphiques, il ne vous dispense évidemment pas de réfléchir à ce que vous êtes en train de visualiser, un graphique très élégant pouvant naturellement être complètement erroné, en particulier si les données de base du graphique ont été mal mesurées... ou endommagées.

## Composition graphique avec ggplot2

La section précédente a montré comment utiliser la fonction `qplot` (*quick plot*). La syntaxe complète de l'extension `ggplot2` passe par une autre fonction, `ggplot`, qui permet de mieux comprendre les différents éléments de sa grammaire graphique. Dans cette section, on va détailler cette syntaxe pour en tirer un graphique plus complexe que les précédents.

Commençons par créer un «treillis de base» au graphique :

```
R> p <- ggplot(data = debt, aes(y = growth, x = ratio))
```

Aucun graphique ne s'affiche ici : en effet, ce que l'on a stocké, dans l'objet `p`, n'est pas un graphique complet, mais une base de travail. Cette base définit les coordonnées `x` et `y` du graphique dans l'argument `aes` (*aesthetics*). Ici, on a choisi de mettre la variable dépendante de Reinhart et Rogoff, `growth` (le taux de croissance du PIB), sur l'axe `y`, et la variable indépendante `ratio` (le ratio «Dette publique / PIB») sur l'axe `x`.

Rajoutons désormais un objet géométrique, `geom_point`, qui va projeter, sur le graphique, des points aux coordonnées précédemment définies, et divisons le graphique par un «petit multiple», en projetant les points de chaque décennie dans une facette différente du graphique. Ce graphique propose une décomposition temporelle de la relation étudiée par Reinhart et Rogoff :

---

4. Plus précisément, cela introduit un retour à la ligne dans le titre de l'axe.

```
R> p + geom_point() +  
  facet_grid(. ~ Decade)
```

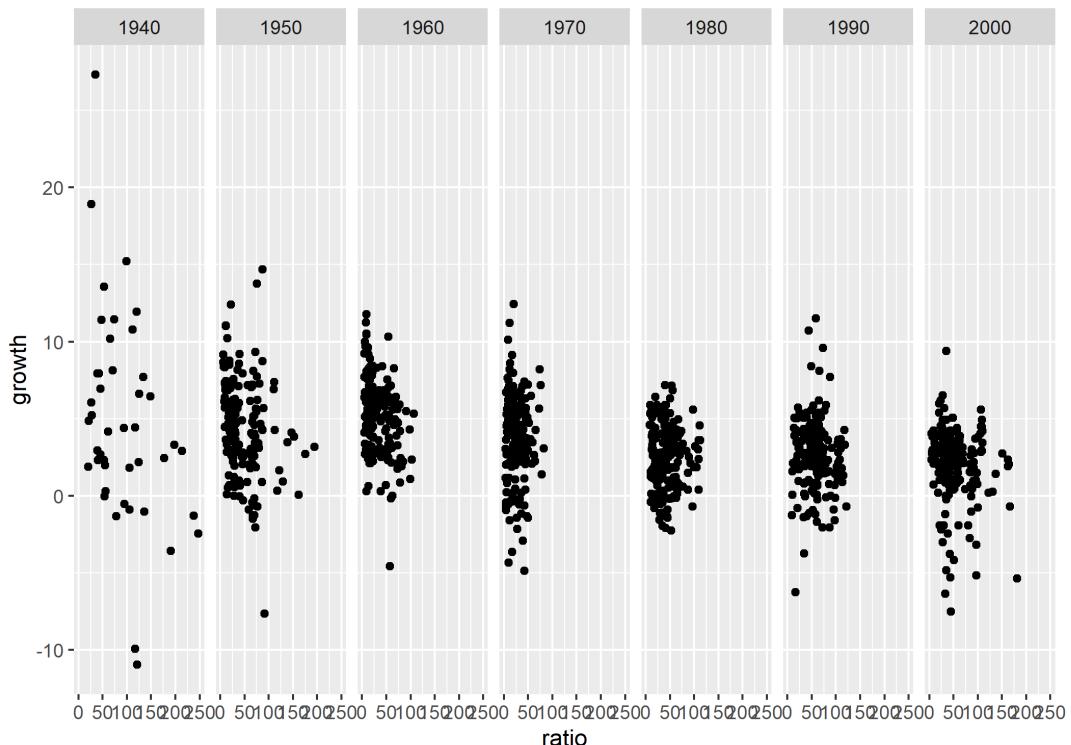


Figure 6

#### NOTE

Le paramètre `facet_grid`, qui utilise aussi la syntaxe «équation, page 929», permet de créer des facettes plus compliquées que celles créées par le paramètre `facet_wrap`, même si, dans nos exemples, on aurait pu utiliser aussi bien l’un que l’autre.

Le graphique ci-dessus présente un problème fréquent : l’axe horizontal du graphique, très important puisque Reinhart et Rogoff évoquent un seuil «fatidique», pour la croissance, de 90% du PIB, est illisible. Grâce à l’argument `scale_x_continuous`, on va pouvoir clarifier cet axe en n’y faisant figurer que certaines valeurs :

```
R> p + geom_point() +
  facet_grid(. ~ Decade) +
  scale_x_continuous(breaks = seq(0, 200, by = 100))
```

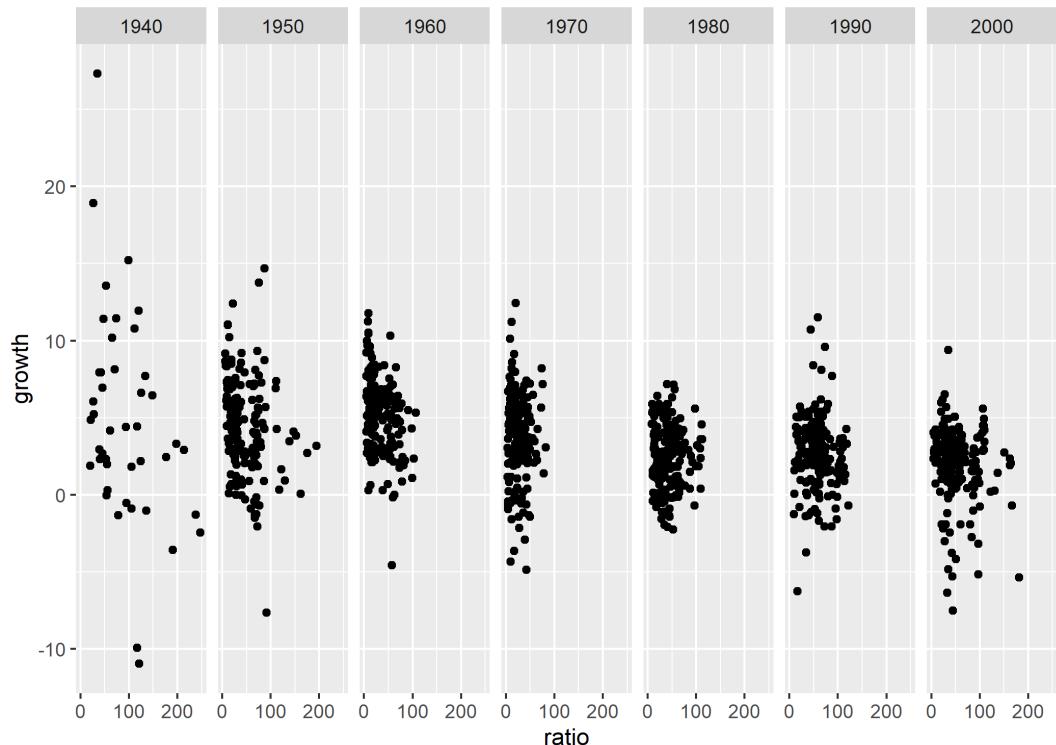


Figure 7

Ces réglages nous conviennent : on va donc les sauvegarder dans l'objet `p`, de manière à continuer de construire notre graphique en incluant ces différents éléments.

```
R> p <- p + geom_point() +
  facet_grid(. ~ Decade) +
  scale_x_continuous(breaks = seq(0, 200, by = 100))
```

## Couleurs et échelles

Abordons désormais un élément-clé de `ggplot2` : la manipulation des paramètres esthétiques. Précédemment, on n'a montré que deux de ces paramètres : `x` et `y`, les coordonnées du graphique. Mais ces paramètres peuvent aussi influencer la couleur des points de notre graphique comme le montre l'exemple suivant :

```
R> p + aes(color = ratio < 90)
```

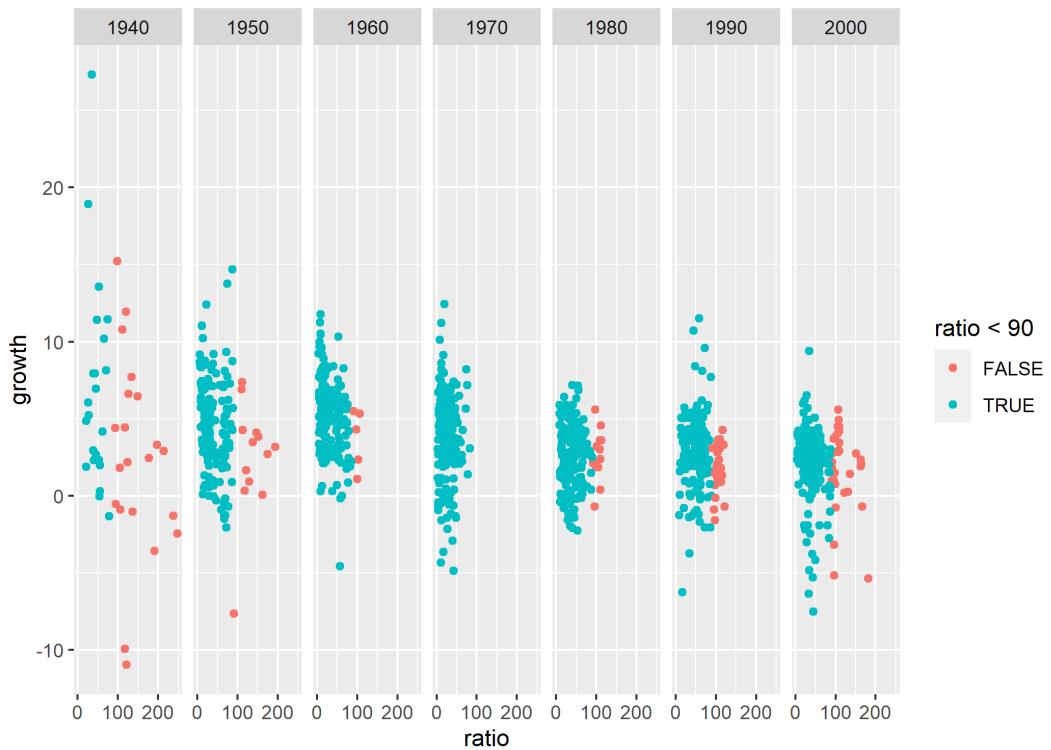


Figure 8

Qu'a-t-on fait ici ? On a rajouté, au graphique stocké dans `p`, un paramètre esthétique qui détermine la couleur de ses points en fonction d'une inégalité, `ratio < 90`, qui est vraie quand le ratio «Dette publique / PIB» est inférieur au seuil «fatidique» de Reinhart et Rogoff, et fausse quand ce ratio dépasse ce seuil. Les couleurs des points correspondent aux couleurs par défaut de `ggplot2`, que l'on peut très facilement modifier avec `scale_colour_brewer` :

```
R> p + aes(color = ratio < 90) +
  scale_colour_brewer(palette = "Set1")
```

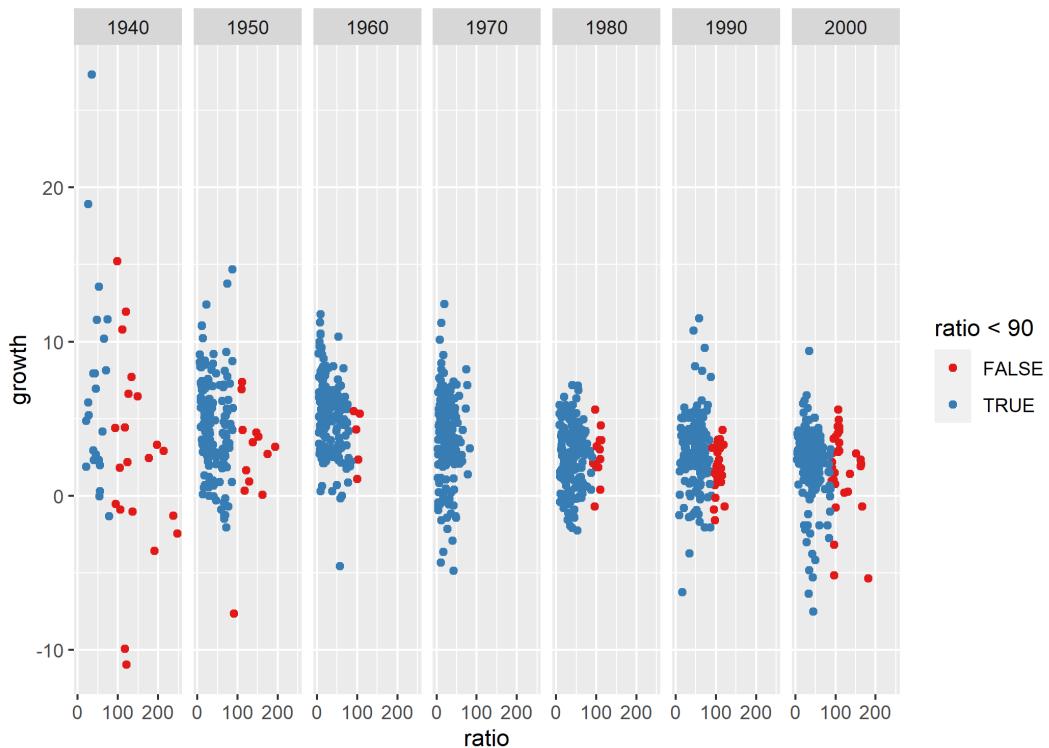


Figure 9

Ici, on a fait appel à la palette de couleur `Set1` de l'éventail de couleurs `ColorBrewer`, qui est automatiquement disponible dans `ggplot2`, et qui est contenu dans l'extension `RColorBrewer`. La palette de couleurs que l'on a choisie affiche les points situés au-dessus du seuil «fatidique» de Reinhart et Rogoff en rouge, les autres en bleu.

Que peut-on dire, à ce stade, du seuil «fatidique» de Reinhart et Rogoff ? On peut observer qu'après la Seconde guerre mondiale, de nombreux pays sont déjà endettés au-delà de ce seuil, et dégagent déjà moins de croissance que les autres. Sur la base de cette trajectoire, de nombreux critiques de Reinhart et Rogoff ont fait remarquer que le raisonnement de Reinhart et Rogoff pose en réalité un sérieux problème d'[inversion du rapport causal](#) entre endettement et croissance au cours du temps.

Envisageons une nouvelle modification des paramètres graphiques. La légende du graphique, qui affiche `FALSE` et `TRUE` en fonction de l'inégalité `ratio < 90`, peut être déroutante. Clarifions un peu cette légende en supprimant son titre et en remplaçant les libellés (*labels*) `FALSE` et `TRUE` par leur signification :

```
R> p <- p + aes(color = ratio < 90) +
  scale_color_brewer("", palette = "Set1",
  labels = c("ratio > 90", "ratio < 90"))
```

Dans le bloc de code ci-dessus, on a stocké l'ensemble de nos modifications dans l'objet `p`, sans l'afficher ; en effet, on souhaite encore procéder à une dernière modification, en rajoutant une [régression locale](#) à travers les points de chaque facette<sup>5</sup>. Après consultation de la documentation de [ggplot2 ici](#) et [là](#), on en arrive au code ci-dessous, où `p` produit le graphique précédent et `geom_smooth` produit la régression locale :

```
R> p + geom_smooth(method = "loess", se = FALSE,
  size = 1, color = "black")
```

```
`geom_smooth()` using formula 'y ~ x'
```

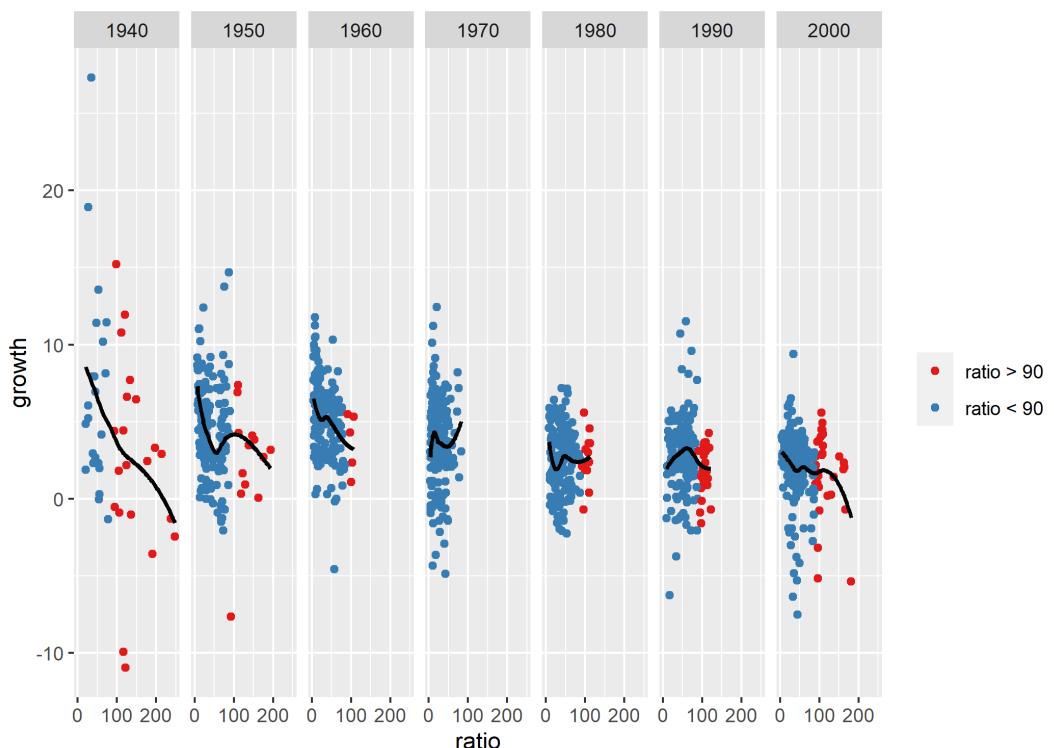


Figure 10

Le graphique permet d'évaluer de manière encore un peu plus précise l'argument de Reinhart et Rogoff, et

5. La régression locale est une variante du calcul de la [moyenne glissante](#) (ou «moyenne mobile») d'une courbe.

en particulier la nature pas si «fatidique» du seuil de 90% du ratio “Dette publique / PIB”, qui sans être une bonne nouvelle pour l’économie, ne détermine pas «fatidiquement» la direction du taux de croissance : si c’était le cas, toutes les courbes du graphique ressembleraient à celles des années 2000. Autrement dit, l’argumentaire de Reinhart et Rogoff laisse clairement à désirer.

## Utilisation des thèmes

Reprendons notre graphique de départ. On va, pour terminer cette démonstration, en construire une version imprimable en noir et blanc, ce qui signifie qu’au lieu d’utiliser des couleurs pour distinguer les points en-deçà et au-delà du seuil «fatidique» de Reinhart et Rogoff, on va utiliser une ligne verticale, produite par `geom_vline` et affichée en pointillés par le paramètre `lty` (*linetype*) :

```
R> ggplot(data = debt, aes(y = growth, x = ratio)) +
  geom_point(color = "grey50") +
  geom_vline(xintercept = 90, lty = "dotted") +
  geom_smooth(method = "loess", size = 1, color = "black", se = FALSE) +
  scale_x_continuous(breaks = seq(0, 200, by = 100)) +
  facet_grid(. ~ Decade) +
  labs(y = "Taux de croissance du produit intérieur brut\n",
       x = "\nRatio dette publique / produit intérieur brut (%)",
       title = "Données Reinhart et Rogoff corrigées, 1946-2009\n") +
  theme_bw() +
  theme(strip.background = element_rect(fill = "grey90", color = "grey50"),
        strip.text = element_text(size = rel(1)),
        panel.grid = element_blank())
```

```
`geom_smooth()` using formula 'y ~ x'
```

Données Reinhart et Rogoff corrigées, 1946-2009

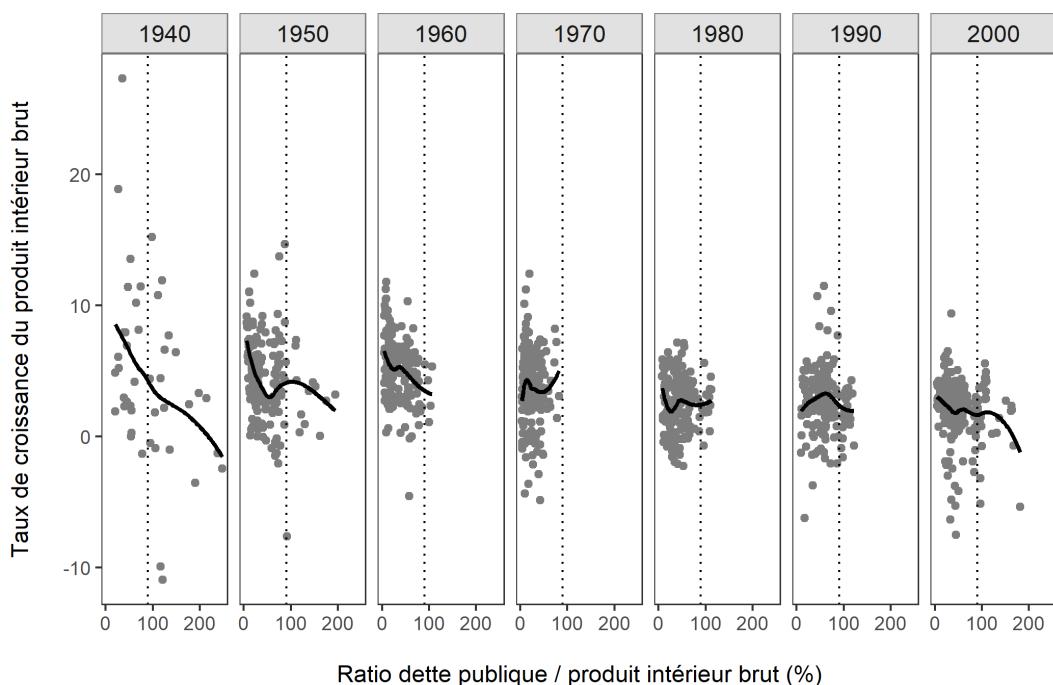


Figure 11

Ce graphique utilise tous les éléments présentés dans ce chapitre, ainsi qu'une dernière nouveauté : l'utilisation d'un thème graphique différent du thème par défaut de [ggplot2](#). Le thème par défaut, qui

s'appelle `theme_grey`, est ici remplacé par un thème moins chargé, `theme_bw` ("black and white"), que l'on a modifié en y rajoutant quelques paramètres supplémentaires :

- le paramètre `strip.background` détermine la couleur du rectangle contenant les titres des facettes, c'est-à-dire les décennies observées ;
- le paramètre `strip.text` détermine la taille des titres des facettes, qui sont ici affichés dans la même taille de texte que le reste du texte ;
- et le paramètre `panel.grid` supprime ici les guides du graphique grâce à l'élément vide `element_blank`, de manière à en alléger la lecture.

Ces différents réglages peuvent être sauvegardés de manière à créer des thèmes réutilisables, comme ceux de l'extension `ggthemes`, ce qui permet par exemple de créer un thème entièrement blanc dans lequel on peut ensuite projeter une carte, ou de produire une série de graphiques homogènes d'un point de vue esthétique.

## Export des graphiques

Les graphiques produits par `ggplot2` peuvent être sauvegardés manuellement, comme expliqué dans le chapitre «Export des graphiques, page 331», ou programmatiquement. Pour sauvegarder le dernier graphique affiché par `ggplot2` au format PNG, il suffit d'utiliser la fonction `ggsave`, qui permet d'en régler la taille (en pouces) et la résolution (en pixels par pouce ; 72 par défaut) :

```
R> ggsave("reinhart-rogoff.png", width = 11, height = 8)
```

De la même manière, pour sauvegarder n'importe quel graphique construit avec `ggplot2` et stocké dans un objet, il suffit de préciser le nom de cet objet, comme ci-dessous, où l'on sauvegarde le graphique contenu dans l'objet `p` au format vectoriel PDF, qui préserve la netteté du texte et des autres éléments du graphique à n'importe quelle résolution d'affichage :

```
R> ggsave("reinhart-rogoff.pdf", plot = p,
           width = 11, height = 8)
```

## Pour aller plus loin

Ce chapitre n'a pu faire la démonstration que d'une infime partie des manières d'utiliser `ggplot2`. En voici une dernière illustration, qui donne une idée des différents types de graphiques que l'extension permet de produire dès que l'on connaît les principaux éléments de sa syntaxe :

```
R> ggplot(data = debt, aes(x = ratio > 90, y = growth)) +
  geom_boxplot() +
  scale_x_discrete(labels = c("< 90", "90+")) +
  facet_grid(. ~ Decade) +
  labs(y = "Taux de croissance du produit intérieur brut\n",
       x = "\nRatio dette publique / produit intérieur brut (%)",
       title = "Données Reinhart et Rogoff corrigées, 1946-2009\n") +
  theme_linedraw() +
  theme(strip.text = element_text(size = rel(1)),
        panel.grid = element_blank())
```

Données Reinhart et Rogoff corrigées, 1946-2009

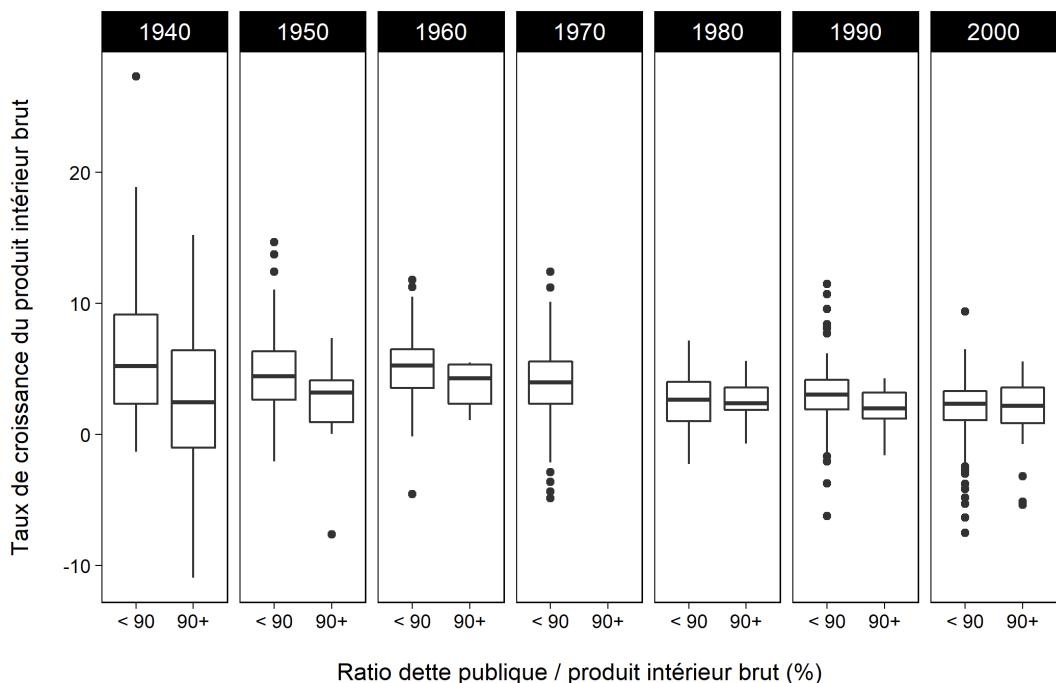


Figure 12

Le code ci-dessus est somme toute très proche du code présenté dans le reste du texte, et en même temps, on a basculé de la visualisation sous forme de série temporelles à une visualisation par boxplots. Ces basculements sont très faciles à envisager dès que l'on maîtrise les principaux éléments de [ggplot2](#), `geom`, `scale` et `facet`, et les paramètres `labs` et `theme` pour effectuer les finitions.

## Ressources essentielles

Pour tout ce qui concerne l'utilisation de `ggplot2`, l'ouvrage de Wickham, en cours d'actualisation, est la ressource essentielle à consulter. L'[ouvrage de Winston Chang](#), qui contient des [dizaines d'exemples](#), le complète utilement, de même que la [documentation en ligne](#) de l'extension. Enfin, le site [StackOverflow](#) contient de très nombreuses questions/réponses sur les subtilités de sa syntaxe.

On trouve aussi très facilement, ailleurs sur Internet, des dizaines de *tutorials* et autres *cheatsheets* pour `ggplot2`, [ici](#) ou [là](#) par exemple.

A noter également une galerie de graphiques sous R avec de très nombreux exemples de graphique `ggplot2` : <http://www.r-graph-gallery.com/portfolio/ggplot2-package/>

## Extensions de `ggplot2`

Il faut signaler, pour terminer, quelques-unes des différentes extensions inspirées de `ggplot2`, dont la plupart sont encore en cours de développement, mais qui permettent d'ores et déjà de produire des centaines de types de graphiques différents, à partir d'une syntaxe graphique proche de celle présentée dans ce chapitre :

- l'extension `ggfortify` permet de visualiser les résultats de différentes fonctions de modélisation avec `ggplot2` ;
- l'extension `ggmap` permet de visualiser des fonds de carte et d'y superposer des éléments graphiques rédigés avec `ggplot2` ;
- l'extension `GGally` rajoute quelques types de graphiques à ceux que `ggplot2` peut produire par défaut ;
- et des extensions comme `ggvis` permettent de produire des graphiques interactifs en utilisant la syntaxe de base de `ggplot2`.



# Graphiques univariés et bivariés avec ggplot2

Retour sur les bases de ggplot2 . . . . .	418
Histogramme . . . . .	420
Densité et répartition cumulée . . . . .	423
Boîtes à moustaches (et représentations associées) . . . . .	426
Diagramme en bâtons . . . . .	431
Nuage de points . . . . .	432
Matrice de nuages de points et matrice de corrélation . . . . .	438
Estimation locale de densité (et représentations associées) . . . . .	441
Diagramme de Cleveland . . . . .	447
Diagrammes en barres . . . . .	450
Comparer plusieurs variables discrètes et/ou continues . . . . .	459
Graphe en mosaïque . . . . .	462
Données labellisées et ggplot2 . . . . .	463
Exporter les graphiques obtenus . . . . .	464

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #03 (statistiques descriptives avec gtsummary et esquisse) sur [YouTube](#).

Après avoir introduit l'extension **ggplot2** au travers d'une étude de cas, page 395, nous reprenons ici les graphiques produits dans les chapitres statistique univariée, page 339 et statistique bivariée, page 365 et montrons comment les réaliser avec **ggplot2**.

## Retour sur les bases de ggplot2

L'extension `ggplot2` nécessite que les données du graphique soient sous la forme d'un tableau de données (`data.frame`) avec une ligne par observation et les différentes valeurs à représenter sous forme de variables du tableau.

Tous les graphiques avec `ggplot2` suivent une même logique. En **premier** lieu, on appellera la fonction `ggplot` en lui passant en paramètre le fichier de données.

`ggplot2` nomme *esthétiques* les différentes propriétés visuelles d'un graphique, à savoir l'axe des x (`x`), celui des y (`y`), la couleur des lignes (`colour`), celle de remplissage des polygones (`fill`), le type de lignes (`linetype`), etc. Une représentation graphique consiste donc à représenter chacune de nos variables d'intérêt selon une esthétique donnée. En **second** lieu, on appellera donc la fonction `aes` pour indiquer la correspondance entre les variables de notre fichier de données et les esthétiques du graphique.

A minima, il est nécessaire d'indiquer en **troisième** lieu une *géométrie*, autrement dit la manière dont les éléments seront représentés visuellement. À chaque géométrie corresponds une fonction commençant par `geom_`, par exemple `geom_point` pour dessiner des points, `geom_line` pour des lignes, `geom_bar` pour des barres ou encore `geom_area` pour des aires. Il existe de nombreuses géométries différentes, chacune prenant en compte certaines esthétiques, certaines étant requises pour cette géométrie et d'autres optionnelles. La liste des esthétiques prises en compte par chaque géométrie en indiquée dans l'aide en ligne de cette dernière.

Pour un document récapitulant les principales géométries et options de `ggplot2`, on pourra se référer à la *Cheat Sheet* officielle disponible à <https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf>. Une version en français est disponible à l'adresse <http://thinkr.fr/pdf/ggplot2-french-cheatsheet.pdf>.

`ggplot2` reposant sur une syntaxe additive, la syntaxe de base d'un graphique sera donc de la forme :

```
R> ggplot(data) + aes(x = Var1, fill = Var2) + geom_bar()
```

De manière alternative, on peut également indiquer la correspondance entre variables et esthétiques comme deuxième argument de la fonction `ggplot`. Les deux syntaxes sont équivalentes.

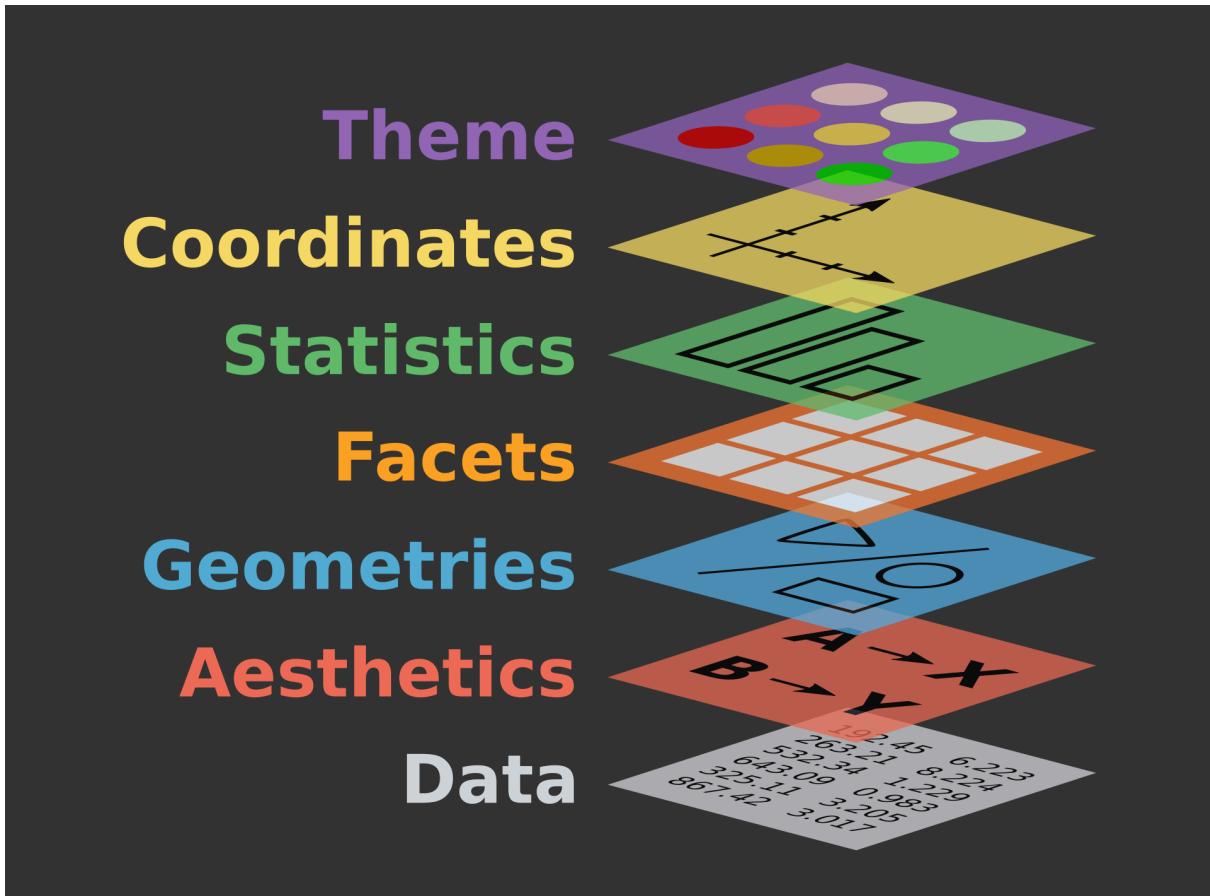
```
R> ggplot(data, aes(x = Var1, fill = Var2)) + geom_bar()
```

Il est ensuite possible de personnaliser de nombreux éléments d'un graphique et notamment :

- les étiquettes ou *labs* (titre, axes, légendes) avec `ggtitle`, `xlab`, `ylab` ou encore la fonction plus générique `labs` ;
- les échelles (*scales*) des différentes esthétiques avec les fonctions commençant par `scale_` ;

- les *facettes* (`facets`) avec les fonctions commençant par `facet_` ;
- le système de *coordonnées* avec les fonctions commençant par `coord_` ;
- la *légende* (`guides`) avec les fonctions commençant par `guide_` ;
- le *thème* du graphiques (mise en forme des différents éléments) avec `theme` .

Ces différents éléments seront abordés plus en détails dans le [chapitre avancé sur ggplot2](#). Dans la suite de ce chapitre, nous nous focaliserons sur les graphiques et options de base.



Grammaire des graphiques de ggplot2

Préparons les données des exemples et chargeons `ggplot2` :

```
R> library(questionr)
library(ggplot2)
data("hdv2003")
d <- hdv2003
```

## Histogramme

Pour un **histogramme**, on aura recours à la géométrie `geom_histogram`. Si l'on a une observation par ligne dans le fichier de données, l'histogramme s'obtient simplement en associant la variable d'intérêt à l'esthétique `x`. Notez la syntaxe de `aes` : le nom des variables est indiqué directement, sans guillemets.

```
R> ggplot(d) +
  aes(x = heures.tv) +
  geom_histogram() +
  ggtitle("Nombres d'heures passées devant la télévision") +
  xlab("Heures") +
  ylab("Effectifs")
```

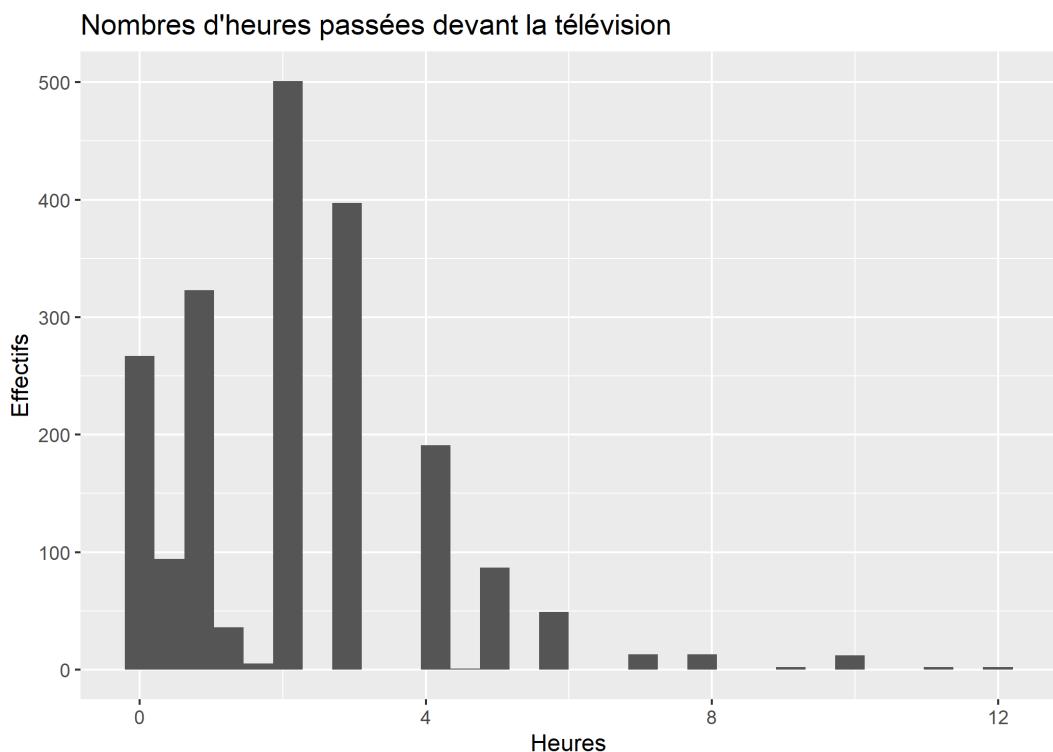


Figure 1. Un histogramme

On peut personnaliser la couleur de remplissage des rectangles en indiquant une valeur fixe pour l'esthétique `fill` dans l'appel de `geom_histogram` (et non via la fonction `aes` puisqu'il ne s'agit pas d'une variable du tableau de données). L'esthétique `colour` permet de spécifier la couleur du trait des rectangles. Enfin, le paramètre `binwidth` permet de spécifier la largeur des barres.

```
R> ggplot(d) +  
  aes(x = heures.tv) +  
  geom_histogram(fill = "orange", colour = "black", binwidth = 2) +  
  ggtitle("Nombres d'heures passées devant la télévision") +  
  xlab("Heures") +  
  ylab("Effectifs")
```

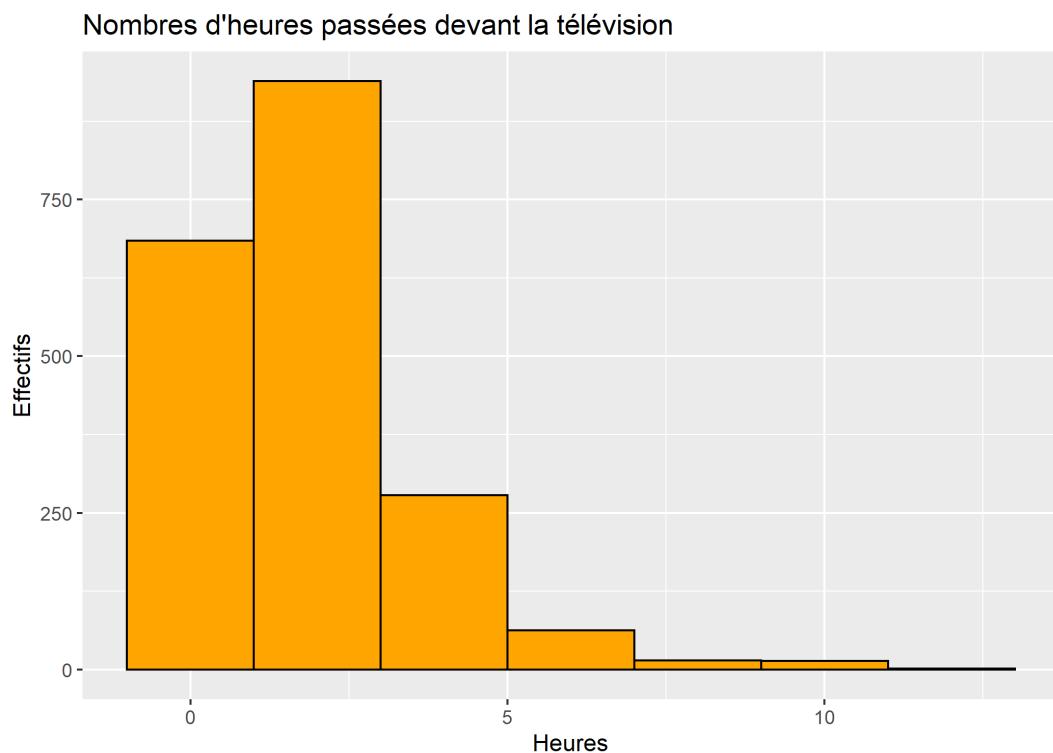


Figure 2. Un histogramme en couleur

Comme avec la fonction `hist`, on peut personnaliser les classes avec l'argument `breaks`.

```
R> ggplot(d) +
  aes(x = heures.tv) +
  geom_histogram(fill ="orange", colour = "black", breaks = c(0, 1, 4,
9, 12)) +
  ggtitle("Nombres d'heures passées devant la télévision") +
  xlab("Heures") +
  ylab("Effectifs")
```

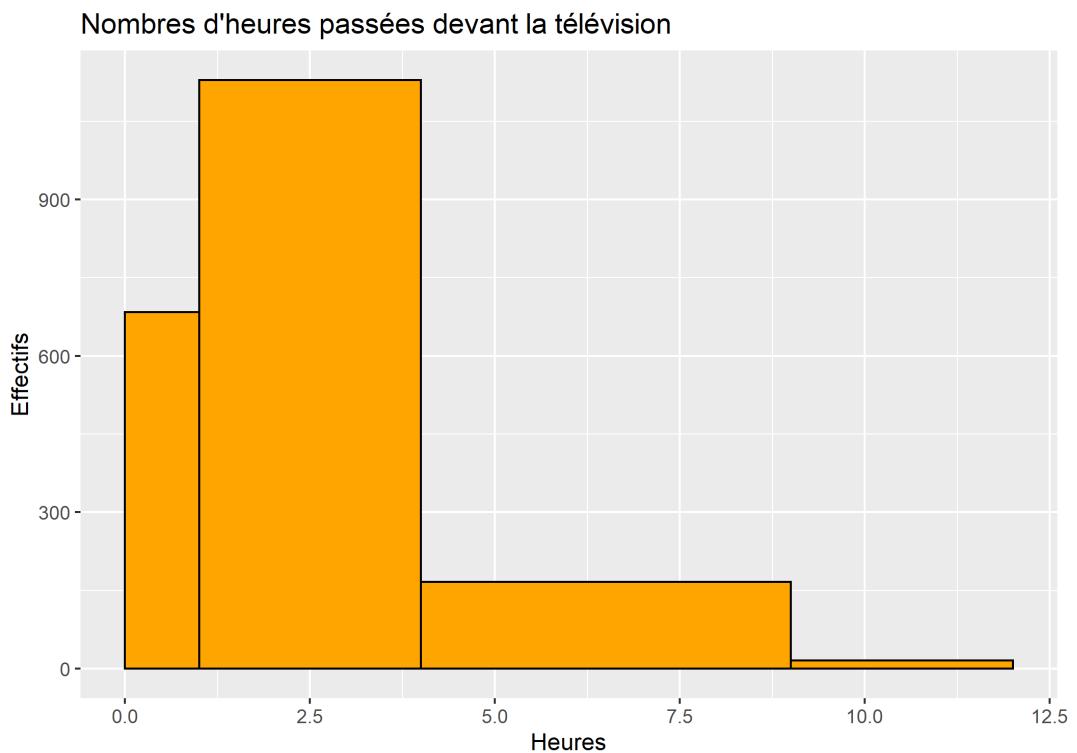


Figure 3. Un histogramme avec classes personnalisées

On peut ajouter à l'axe des x des tirets représentant la position des observations à l'aide de `geom_rug`.

```
R> ggplot(d) +
  aes(x = heures.tv) +
  geom_histogram(fill = "orange", colour = "black", binwidth = 2) +
  geom_rug() +
  ggtitle("Nombres d'heures passées devant la télévision") +
  xlab("Heures") +
  ylab("Effectifs")
```

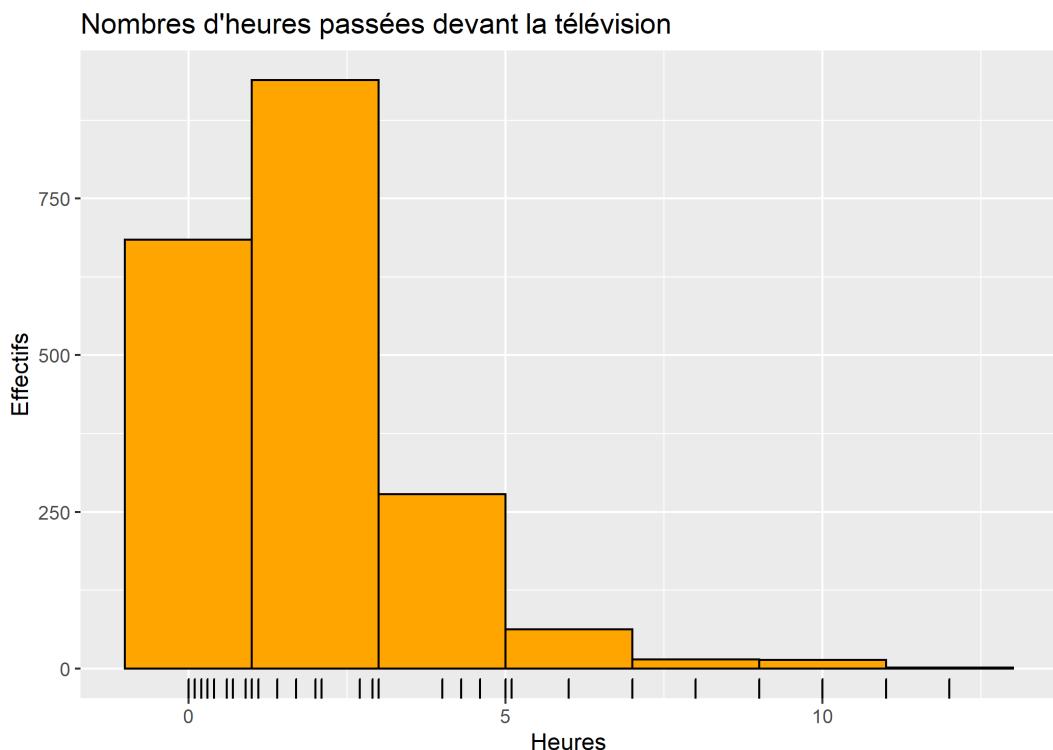


Figure 4. Un histogramme avec `geom_rug()`

## Densité et répartition cumulée

Une courbe de densité s'obtient aisément avec la géométrie `geom_density`.

```
R> ggplot(d) +  
  aes(x = heures.tv) +  
  geom_density() +  
  ggtitle("Nombres d'heures passées devant la télévision") +  
  xlab("Heures") +  
  ylab("Densité")
```

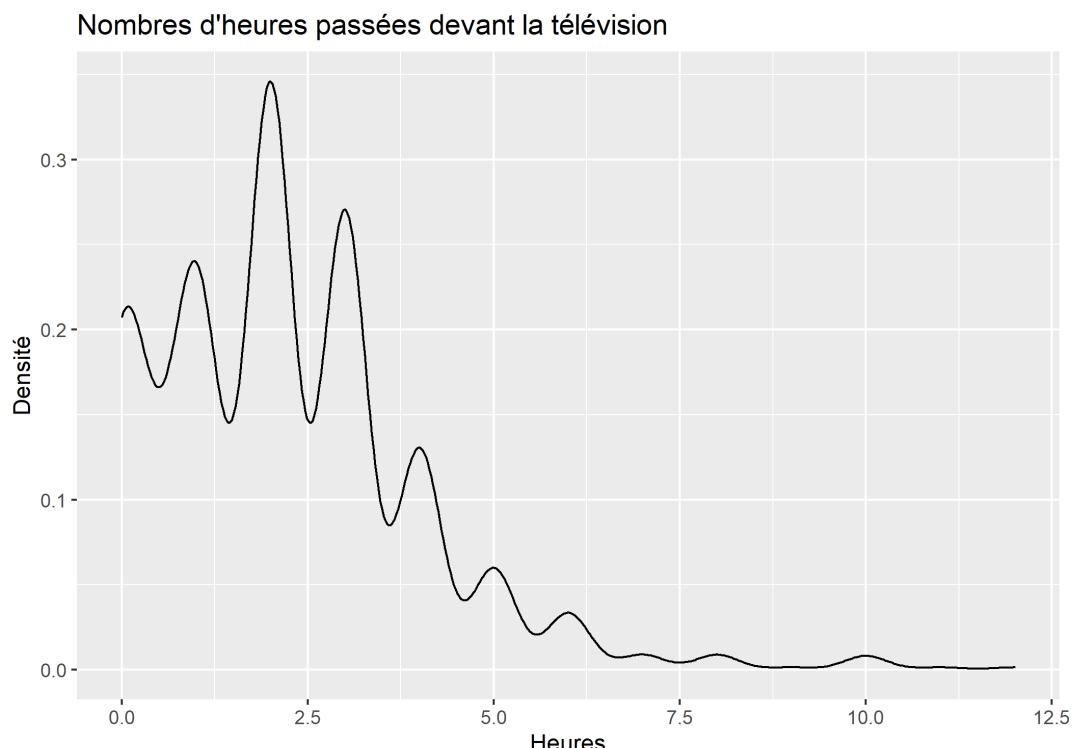


Figure 5. Courbe de densité

On peut personnaliser la fenêtre d'ajustement avec l'argument `adjust`.

```
R> ggplot(d) +
  aes(x = heures.tv) +
  geom_density(adjust = 1.5) +
  ggtitle("Nombres d'heures passées devant la télévision") +
  xlab("Heures") +
  ylab("Densité")
```

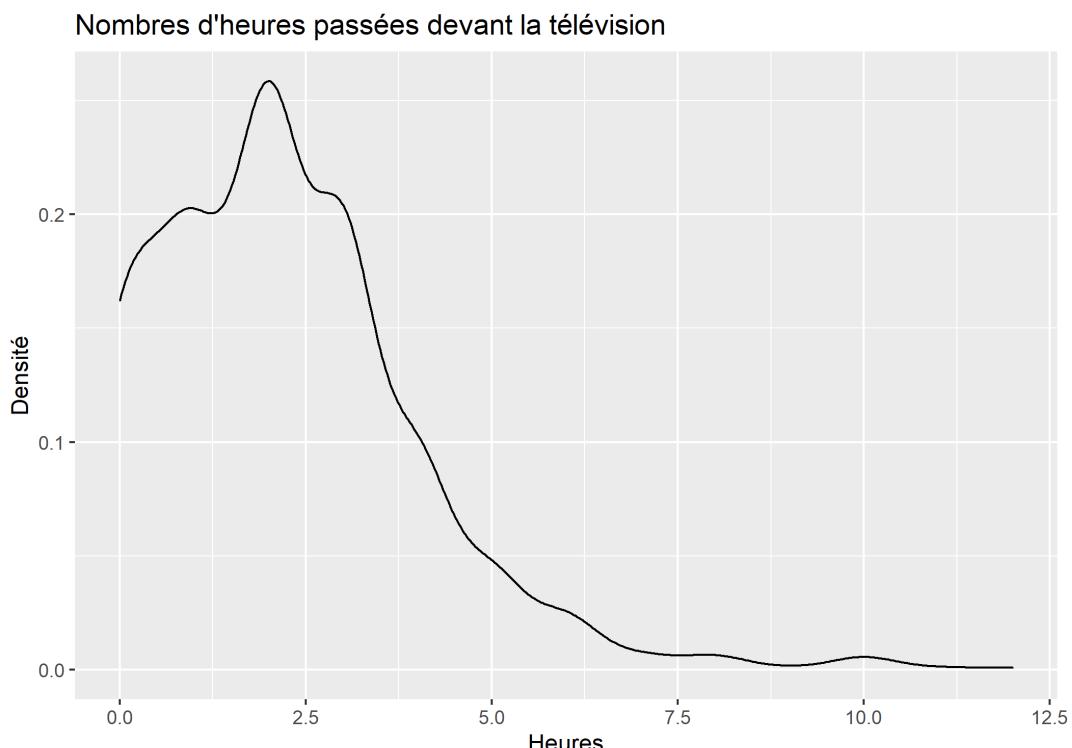


Figure 6. Courbe de densité avec fenêtre d'ajustement personnalisée

Pour la fonction de répartition empirique ou *empirical cumulative distribution function* en anglais, on utilisera la statistique `stat_ecdf`. Au passage, on notera qu'il est possible d'ajouter une couche à un graphique en appelant soit une géométrie, soit une statistique.

```
R> ggplot(d) +  
  aes(x = heures.tv) +  
  stat_ecdf() +  
  xlab("Heures") +  
  ylab("Fonction de répartition cumulée")
```

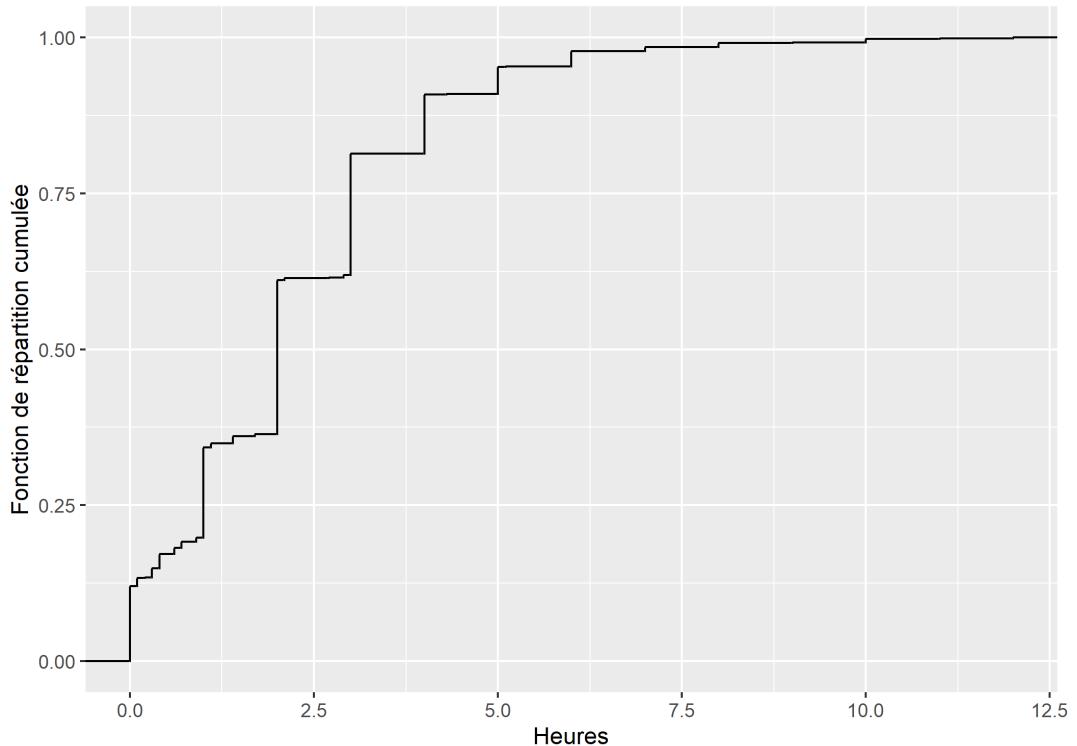
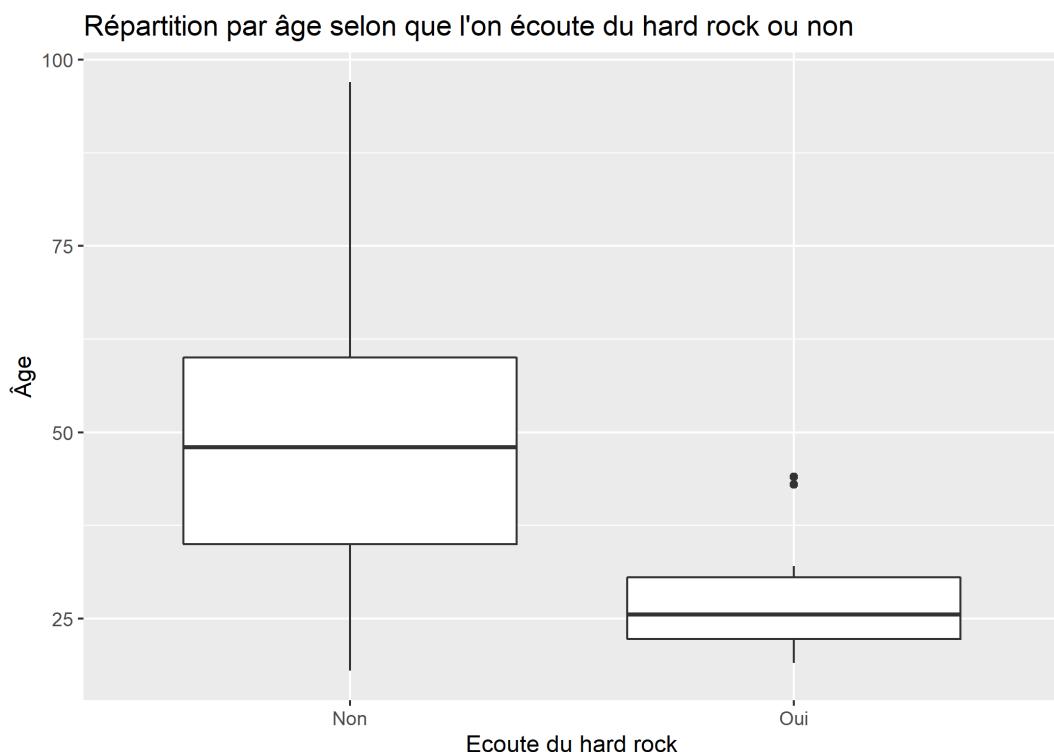


Figure 7. Fonction de répartition empirique cumulée

## Boîtes à moustaches (et représentations associées)

La géométrie `geom_boxplot` nécessite *a minima* deux esthétiques : `x` et `y`. Pour représenter une variable quantitative selon une variable catégorielle, on fera donc :

```
R> ggplot(d) +
  aes(x = hard.rock, y = age) +
  geom_boxplot() +
  xlab("Ecoute du hard rock") +
  ylab("Âge") +
  ggtitle("Répartition par âge selon que l'on écoute du hard rock ou non")
```



*Figure 8. Boîtes à moustache*

Lorsque l'on souhaite représenter une seule variable quantitative (statistique univariée), on passera alors une constante à l'esthétique `x`.

```
R> ggplot(d) +  
  aes(x = "Nombre d'heures passées devant la télévision", y = heures.tv)  
+  
  geom_boxplot() +  
  xlab("") +  
  ylab("Heures quotidiennes")
```

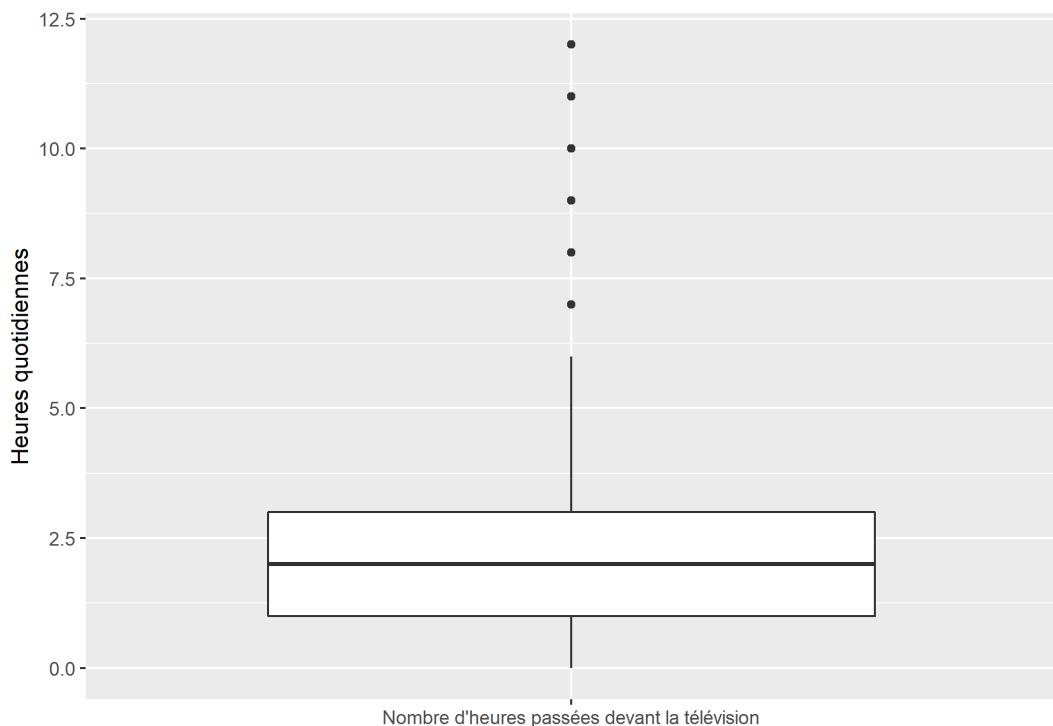


Figure 9. Boîte à moustache (une seule variable)

Une représentation alternative aux boîtes à moustaches ou *boxplots* sont les graphiques en violon ou *violin plots*, qui représentent la densité de distribution. Ils s'obtiennent avec la géométrie `geom_violin`.

```
R> ggplot(d) +
  aes(x = hard.rock, y = age) +
  geom_violin() +
  xlab("Ecoute du hard rock") +
  ylab("Âge") +
  ggtitle("Répartition par âge selon que l'on écoute du hard rock ou non")
```

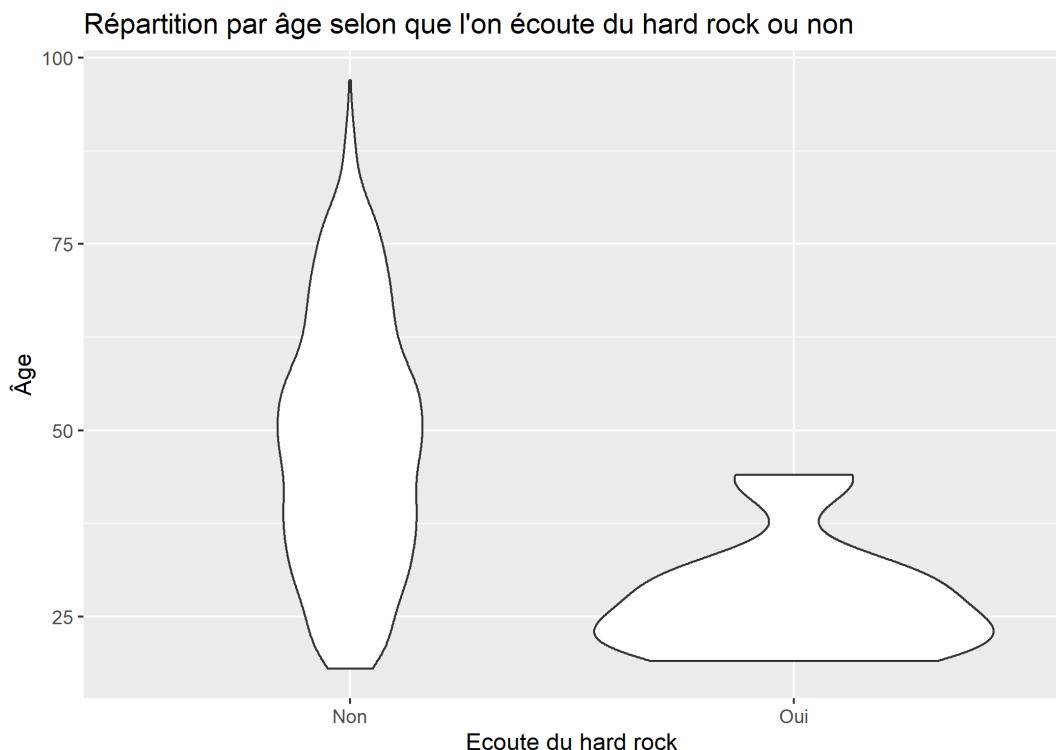


Figure 10. Graphiques en violon ou violin plot

Les boîtes à moustache peuvent parfois être trompeuses car ne représentant qu'imparfairement la distribution d'une variable quantitative. Voir par exemple [The boxplot and its pitfalls](#) sur <https://www.data-to-viz.com>.

Les graphiques de pirates ou *pirateplot* sont une visualisation alternative qui combinent :

- un nuage de points représentant les données brutes ;
- une barre verticale représentant la moyenne ;
- un rectangle traduisant l'intervalle de confiance de cette moyenne ;
- un «violon» indiquant la distribution.

L'extension **ggpirate** fournit une géométrie `geom_pirate` pour **ggplot2**. Cette extension n'étant

disponible que sur **GitHub**, on l'installera avec la commande :

```
R> devtools:::install_github("mikabr/ggpirate")
```

```
R> library(ggpirate)
ggplot(d) +
  aes(x = hard.rock, y = age, colour = hard.rock, fill = hard.rock) +
  geom_pirate() +
  xlab("Ecoute du hard rock") +
  ylab("Âge") +
  ggtitle("Répartition par âge selon que l'on écoute du hard rock ou non") +
  theme_minimal() +
  theme(panel.grid.minor = element_blank())
```

```
No summary function supplied, defaulting to `mean_se()`
No summary function supplied, defaulting to `mean_se()`
```

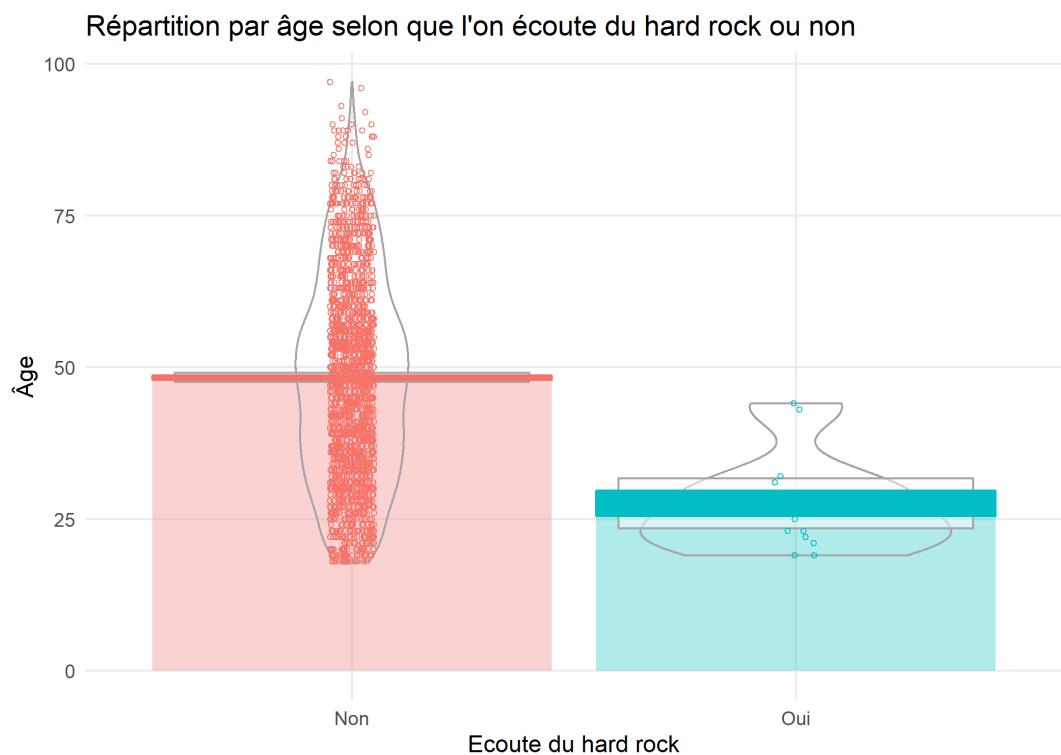


Figure 11. Graphiques de «pirates» ou pirateplot

## Diagramme en bâtons

Un diagramme en bâtons s'obtient avec la géométrie `geom_bar`.

```
R> ggplot(d) +
  aes(x = freres.soeurs) +
  geom_bar() +
  xlab("Nombre de frères et soeurs") +
  ylab("Effectifs")
```

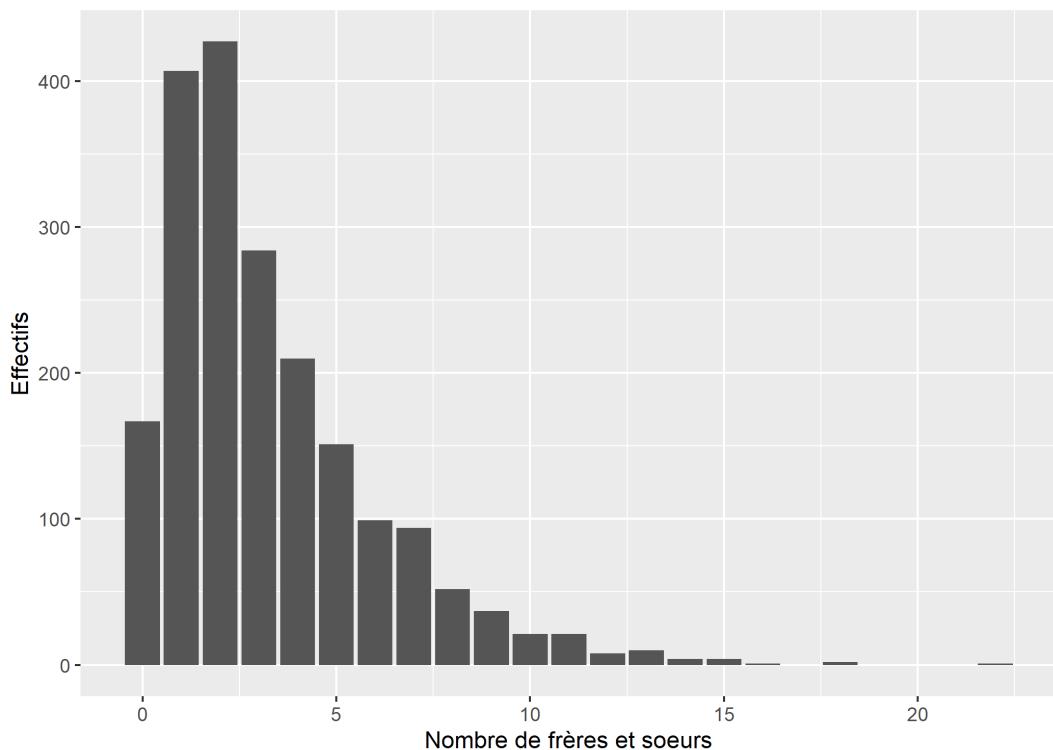


Figure 12. Diagramme en bâtons

La largeur des barres par défaut est de 0,9. Dès lors, le graphique ressemble plus à un histogramme qu'à un diagramme en bâtons. On peut personnaliser ce paramètre avec l'argument `width`.

```
R> ggplot(d) +  
  aes(x = freres.soeurs) +  
  geom_bar(width = .2) +  
  xlab("Nombre de frères et soeurs") +  
  ylab("Effectifs")
```

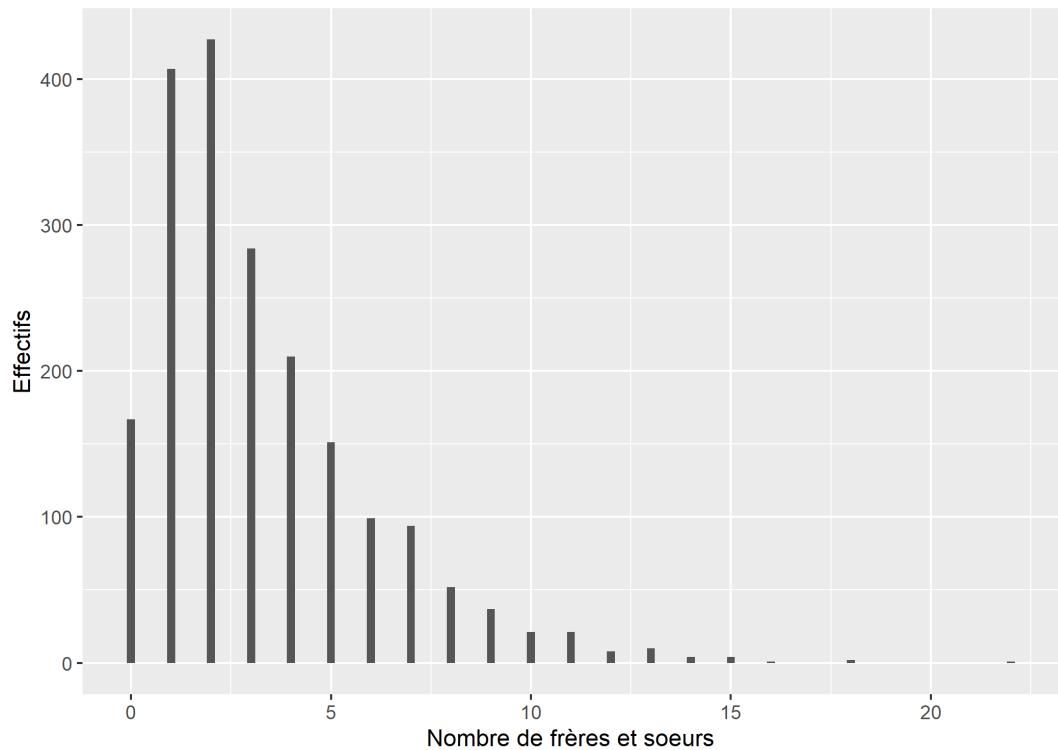


Figure 13. Diagramme en bâtons avec ajustement de la largeur des barres

## Nuage de points

Un nuage de points se représente facilement avec la géométrie `geom_point`.

```
R> ggplot(d) +  
  aes(x = age, y = heures.tv) +  
  geom_point() +  
  xlab("Âge") +  
  ylab("Heures quotidiennes de télévision")
```

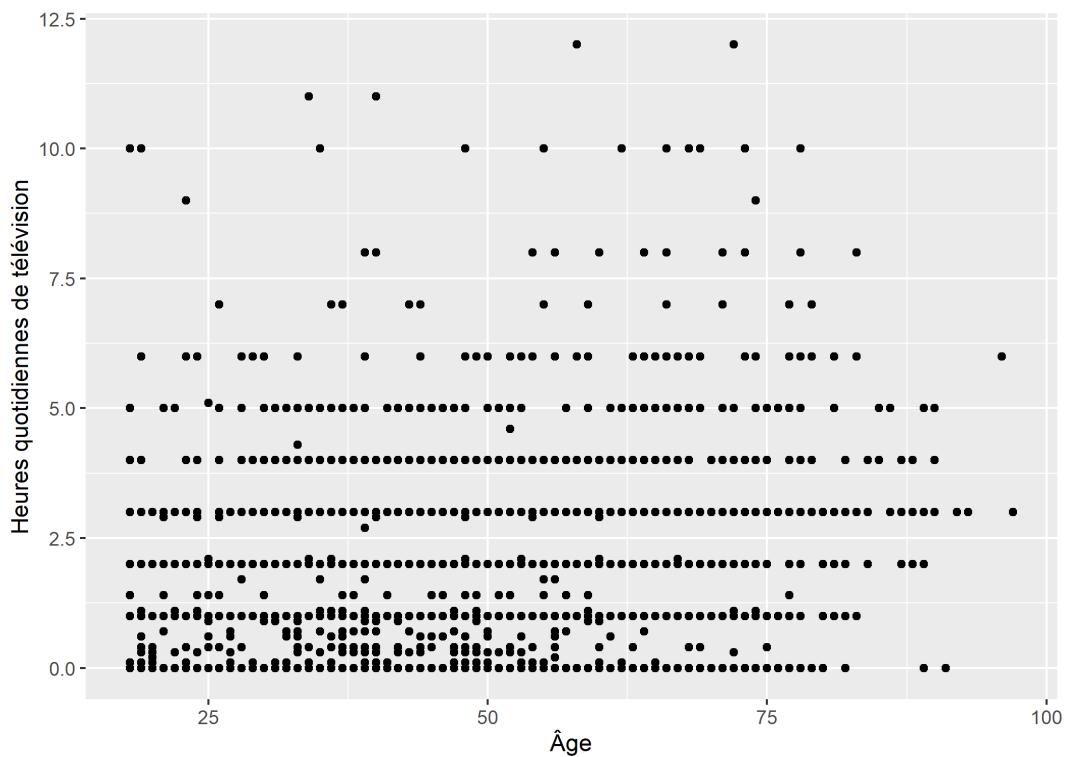


Figure 14. Nuage de points

On pourra personnaliser la couleur des points avec `colour` et le niveau de transparence avec `alpha`.

```
R> ggplot(d) +  
  aes(x = age, y = heures.tv) +  
  geom_point(colour = "red", alpha = .2) +  
  xlab("Âge") +  
  ylab("Heures quotidiennes de télévision")
```

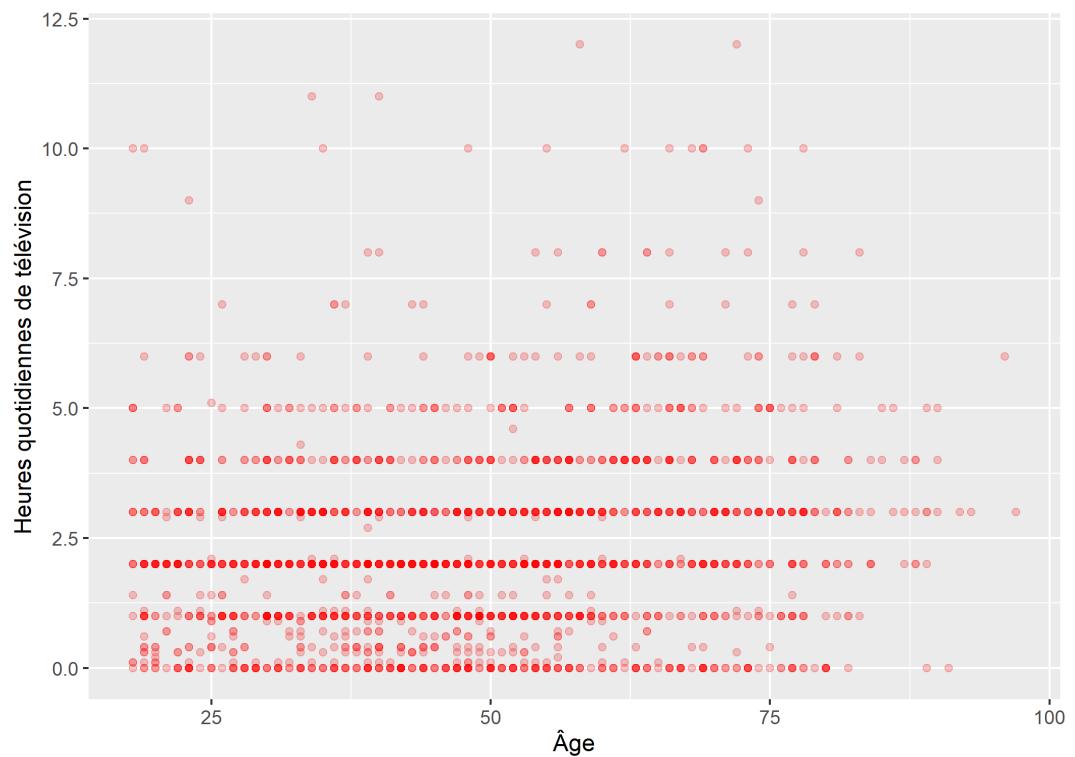
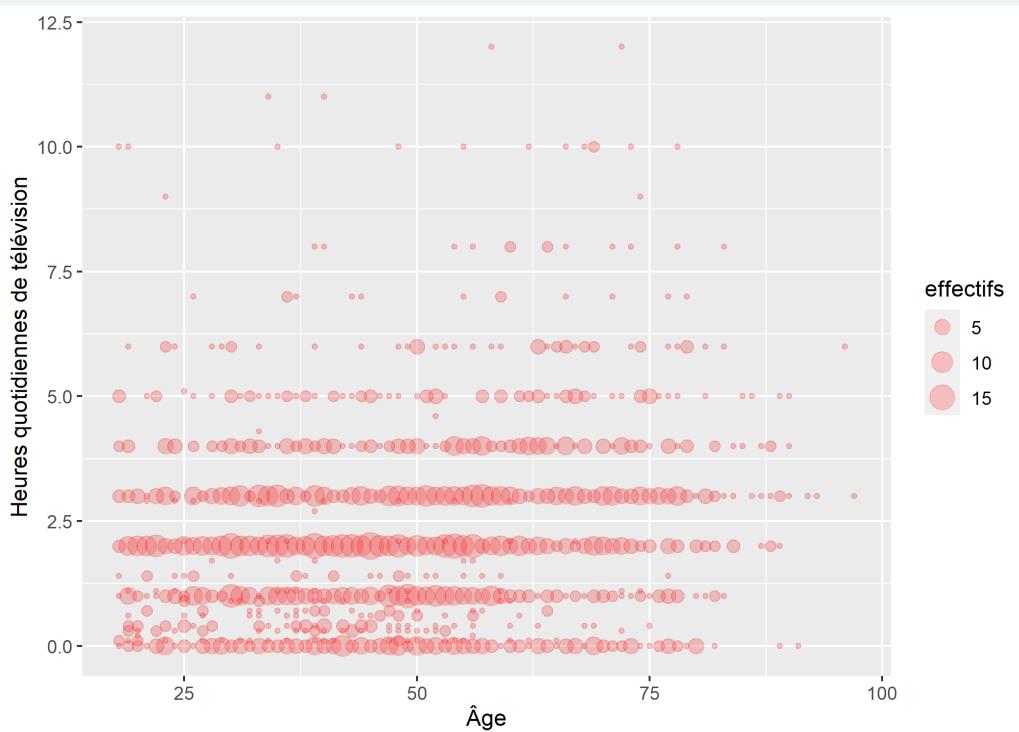


Figure 15. Nuage de points semi-transparents

**NOTE**

Une variante à `geom_point` est `geom_count` qui compte le nombre d'observations situées sur le même point et dessine des points dont la taille est proportionnelle à ce nombre d'observations.

```
R> ggplot(d) +
  aes(x = age, y = heures.tv) +
  geom_count(colour = "red", alpha = .2) +
  xlab("Âge") +
  ylab("Heures quotidiennes de télévision") +
  labs(size = "effectifs")
```



*Figure 16. Nuage de points de taille proportionnelle au nombre d'observations*

Pour représenter une troisième variable quantitative, on pourra faire varier la taille des points avec l'esthétique `size`. Pour une variable qualitative, on pourra faire varier la couleur avec `colour`. Pour faciliter la lecture, on positionnera la légende en bas du graphique, en modifiant l'argument `legend.position` via la fonction `theme`.

```
R> data("rp99")
rp99$prop.proprio <- 0
rp99[rp99$propriو >= mean(rp99$propriو), ]$prop.proprio <- 1
rp99$propriو <- factor(rp99$propriو, 0:1, c("non", "oui"))
ggplot(rp99) +
  aes(x = dipl.aucun, y = tx.chom, size = pop.tot, colour = prop.propri
o) +
  geom_point(alpha = .5) +
  xlab("% sans diplôme") +
  ylab("Taux de chômage") +
  labs(size = "Population totale", colour = "Proportion de propriétaires
supérieure à la moyenne") +
  theme(
    legend.position = "bottom",
    legend.box = "vertical"
  )
)
```

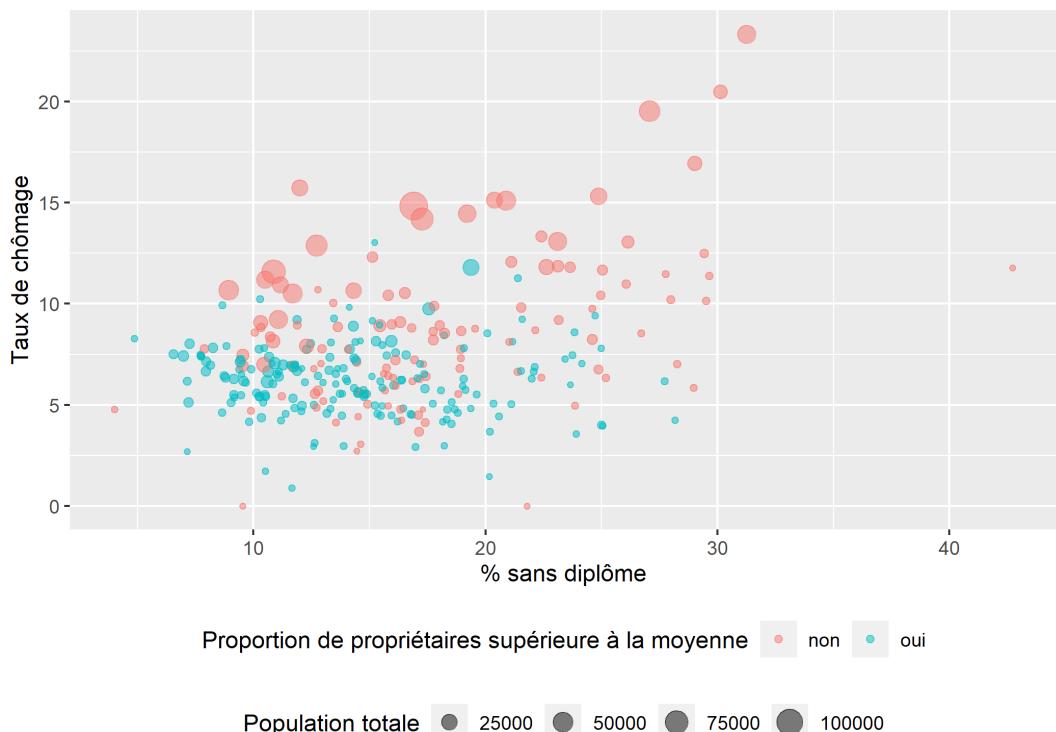


Figure 17. Nuage de points proportionnels

`geom_smooth` permet d'ajouter au graphique une moyenne mobile du nuage de points avec son intervalle de confiance. Notez que l'on ajoute `geom_smooth` au graphique avant `geom_point` puisque l'ordre dans lequel sont affichées les différentes couches du graphique dépend de l'ordre dans lequel elles

ont été ajoutées. Dans cet exemple, nous souhaitons afficher les points «au-dessus» de la moyenne mobile.

```
R> ggplot(rp99) +
  aes(x = dipl.sup, y = cadres) +
  geom_smooth() +
  geom_point() +
  xlab("% de diplômés du supérieur") +
  ylab("% de cadres")
```

`geom\_smooth()` using method = 'loess' and formula 'y ~ x'

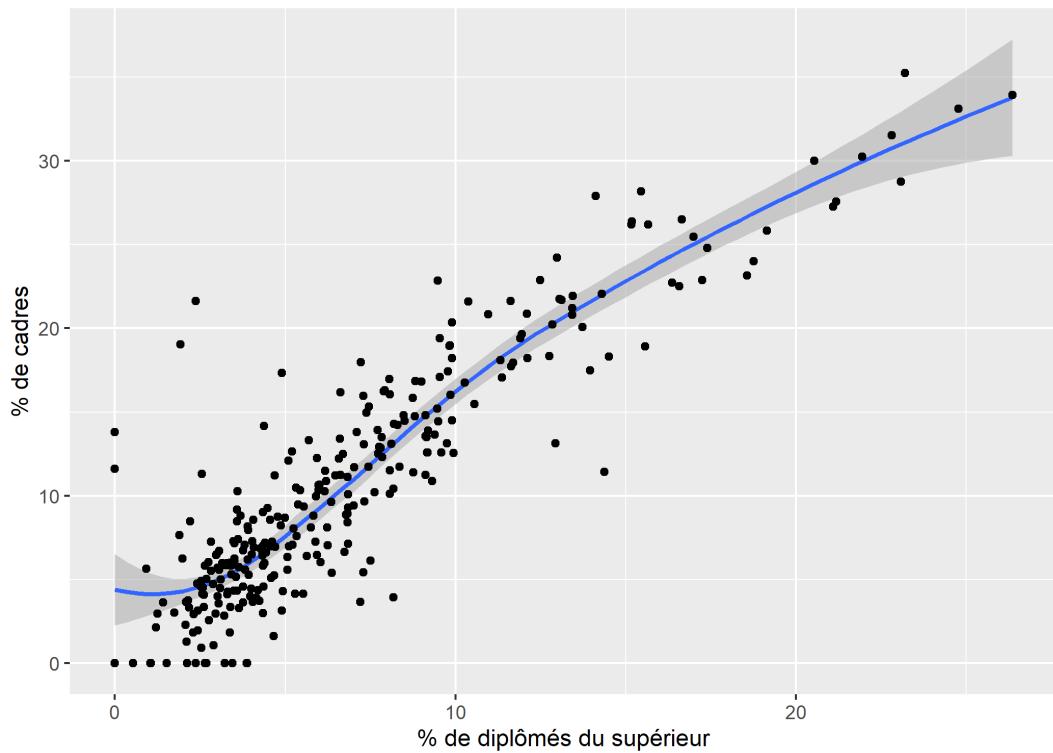


Figure 18. Nuage de points avec moyenne mobile

Si l'on préfère afficher plutôt la droite de régression, on indiquera à `geom_smooth` l'argument `method = "lm"`.

```
R> ggplot(rp99) +  
  aes(x = dipl.sup, y = cadres) +  
  geom_smooth(method = "lm") +  
  geom_point() +  
  xlab("% de diplômés du supérieur") +  
  ylab("% de cadres")
```

`geom\_smooth()` using formula 'y ~ x'

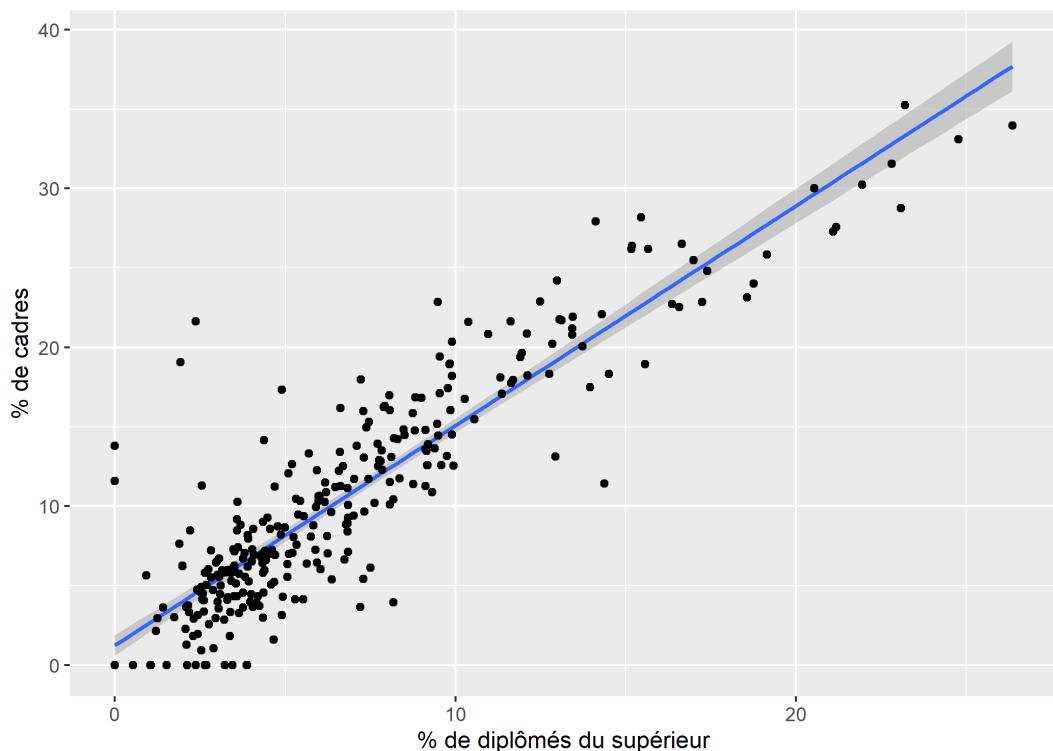


Figure 19. Nuage de points avec droite de régression linéaire

## Matrice de nuages de points et matrice de corrélation

[ggplot2](#) ne fournit pas de fonction native pour la réalisation d’une matrice de nuages de points. Cependant, il existe plusieurs extensions permettant d’étendre [ggplot2](#). Parmi celles-ci, l’extension [GGally](#) propose une fonction [ggpairs](#) correspondant exactement à notre besoin.

```
R> library(GGally)
ggpairs(rp99[, c("proprio", "hlm", "locataire", "maison")])
```

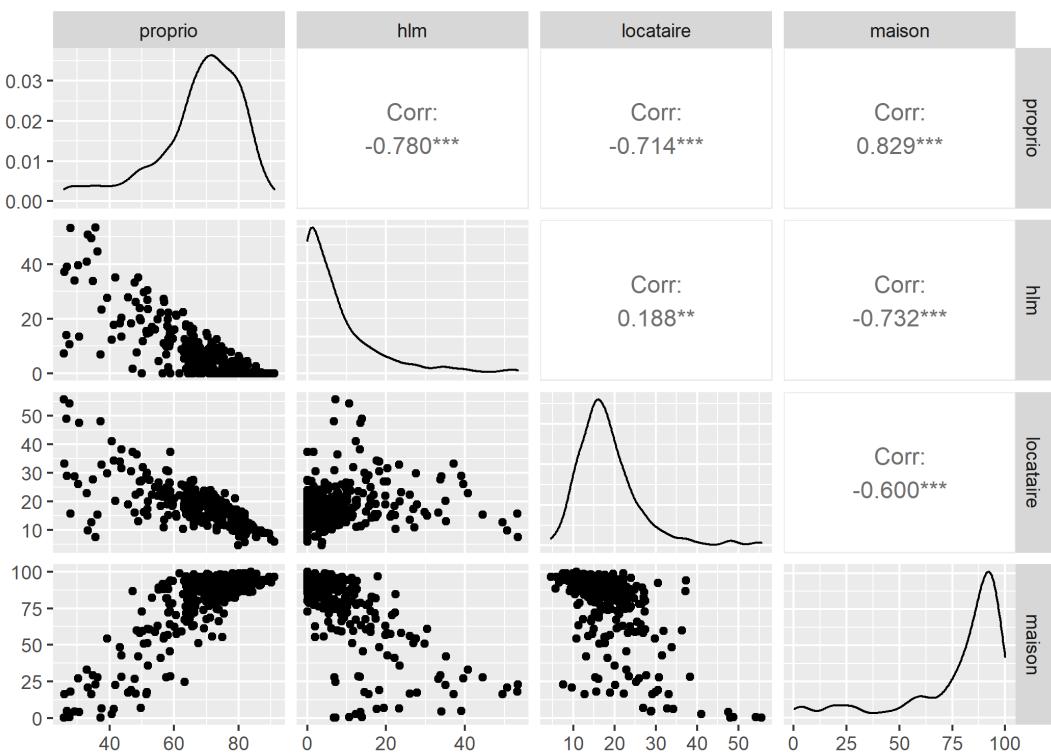


Figure 20. Matrice de nuages de points

`ggpairs` accepte même des variables catégorielles ainsi que des esthétiques supplémentaires, offrant ainsi plus de possibilités que la fonction `pairs`<sup>1</sup>.

1. Pour plus de détails, on pourra lire <https://tgmstat.wordpress.com/2013/11/13/plot-matrix-with-the-r-package-ggally/>.

```
R> ggpairs(rp99[, c("hlm", "locataire", "maison", "prop.proprio")], aes(color = prop.proprio))
```

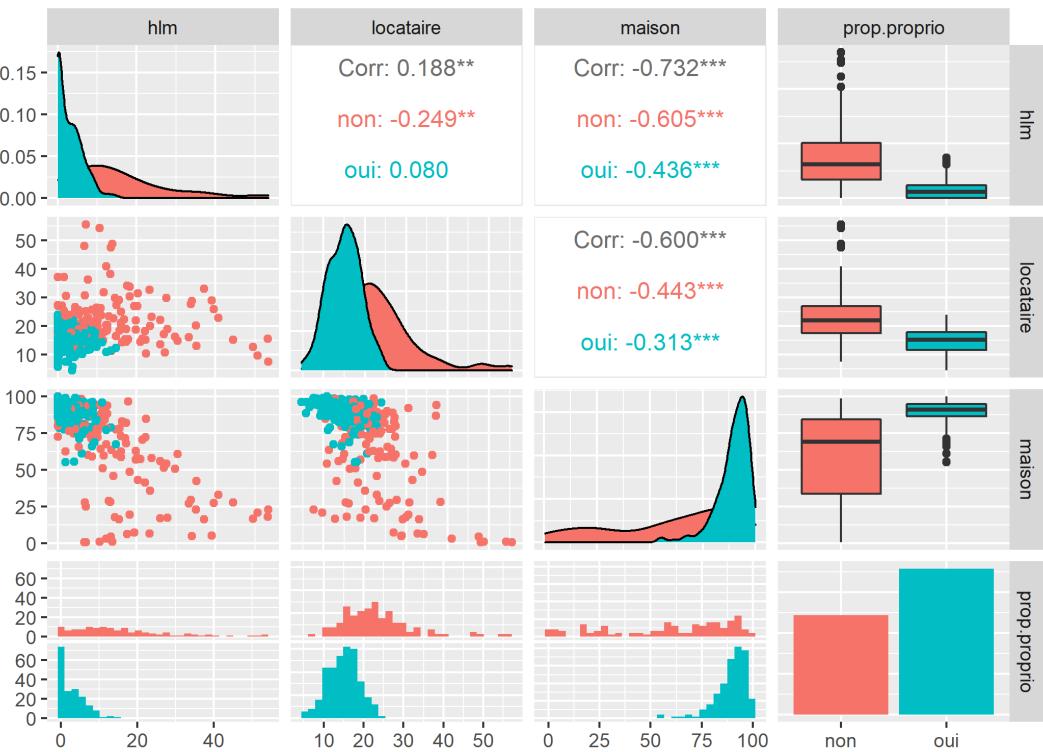


Figure 21. Matrice de nuages de points avec variables catégorielles

**GGally** propose également une fonction `ggcorr` permettant d'afficher une matrice de corrélation entre variables quantitatives<sup>2</sup>.

2. Pour une présentation détaillée de cette fonction et de ses options, voir <https://briatte.github.io/ggcorr/>.

```
R> ggc当地 (rp99)
```

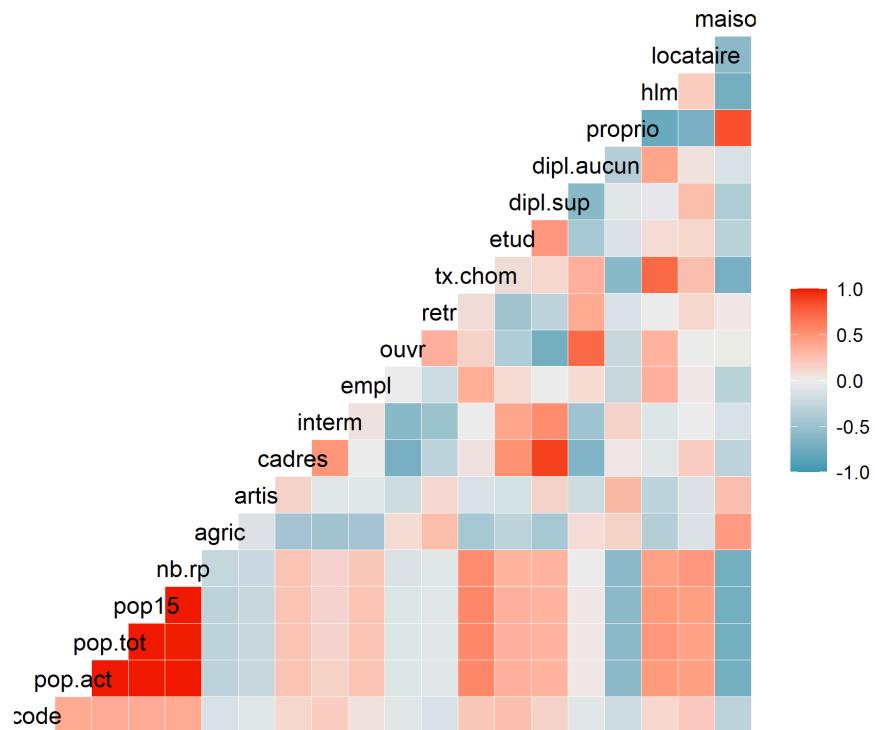


Figure 22. Matrice de corrélation

## Estimation locale de densité (et représentations associées)

On peut aisément représenter une estimation locale de densité avec la géométrie `geom_density_2d`.

```
R> ggplot(d) +  
  aes(x = age, y = heures.tv) +  
  geom_density_2d() +  
  xlab("Âge") +  
  ylab("Heures quotidiennes de télévision")
```

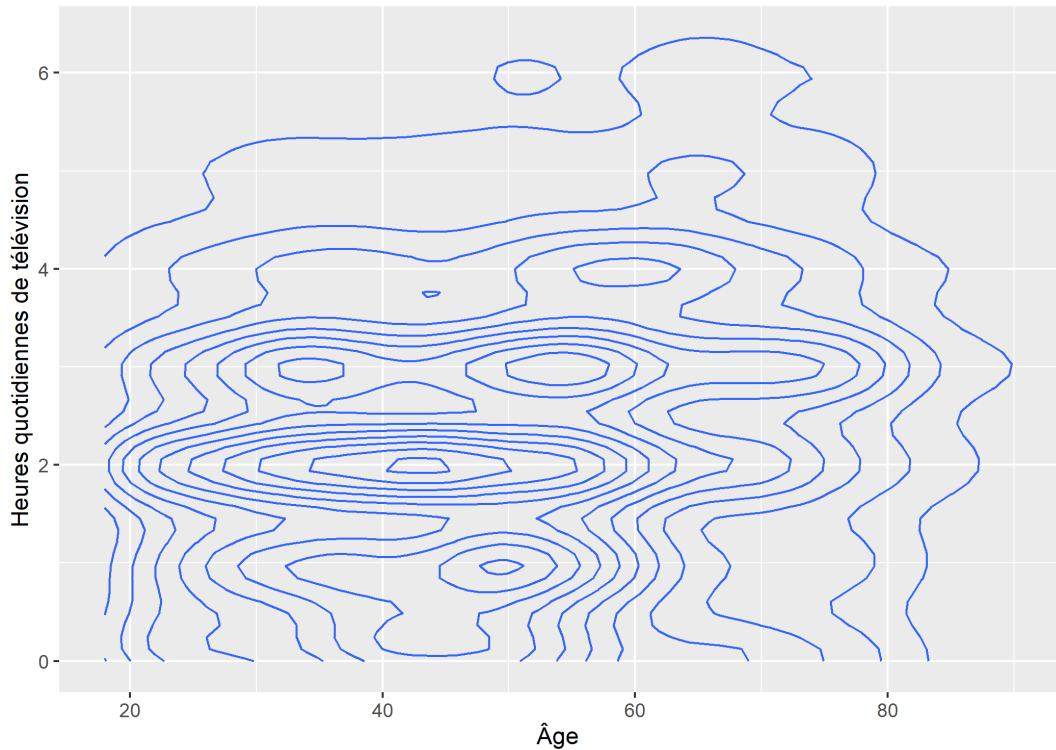


Figure 23. Estimation locale de densité (contours)

Par défaut, le résultat est représenté sous forme de contours. Pour obtenir une représentation avec des polygones, on appellera la statistique `stat_density_2d` en forçant la géométrie.

```
R> ggplot(d) +  
  aes(x = age, y = heures.tv, fill = after_stat(level)) +  
  stat_density_2d(geom = "polygon") +  
  xlab("Âge") +  
  ylab("Heures quotidiennes de télévision") +  
  labs(fill = "Densité")
```

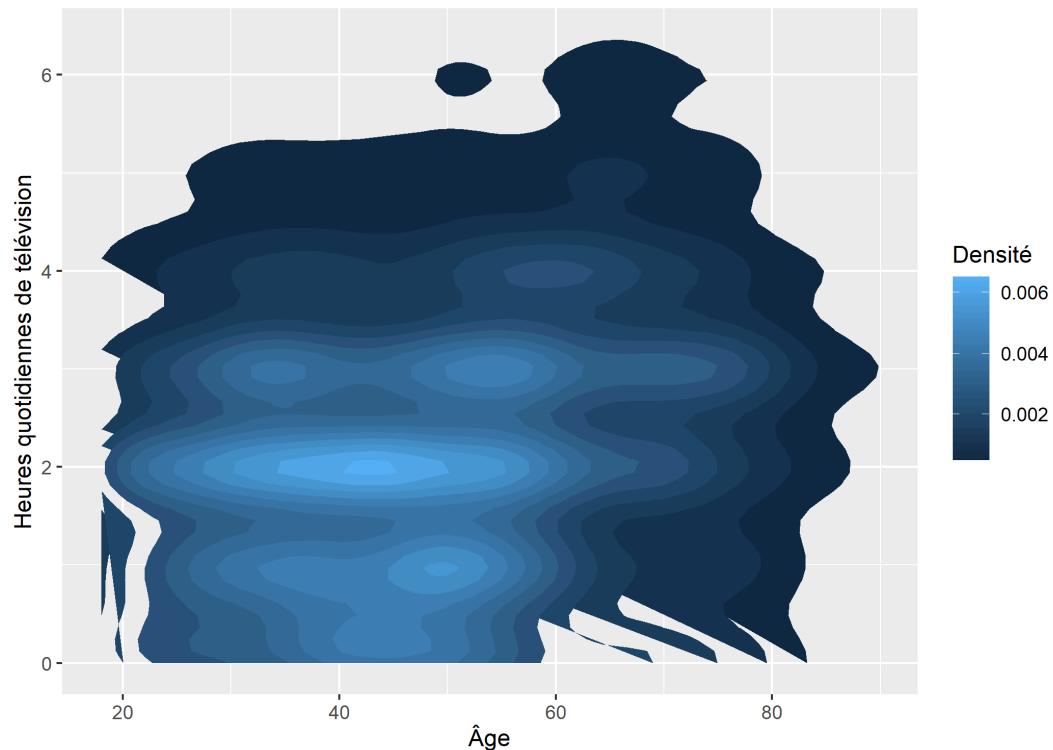


Figure 24. Estimation locale de densité (contours)

ggplot2 propose également deux géométries, `geom_bin2d` et `geom_hex`, permettant d'effectuer à un comptage des effectifs en deux dimensions.

```
R> ggplot(d) +  
  aes(x = age, y = heures.tv) +  
  geom_bin2d() +  
  xlab("Âge") +  
  ylab("Heures quotidiennes de télévision") +  
  labs(fill = "Effectifs")
```

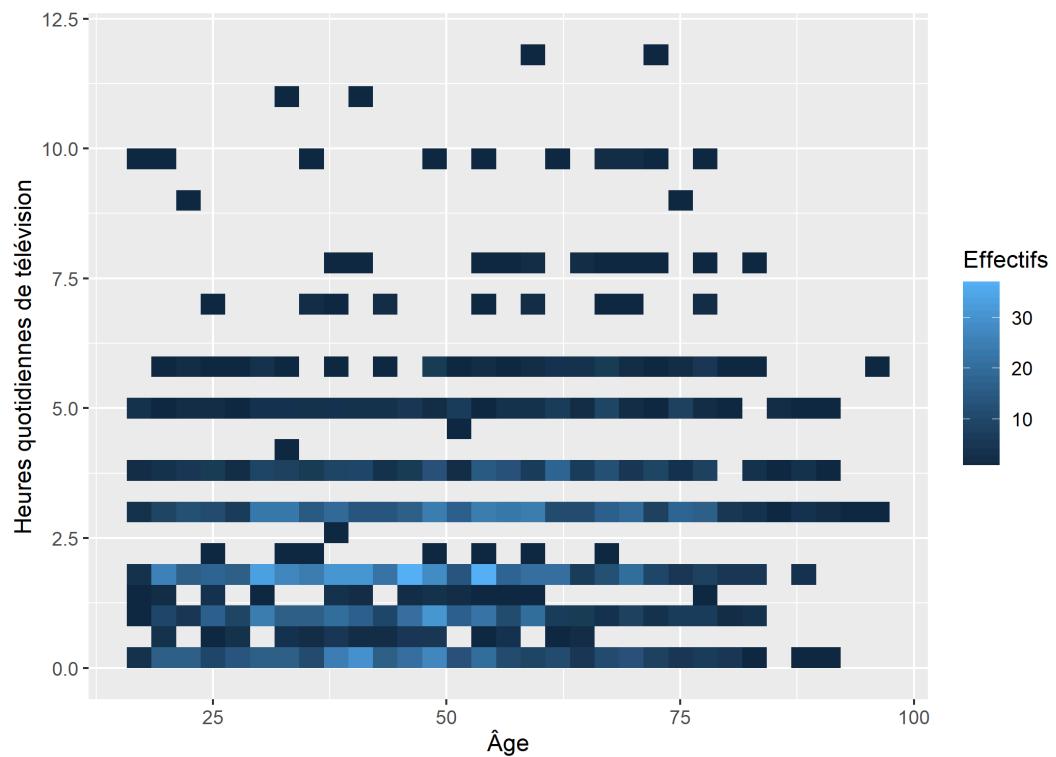


Figure 25. Effectifs en deux dimensions

```
R> ggplot(d) +  
  aes(x = age, y = heures.tv) +  
  geom_hex() +  
  xlab("Âge") +  
  ylab("Heures quotidiennes de télévision") +  
  labs(fill = "Effectifs")
```

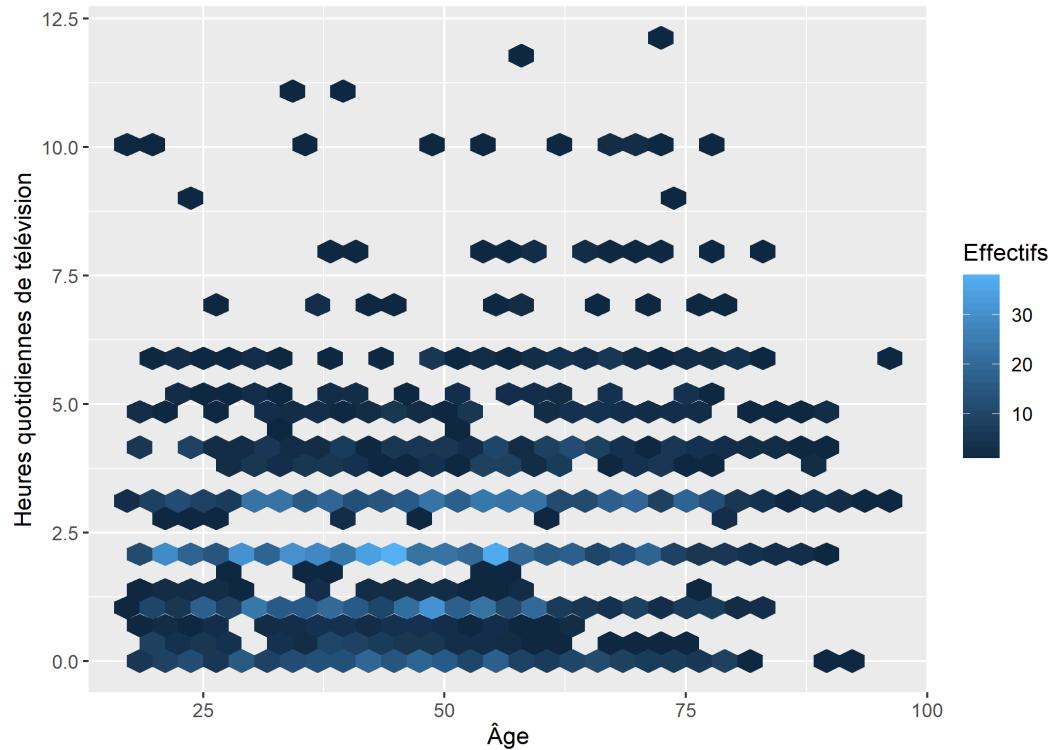


Figure 26. Effectifs en deux dimensions (hexagones)

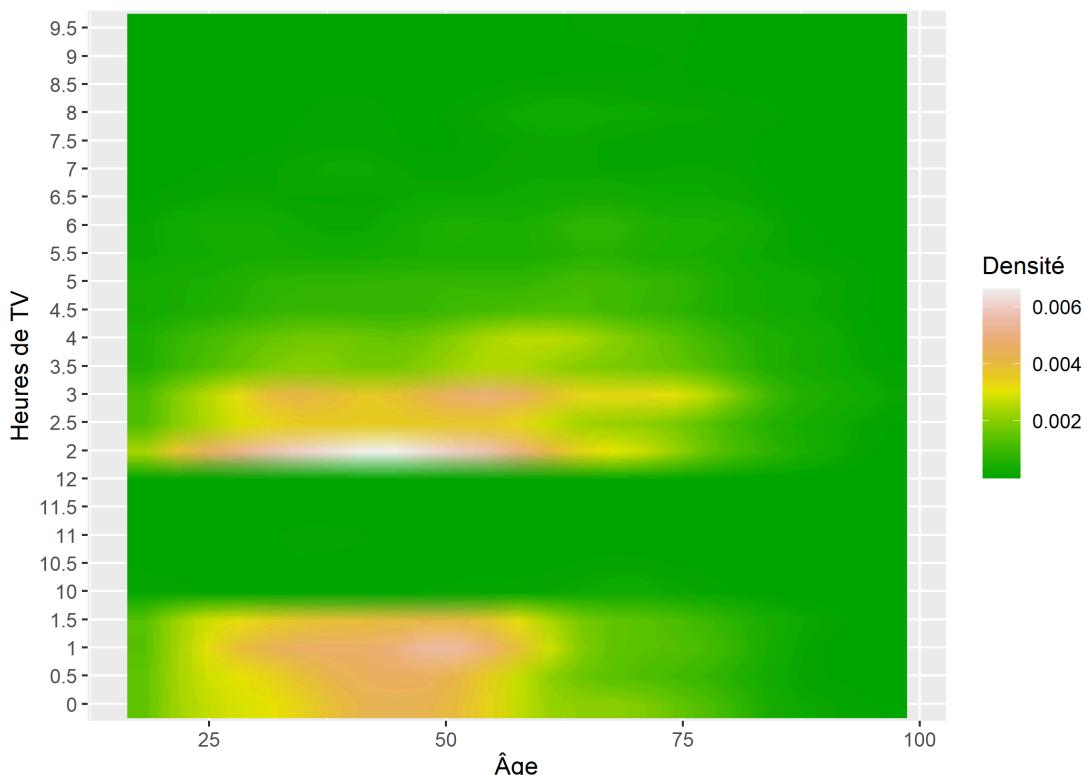
#### NOTE

Pour reproduire à l'identique l'exemple donné dans le chapitre statistique bivariée, page 366, on aura besoin de la méthode `tidy` de l'extension `broom` afin de transformer le résultat de `kde2d` en un tableau de données exploitables par `ggplot2`. `tidy` est une méthode générique permettant de transformer un grand nombre d'objets (et en particulier les résultats d'un modèle) en un tableau de données exploitable by `ggplot2`.

```
R> library(MASS)
tmp <- d[, c("age", "heures.tv")]
tmp <- tmp[complete.cases(tmp), ]
library(broom)
tmp <- tidy(kde2d(tmp$age, tmp$heures.tv))
str(tmp)
```

```
tibble [625 x 3] (S3:tbl_df/tbl/data.frame)
$ x: num [1:625] 18 18 18 18 18 18 18 18 18 ...
$ y: chr [1:625] "0" "0.5" "1" "1.5" ...
$ z: num [1:625] 0.00147 0.00148 0.00136 0.00137 0.00238 ...
```

```
R> ggplot(tmp) +
  aes(x = x, y = y, fill = z) +
  geom_raster(interpolate = TRUE) +
  scale_fill_gradientn(colors = terrain.colors(14)) +
  labs(x = "Âge", y = "Heures de TV", fill = "Densité")
```



## Diagramme de Cleveland

Pour un diagramme de Cleveland, on aura recours à la géométrie `geom_point`. Cependant, il faudra lui préciser que l'on souhaite utiliser la statistique `stat_count` afin que les effectifs soient calculés pour chaque valeur de `x`.

```
R> ggplot(d) +
  aes(x = c1so) +
  geom_point(stat = "count") +
  xlab("Sentiment d'appartenance à une classe sociale") +
  ylab("Effectifs")
```

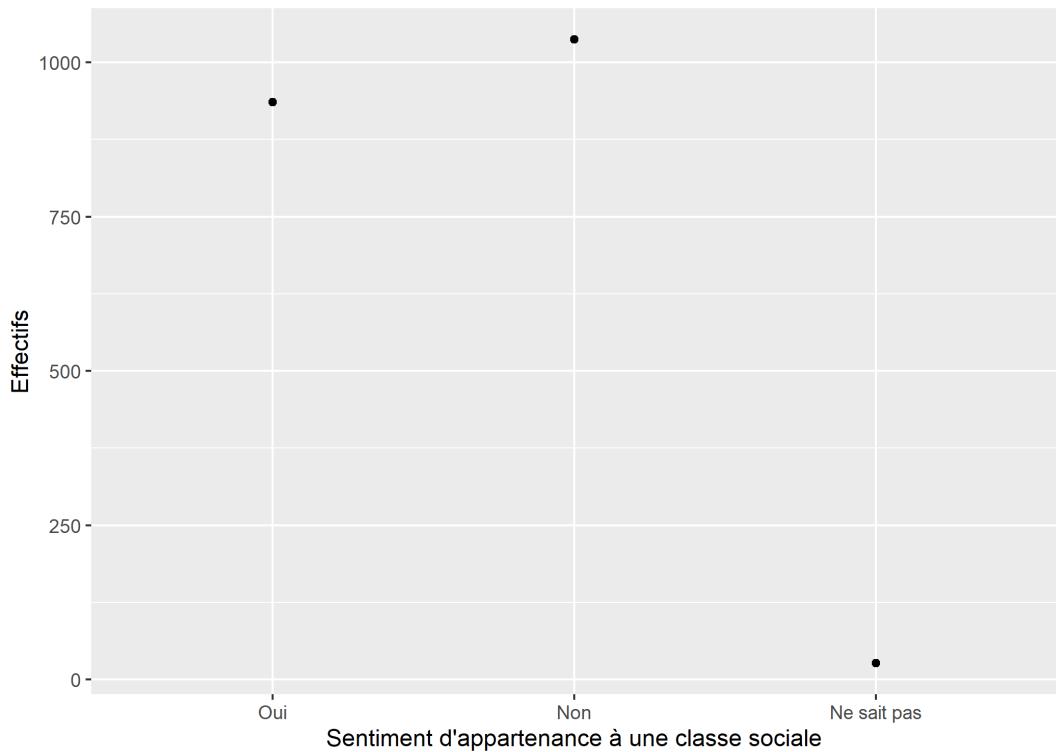


Figure 27. Diagramme de Cleveland

Une alternative, notamment si l'on souhaite un diagramme de Cleveland ordonné, consiste à calculer les effectifs de chaque modalité en amont. `ggplot2` ayant besoin d'un tableau de données en entrée, nous calculerons notre tableau de fréquences avec `xtabs` et le transformerons en tableau de données avec `as.data.frame`. Pour que les niveaux de qualification soient représentés selon leur effectif, il est nécessaire d'ordonner les étiquettes du facteur de manière adéquate. Enfin, nous utiliserons `coord_flip` pour intervertir l'axe des `x` et celui des `y`.

```
R> tab <- as.data.frame(xtabs(~qualif, d))
  tab$qualif <- factor(tab$qualif, levels = tab$qualif[order(tab$Freq)])
  str(tab)
```

```
'data.frame':    7 obs. of  2 variables:
```

```
$ qualif: Factor w/ 7 levels "Autre","Technicien",...: 4 6 2 3 5 7 1  
$ Freq : int 203 292 86 160 260 594 58
```

```
R> ggplot(tab) +  
  aes(x = qualif, y = Freq) +  
  geom_point() +  
  xlab("Niveau de qualification") +  
  ylab("Effectifs") +  
  coord_flip()
```

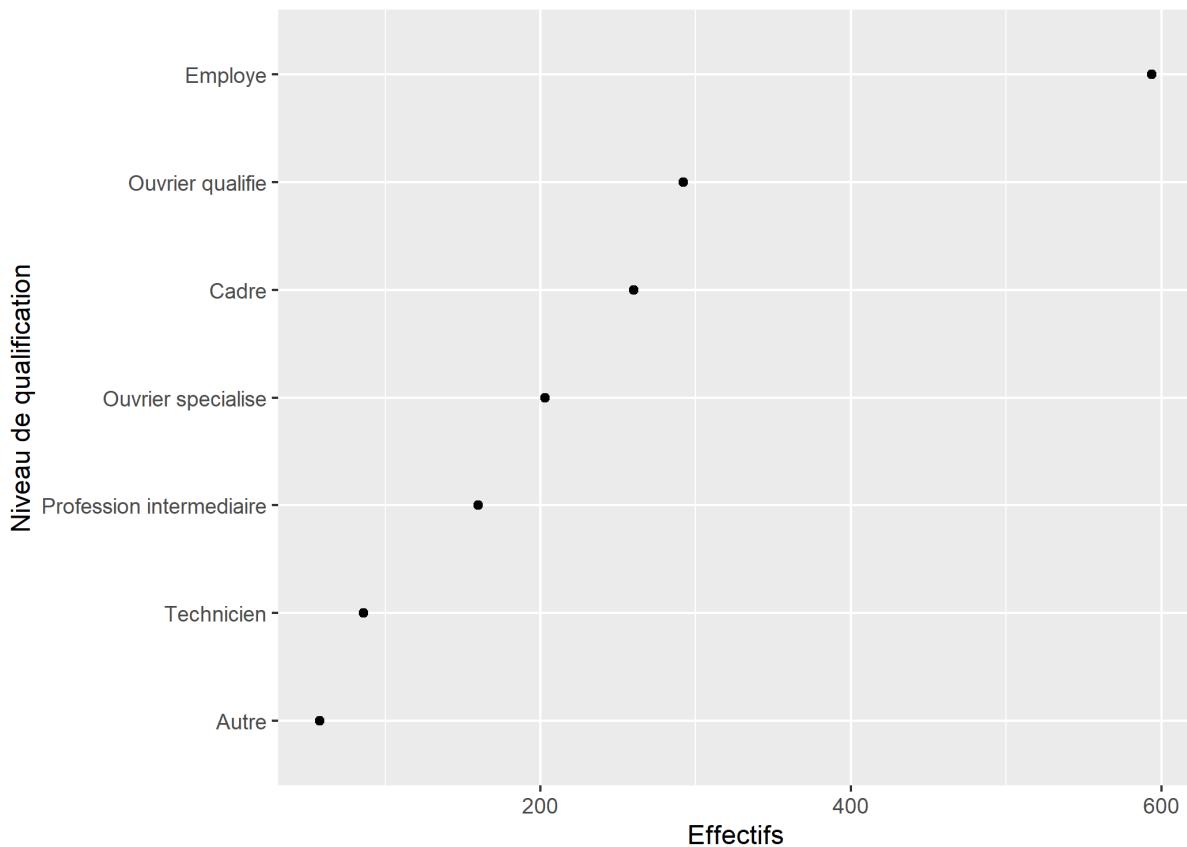
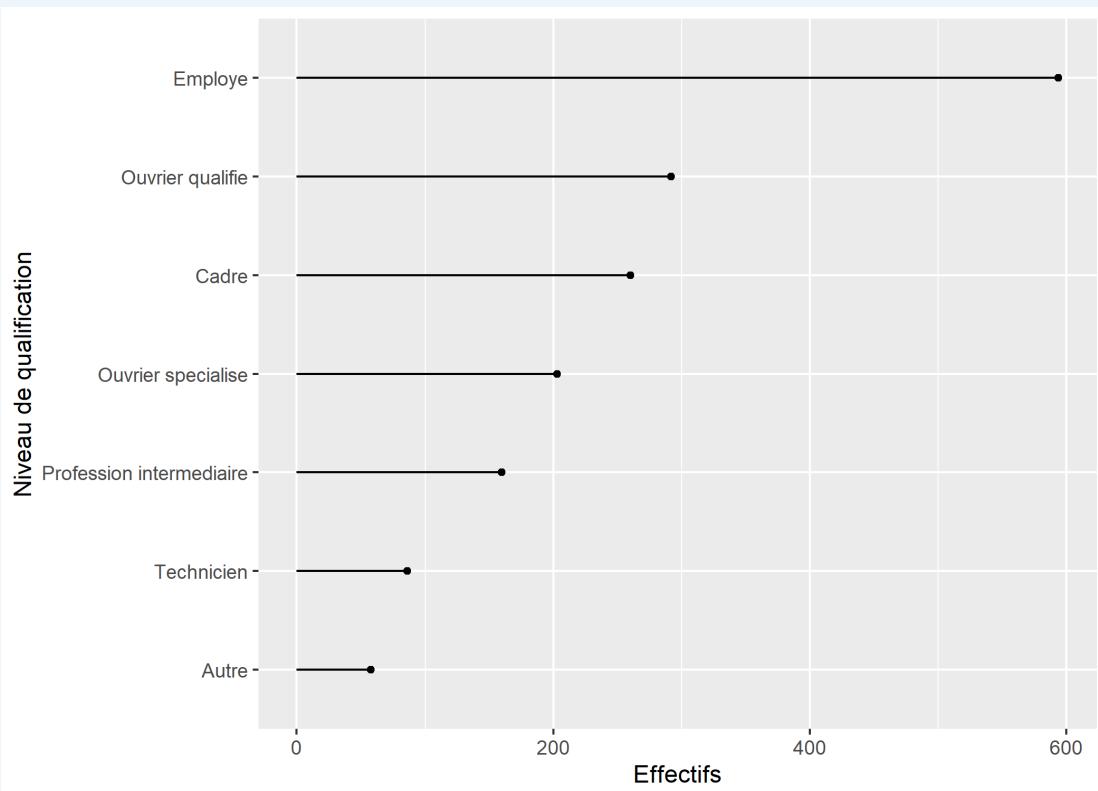


Diagramme de Cleveland ordonné

#### NOTE

L’extension `ggalt` propose quelques géométries supplémentaires pour `ggplot2`. L’une d’elles dite «en sucettes» (`geom_lollipop`) propose une représentation graphique au croisement entre un diagramme en bâtons et un diagramme de Cleveland.

```
R> library(ggalt)
ggplot(tab) +
  aes(x = qualif, y = Freq) +
  geom_lollipop() +
  xlab("Niveau de qualification") +
  ylab("Effectifs") +
  coord_flip()
```



## Diagrammes en barres

Un diagramme en barres se construit avec la géométrie `geom_bar`.

```
R> d$qualreg <- as.character(d$qualif)
d$qualreg[d$qualif %in% c("Ouvrier specialise", "Ouvrier qualifie")] <-
  "Ouvrier"
d$qualreg[d$qualif %in% c("Profession intermediaire",
  "Technicien")] <- "Intermediaire"
ggplot(d) +
  aes(x = qualreg, fill = sport) +
  geom_bar() +
  xlab("CSP") +
  ylab("Effectifs") +
  labs(fill = "Pratique du sport")
```

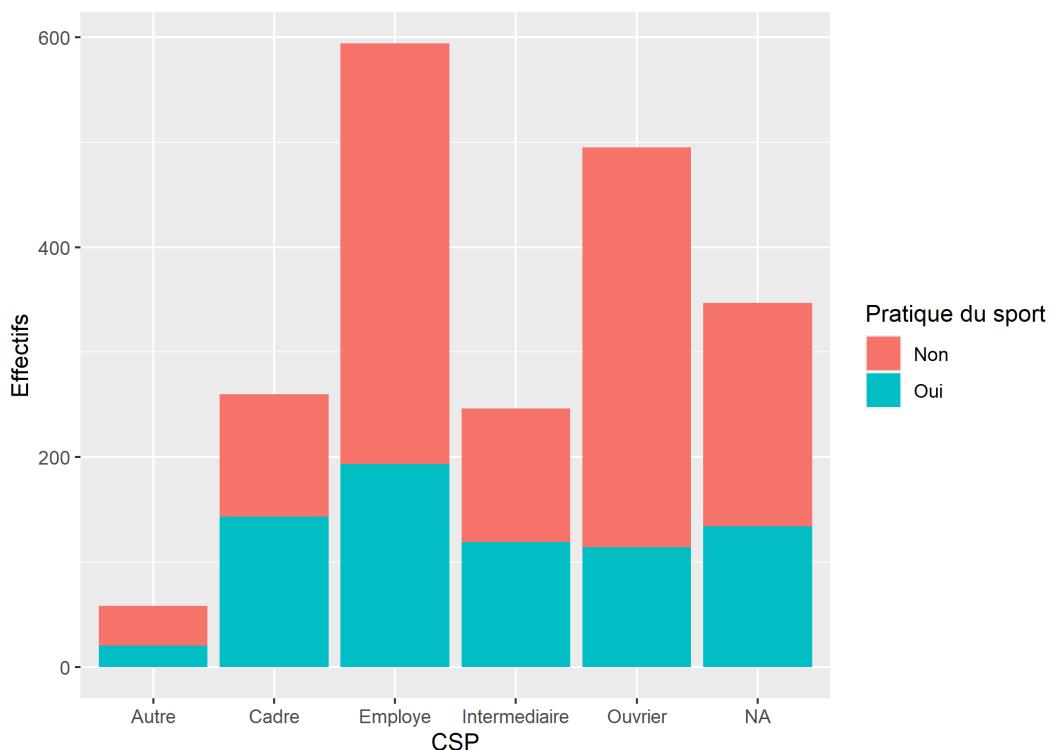


Figure 28. Diagramme en barres

On peut modifier la position des barres avec le paramètre `position`.

```
R> ggplot(d) +
  aes(x = qualreg, fill = sport) +
  geom_bar(position = "dodge") +
  xlab("CSP") +
  ylab("Effectifs") +
  labs(fill = "Pratique du sport")
```

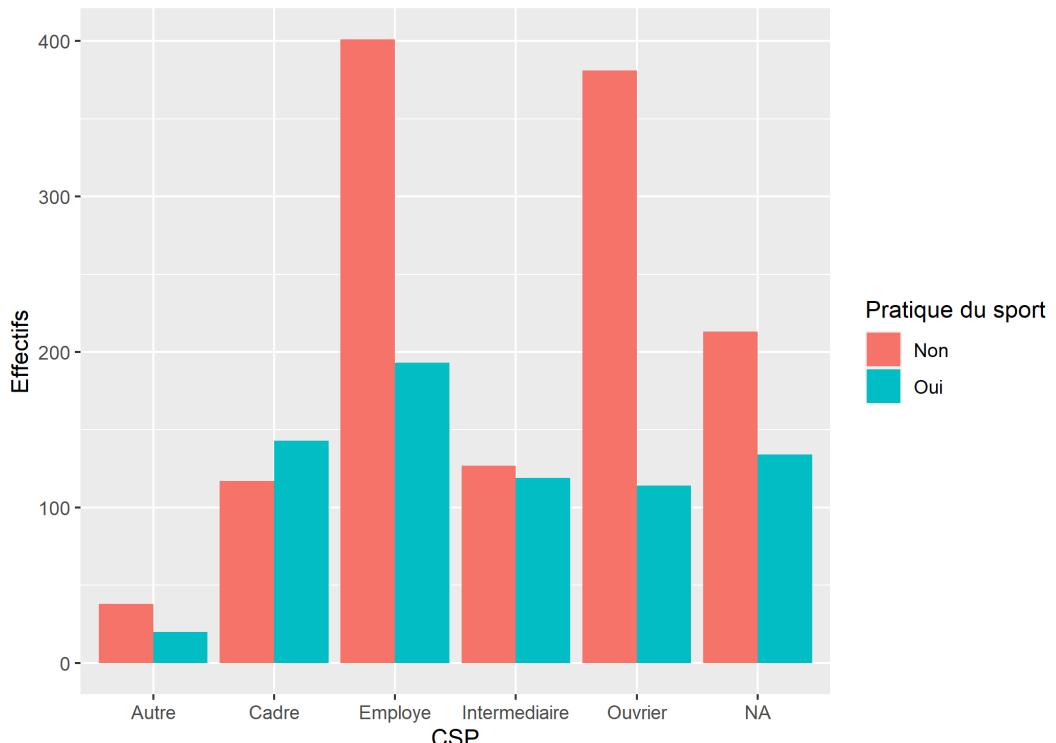


Figure 29. Diagramme en barres côté à côté

Pour des barres cumulées, on aura recours à `position = "fill"`. Pour que les étiquettes de l'axe des y soient représentées sous forme de pourcentages (i.e. 25% au lieu de 0.25), on aura recours à la fonction `percent` de l'extension `scales`, qui sera transmise à `ggplot2` via `scale_y_continuous`.

```
R> library(scales)
ggplot(d) +
  aes(x = qualreg, fill = sport) +
  geom_bar(position = "fill") +
  xlab("CSP") +
  ylab("Proportion") +
  labs(fill = "Pratique du sport") +
  scale_y_continuous(labels = percent)
```

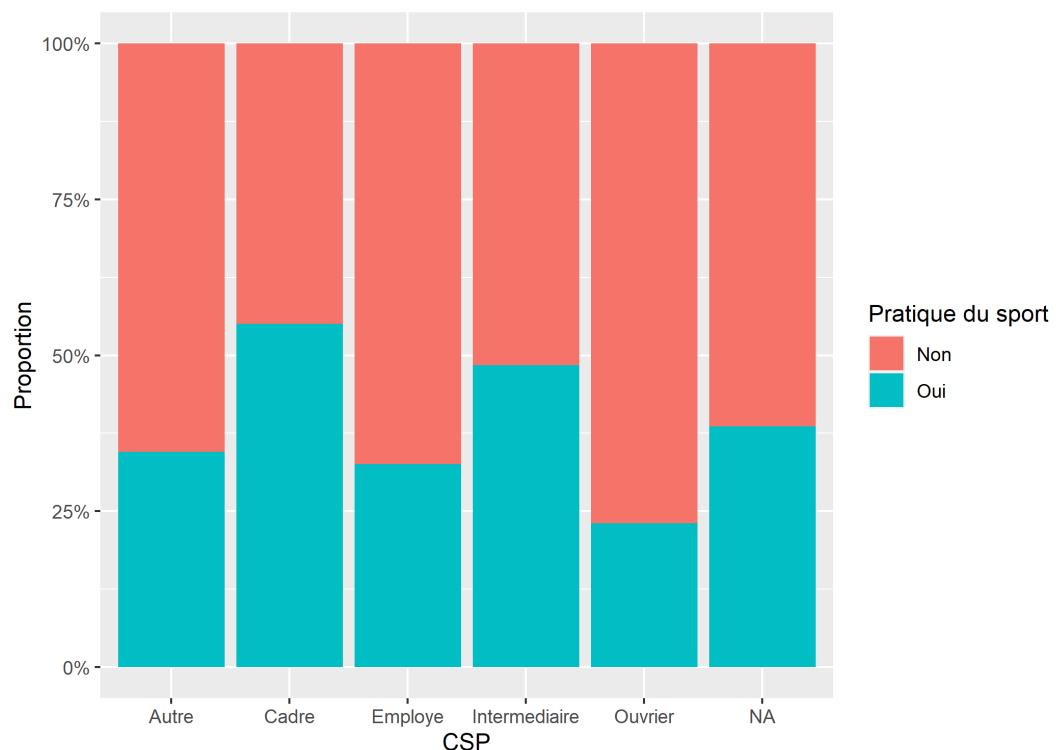


Figure 30. Diagramme en barres cumulées

## NOTE

### Ajouter des étiquettes sur un diagramme en barre

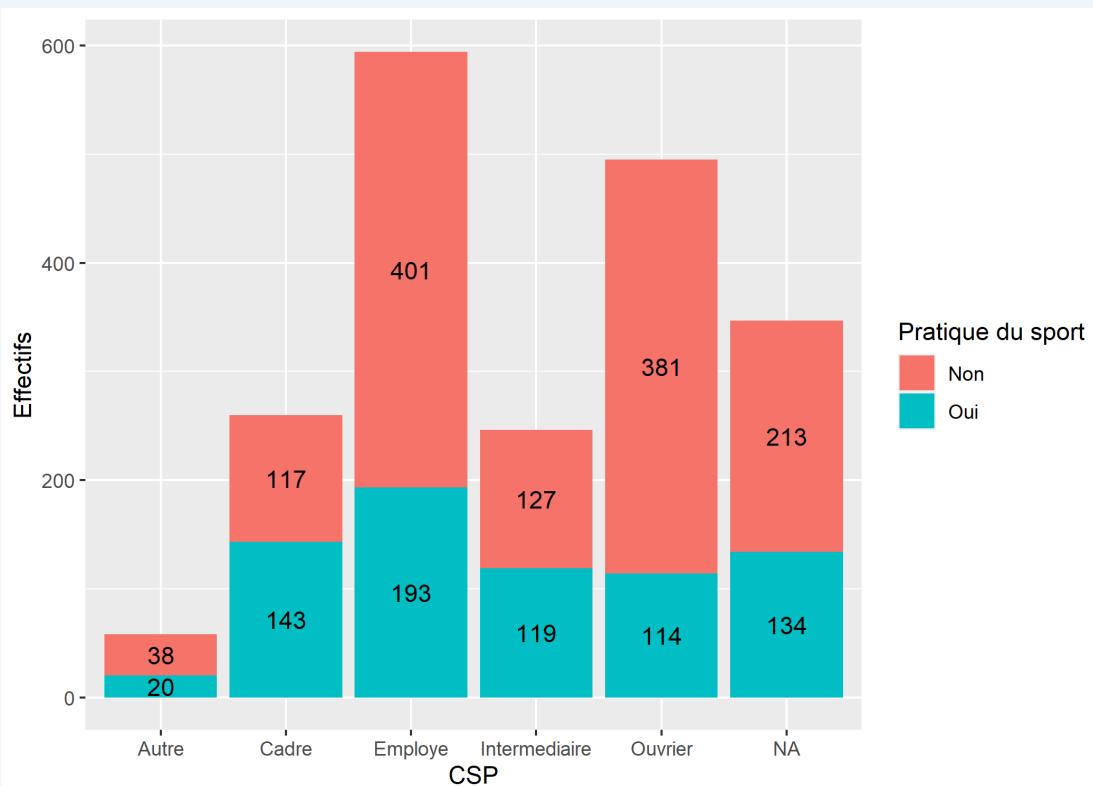
Il est facile d'ajouter des étiquettes en ayant recours à `geom_text`, à condition de lui passer les bon paramètres.

Tout d'abord, il faudra préciser `stat = "count"` pour indiquer que l'on souhaite utiliser la statistique `stat_count` qui est celle utilisé par défaut par `geom_bar`. C'est elle qui permet de compter le nombre d'observation.

Il faut ensuite utiliser l'esthétique `label` pour indiquer ce que l'on souhaite afficher comme étiquettes. `after_stat(count)` permet d'accéder à la variable calculée par `stat_count`.

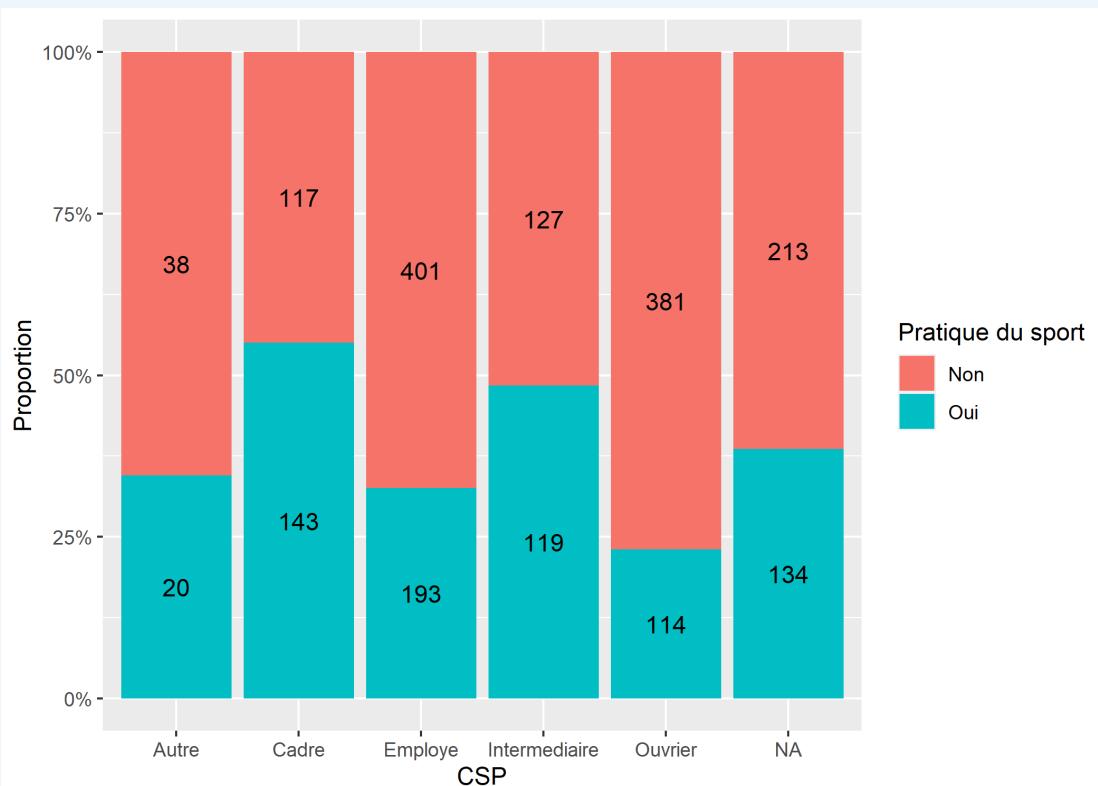
Enfin, il faut indiquer la position verticale avec `position_stack`. En précisant un ajustement de vertical de `0.5`, on indique que l'on souhaite positionner l'étiquette au milieu.

```
R> ggplot(d) +
  aes(x = qualreg, fill = sport) +
  geom_bar() +
  geom_text(aes(label = after_stat(count)), stat = "count", position = position_stack(.5)) +
  xlab("CSP") +
  ylab("Effectifs") +
  labs(fill = "Pratique du sport")
```



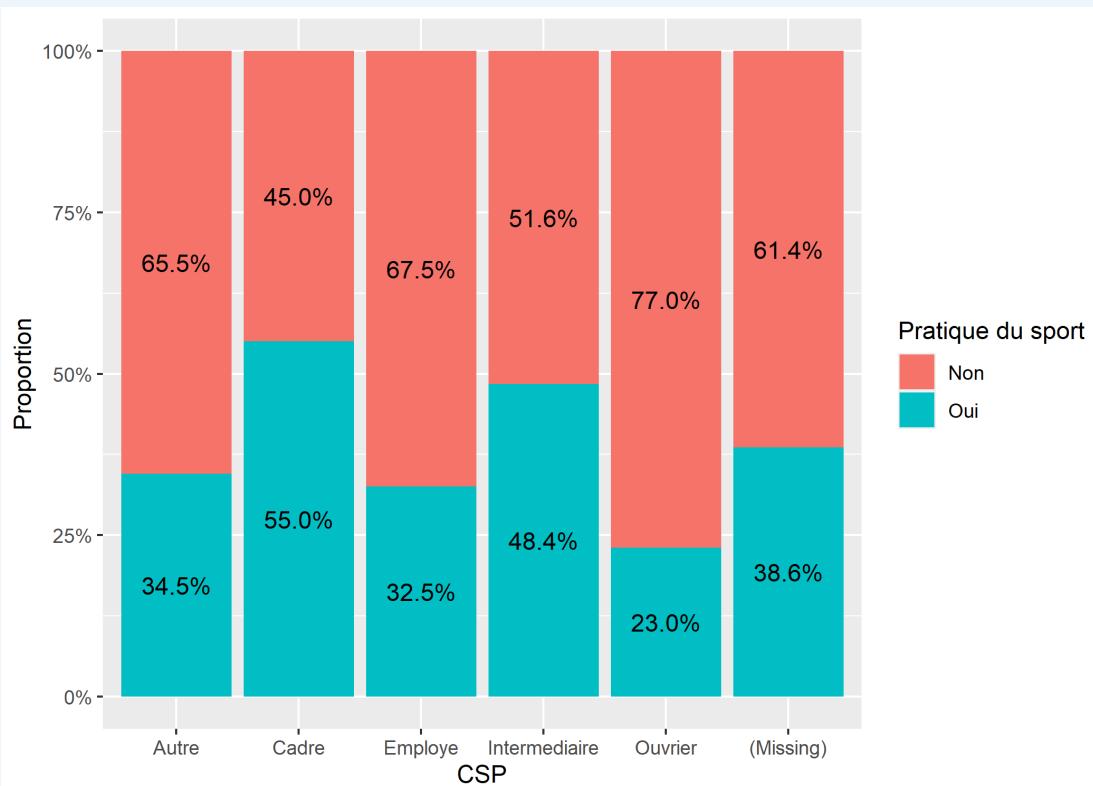
Pour un graphique en barre cumulées, on peut utiliser de manière similaire [position\\_fill](#).

```
R> ggplot(d) +
  aes(x = qualreg, fill = sport) +
  geom_bar(position = "fill") +
  geom_text(aes(label = after_stat(count)), stat = "count", position = position_fill(.5)) +
  xlab("CSP") +
  ylab("Proportion") +
  labs(fill = "Pratique du sport") +
  scale_y_continuous(labels = percent)
```



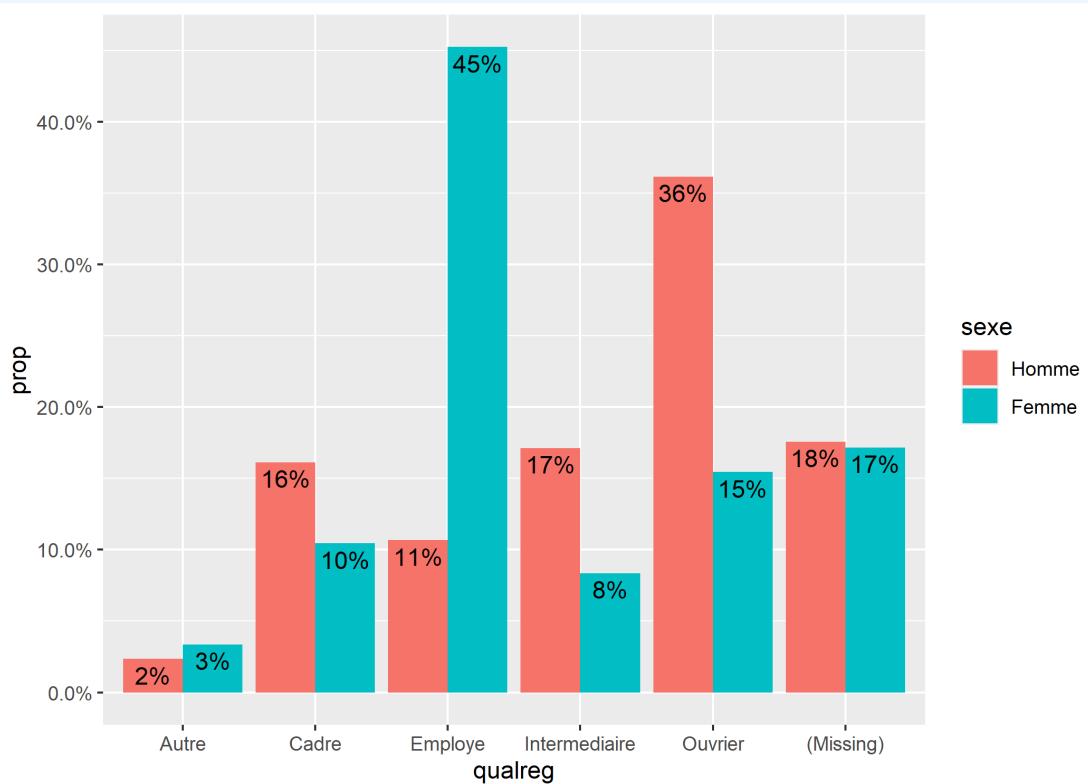
On ne peut afficher directement les proportions avec `stat_count`. Cependant, l'extension **GGally** fournit une statistique `stat_prop` dont le dénominateur peut être personnalisé via l'esthétique `by` (attention : l'esthétique attendue doit impérativement être un facteur).

```
R> library(GGally)
d$qualreg <- forcats::fct_explicit_na(factor(d$qualreg))
ggplot(d) +
  aes(x = qualreg, fill = sport) +
  geom_bar(position = "fill") +
  geom_text(aes(by = qualreg), stat = "prop", position = position_fill
l(.5)) +
  xlab("CSP") +
  ylab("Proportion") +
  labs(fill = "Pratique du sport") +
  scale_y_continuous(labels = scales::percent)
```



On peut aussi comparer facilement deux distributions, ici la proportion de chaque niveau de qualification au sein chaque sexe.

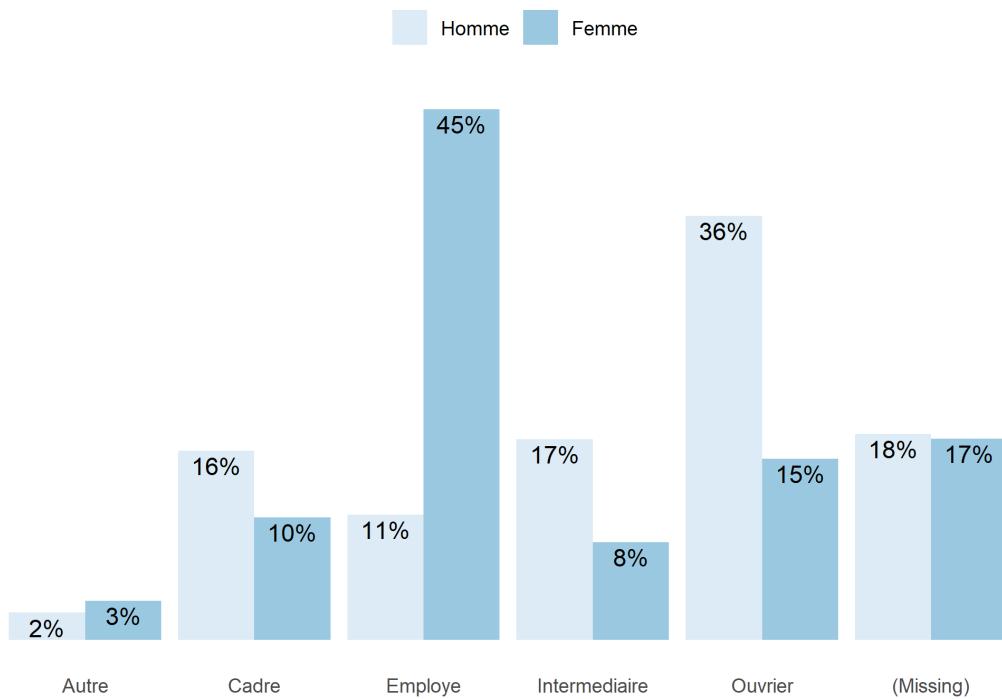
```
R> p <- ggplot(d) +
  aes(
    x = qualreg, fill = sexe, by = sexe, y = after_stat(prop),
    label = scales::percent(after_stat(prop), accuracy = 1),
  ) +
  geom_bar(stat = "prop", position = position_dodge(.9)) +
  geom_text(
    aes(y = after_stat(prop) - .005, stat = "prop",
        position = position_dodge(.9), vjust = "top"
    ) +
  scale_y_continuous(labels = percent)
p
```



Il est possible d'alléger le graphique en retirant des éléments superflus.

```
R> p +
  theme_light() +
  xlab("") +
  ylab("") +
  labs(fill = "") +
  ggtitle("Distribution de la population selon le niveau de qualification,
par sexe") +
  theme(
    panel.grid = element_blank(),
    panel.border = element_blank(),
    axis.text.y = element_blank(),
    axis.ticks = element_blank(),
    legend.position = "top"
  ) +
  scale_fill_brewer()
```

Distribution de la population selon le niveau de qualification, par sexe



## Comparer plusieurs variables discrètes et/ou

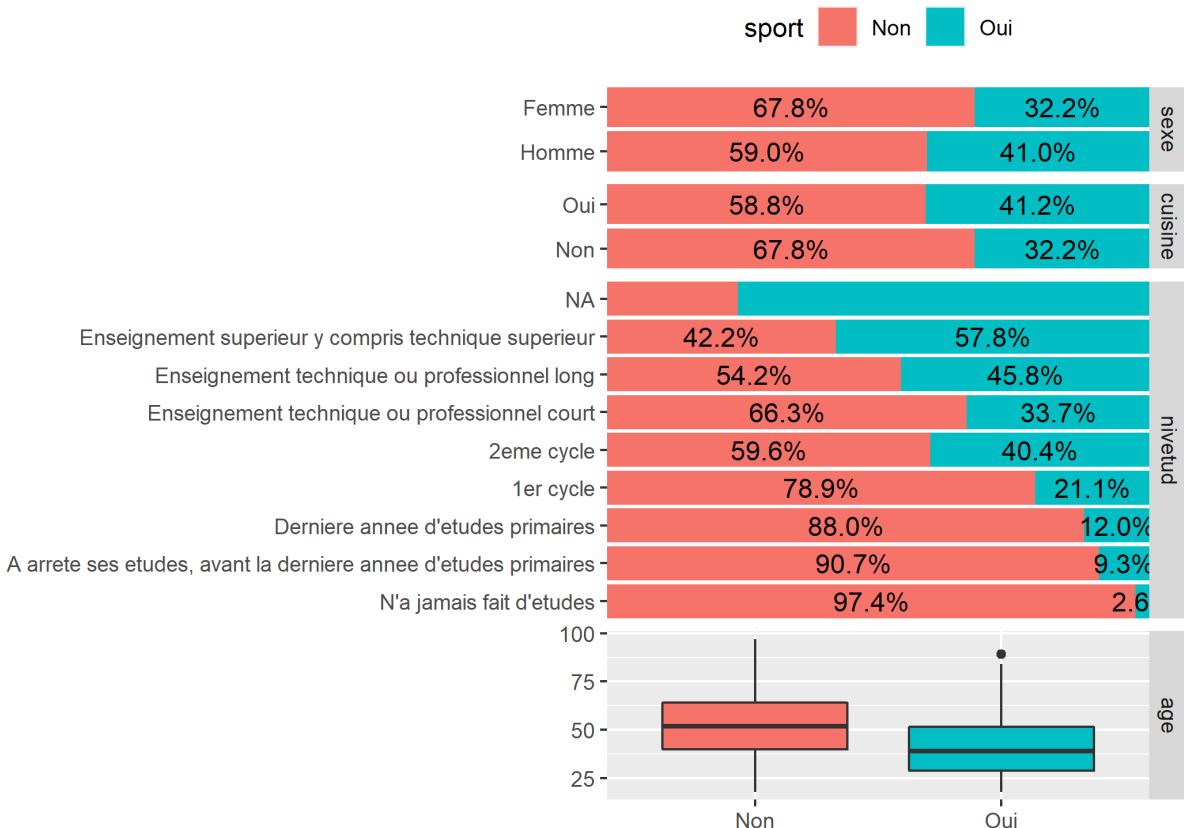
## continues

L'extension `GGally`, déjà évoquée, fournit également deux fonctions utiles pour comparer plusieurs variables discrètes (et/ou continues) : `ggbivariate` et `ggtab`.

`ggbivariate` produit des graphiques en barre pour deux variables discrètes et des boîtes à moustache lorsque l'on compare une variable discrète avec une variable continue.

```
R> library(GGally)
  ggbivariate(data = d, outcome = "sport", explanatory = c("sexe", "cuisine", "nivetud", "age"))
```

Warning: Removed 112 rows containing non-finite values  
(stat\_prop).



`ggtab` quant à elle affiche un tableau croisé avec le nombre d'observations pour deux variables discrètes et les médianes avec l'intervalle interquartile pour une variable discrète croisée avec une

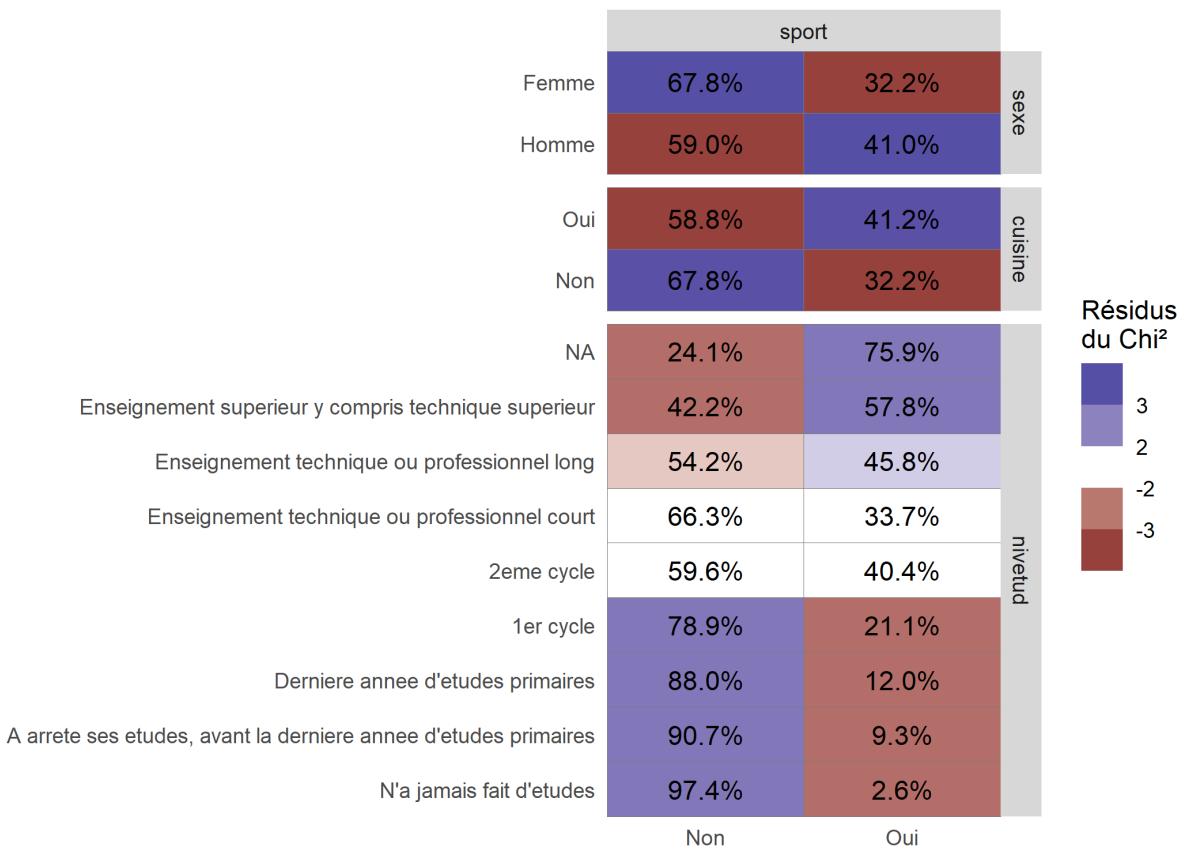
variable continue.

```
R> ggtab(data = d, columnsX = "sport", columnsY = c("sexe", "cuisine", "nivetud", "age"))
```

		sport		
		Femme	Homme	sexe
Oui		747	354	
Non		530	369	
		518	363	cuisine
		759	360	
NA		27	85	
Enseignement supérieur y compris technique supérieur		186	255	
Enseignement technique ou professionnel long		71	60	
Enseignement technique ou professionnel court		307	156	
2ème cycle		109	74	nivetud
1er cycle		161	43	
Dernière année d'études primaires		300	41	
A arrêté ses études, avant la dernière année d'études primaires		78	8	
N'a jamais fait d'études		38	1	
		Median: 52 [40-64]	Median: 39 [29-51.5]	age
		Non	Oui	

Il est possible de remplacer les effectifs par les pourcentages en ligne et de colorier les cellules en fonction des résidus du Chi<sup>2</sup> qui permettent de visualiser les cellules du tableau sur- ou sous-représentées.

```
R> ggtabulate(
  data = d,
  columnsX = "sport",
  columnsY = c("sexe", "cuisine", "nivetud"),
  cells = "row.prop",
  fill = "std.resid",
  legend = 1
) + labs(fill = "Résidus\ndu Chi2)
```



Pour plus de possibilités de personnalisation, voir la fonction `ggduo` de **GGally**.

## Graphe en mosaïque

Il n'y a pas, à ce jour, d'implémentation officielle des graphiques en mosaïque sous **ggplot2**. On pourra néanmoins se référer à l'extension **ggeomosaic** et sa fonction `geom_mosaic`<sup>3</sup>.

3. Pour information, une première implémentation expérimentale avait été proposée avec l'extension **productplots**, voir <https://github.com/hadley/productplots> et <http://vita.had.co.nz/papers/prodplots.html>

```
R> library(ggmosaic)
levels(d$sport) <- c("Ne fait pas de sport", "Pratique un sport")
levels(d$cuisine) <- c("Ne cuisine pas", "Cuisine")
levels(d$sexe) <- c("H", "F")

ggplot(data = d) +
  geom_mosaic(aes(x = product(sport, cuisine), fill = sexe, na.rm = TRUE)) +
  xlab("") + ylab("") +
  theme(legend.position = "bottom")
```

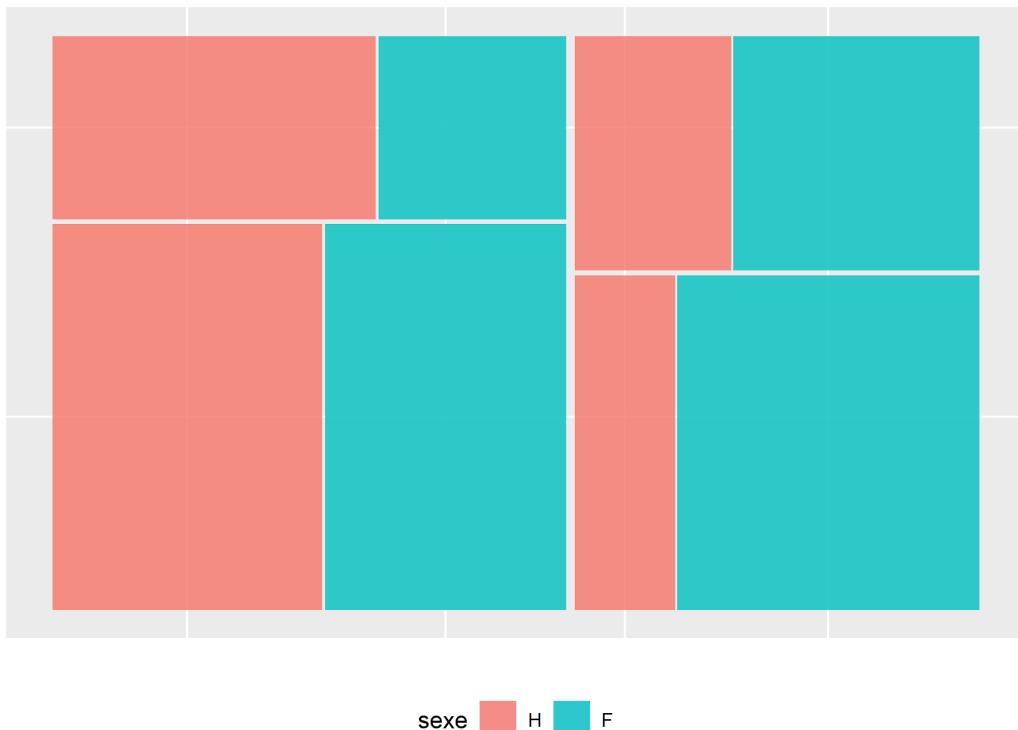


Figure 31. Graphique en mosaïque

## Données labellisées et ggplot2

`ggplot2` tient compte du type des variables, attendant à ce que les variables catégorielles soient présentées sous forme de facteurs. Si l'on utilise des données labellisées (voir le chapitre dédié, page 123), nos variables catégorielles seront stockées sous la forme d'un vecteur numérique avec des étiquettes. Il sera donc nécessaire de convertir ces variables en facteurs, tout simplement avec la fonction `to_factor`

de l’extension **labelled** qui pourra utiliser les étiquettes de valeurs comme modalités du facteur.

## Exporter les graphiques obtenus

Les graphiques produits par **ggplot2** peuvent être sauvegardés manuellement, comme expliqué dans le chapitre «Export des graphiques, page 331», ou programmatiquement. Pour sauvegarder le dernier graphique affiché par **ggplot2** au format PNG, il suffit d’utiliser la fonction **ggsave**, qui permet d’en régler la taille (en pouces) et la résolution (en pixels par pouce ; 72 par défaut) :

```
R> ggsave("mon_graphique.png", width = 11, height = 8)
```

De la même manière, pour sauvegarder n’importe quel graphique construit avec **ggplot2** et stocké dans un objet, il suffit de préciser le nom de cet objet, comme ci-dessous, où l’on sauvegarde le graphique contenu dans l’objet **p** au format vectoriel PDF, qui préserve la netteté du texte et des autres éléments du graphique à n’importe quelle résolution d’affichage :

```
R> ggsave("mon_graphique.pdf", plot = p,
           width = 11, height = 8)
```

# Données pondérées

Options de certaines fonctions .....	466
Données pondérées avec l'extension survey .....	466
gtsummary et survey .....	469
Graphiques natifs avec survey .....	471
Graphiques ggplot2 .....	472
Extraire un sous-échantillon .....	475
Modèles logistiques .....	475
dplyr et survey .....	475
Conclusion .....	475

S'il est tout à fait possible de travailler avec des données pondérées sous R, cette fonctionnalité n'est pas aussi bien intégrée que dans la plupart des autres logiciels de traitement statistique. En particulier, il y a plusieurs manières possibles de gérer la pondération. Cependant, lorsque l'on doit également prendre un compte un plan d'échantillonnage complexe (voir section dédiée ci-après), R fournit tous les outils nécessaires, alors que dans la plupart des logiciels propriétaires, il faut disposer d'une extension adéquate, pas toujours vendue de base avec le logiciel.

Dans ce qui suit, on utilisera le jeu de données tiré de l'enquête *Histoire de vie* et notamment sa variable de pondération *poids*<sup>1</sup>.

```
R> library(questionr)
data(hdv2003)
d <- hdv2003
range(d$poids)
```

```
[1] 78.07834 31092.14132
```

---

1. On notera que cette variable est utilisée à titre purement illustratif. Le jeu de données étant un extrait d'enquête et la variable de pondération n'ayant pas été recalculée, elle n'a ici à proprement parler aucun sens.

## Options de certaines fonctions

Tout d’abord, certaines fonctions de R acceptent en argument un vecteur permettant de pondérer les observations (l’option est en général nommée `weights` ou `row.w`). C’est le cas par exemple des méthodes d’estimation de modèles linéaires<sup>2</sup> (`lm`) ou de modèles linéaires généralisés<sup>3</sup> (`glm`) ou dans les analyses de correspondances<sup>4</sup> des extensions `ade4` ou `FactoMineR`.

Par contre cette option n’est pas présente dans les fonctions de base comme `mean`, `var`, `table` ou `chisq.test`.

## Données pondérées avec l’extension survey

L’extension `survey` est spécialement dédiée au traitement d’enquêtes ayant des techniques d’échantillonnage et de pondération potentiellement très complexes.

L’extension s’installe comme la plupart des autres :

```
R> install.packages("survey")
```

Le site officiel (en anglais) comporte beaucoup d’informations, mais pas forcément très accessibles : <http://r-survey.r-forge.r-project.org/>.

Pour utiliser les fonctionnalités de l’extension, on doit d’abord définir le plan d’échantillonnage ou *design* de notre enquête, c’est-à-dire indiquer quel type de pondération nous souhaitons lui appliquer.

Dans un premier temps, nous utiliserons le plan d’échantillonnage le plus simple, avec une variable de pondération déjà calculée. Pour d’autres types de plan d’échantillonnage, voir la chapitre sur les plans d’échantillonnage complexes, page 501.

Ceci se fait à l’aide de la fonction `svydesign` :

```
R> library(survey)
dw <- svydesign(ids = ~1, data = d, weights = ~ d$poids)
```

Cette fonction crée un nouvel objet, que nous avons nommé `dw`. Cet objet n’est pas à proprement parler un tableau de données, mais plutôt un tableau de données plus une méthode de pondération. `dw` et

- 
2. Voir le chapitre régression linéaire, page 505.
  3. Voir le chapitre sur la régression logistique, page 507.
  4. Voir le chapitre dédié à l’analyse des correspondances, page 557.

`d` sont des objets distincts, les opérations effectuées sur l'un n'ont pas d'influence sur l'autre. On peut cependant retrouver le contenu de `d` depuis `dw` en utilisant `dw$variables` :

```
R> str(d$age)
```

```
int [1:2000] 28 23 59 34 71 35 60 47 20 28 ...
```

```
R> str(dw$variables$age)
```

```
int [1:2000] 28 23 59 34 71 35 60 47 20 28 ...
```

Lorsque notre plan d'échantillonnage est déclaré, on peut lui appliquer une série de fonctions permettant d'effectuer diverses opérations statistiques en tenant compte de la pondération. On citera notamment :

- `svymean`, `svyvar`, `svytotal`, `svyquantile` : statistiques univariées (moyenne, variance, total, quantiles)
- `svytable` : tri à plat et tableau croisé
- `svychisq` : test du  $\chi^2$
- `svyby` : statistiques selon un facteur
- `svyttest` : test t de Student de comparaison de moyennes
- `svyciprop` : intervalle de confiance d'une proportion
- `svyglm` : modèles linéaires généralisés (dont régression logistique)
- `svyplot`, `svyhist`, `svyboxplot` : fonctions graphiques

D'autres fonctions sont disponibles, comme `svyratio`, mais elles ne seront pas abordées ici.

Pour ne rien arranger, ces fonctions prennent leurs arguments sous forme de formules<sup>5</sup>, c'est-à-dire pas de la manière habituelle. En général l'appel de fonction se fait en spécifiant d'abord les variables d'intérêt sous forme de formule, puis l'objet `survey.design`.

Voyons tout de suite quelques exemples<sup>6</sup> :

```
R> svymean(~age, dw)
```

```
mean      SE
age 46.347 0.5284
```

5. Pour plus de détails sur les formules, voir le chapitre dédié, page 929.

6. Pour d'autres exemples, voir [http://www.ats.ucla.edu/stat/r/faq/svy\\_r\\_oscluster.htm](http://www.ats.ucla.edu/stat/r/faq/svy_r_oscluster.htm) (en anglais).

```
R> svyquantile(~age, dw, quantile = c(0.25, 0.5, 0.75), ci = TRUE)
```

```
$quantiles  
 0.25 0.5 0.75  
age   31  45  60  
  
$CIs  
, , age  
  
 0.25 0.5 0.75  
(lower  30  43  58  
upper)  32  47  62
```

```
R> svyvar(~heures.tv, dw, na.rm = TRUE)
```

```
  variance      SE  
heures.tv  2.9886 0.1836
```

Les tris à plat se déclarent en passant comme argument le nom de la variable précédé d'un tilde (~), tandis que les tableaux croisés utilisent les noms des deux variables séparés par un signe plus (+) et précédés par un tilde (~).

```
R> svytable(~sexe, dw)
```

```
sexe  
Homme    Femme  
5149382 5921844
```

```
R> svytable(~ sexe + clso, dw)
```

```
clso  
sexe          Oui        Non Ne sait pas  
Homme 2658744.04 2418187.64    72450.75  
Femme 2602031.76 3242389.36    77422.79
```

La fonction `freq` peut être utilisée si on lui passe en argument non pas la variable elle-même, mais son tri à plat obtenu avec `svytable` :

```
R> tab <- svytable(~peche.chasse, dw)  
freq(tab, total = TRUE)
```

On peut également récupérer le tableau issu de `svytable` dans un objet et le réutiliser ensuite comme n'importe quel tableau croisé :

```
R> tab <- svytable(~ sexe + cuso, dw)
tab
```

		cuso		
sexe		Oui	Non	Ne sait pas
Homme		2658744.04	2418187.64	72450.75
Femme		2602031.76	3242389.36	77422.79

Les fonctions `lprop` et `cprop` de `questionr` sont donc tout à fait compatibles avec l'utilisation de `survey`.

```
R> lprop(tab)
```

		cuso			Total
sexe		Oui	Non	Ne sait pas	
Homme		51.6	47.0	1.4	100.0
Femme		43.9	54.8	1.3	100.0
Ensemble		47.5	51.1	1.4	100.0

Le principe de la fonction `svyby` est similaire à celui de `tapply`<sup>7</sup>. Elle permet de calculer des statistiques selon plusieurs sous-groupes définis par un facteur. Par exemple :

```
R> svyby(~age, ~sexe, dw, svymean)
```

## gtsummary et survey

L'extension `gtsummary` fournit une fonction `tbl_svysummary`, similaire à `tbl_summary`, mais adaptée aux objets `survey`.

```
R> library(gtsummary)
theme_gtsummary_language("fr", decimal.mark = ",",
big.mark = " ")
```

```
Setting `language: fr` theme
```

---

7. La fonction `tapply` est présentée plus en détails dans le chapitre [Manipulation de données](#).

```
R> dw %>%tbl_svysummary(include = c("age", "sexe", "clso", "peche.chasse"))
```

Caractéristique	N = 11 071 226 <sup>1</sup>
age	45 (31 – 60)
sexe	
Homme	5 149 382 (47%)
Femme	5 921 844 (53%)
clso	
Oui	5 260 776 (48%)
Non	5 660 577 (51%)
Ne sait pas	149 874 (1,4%)
peche.chasse	
Non	9 716 683 (88%)
Oui	1 354 544 (12%)

<sup>1</sup> Statistique présentée: Médiane (El); n (%)

```
R> dw %>%
 tbl_svysummary(
  include = c("age", "sexe", "clso", "peche.chasse"),
  by = "sexe"
) %>%
add_overall(last = TRUE) %>%
add_p()
```

Caractéristique	Homme, N = 5 149 382 <sup>1</sup>	Femme, N = 5 921 844 <sup>1</sup>	Total, N = 11 071 226 <sup>1</sup>	p-value <sup>2</sup>
age	44 (30 – 59)	45 (32 – 61)	45 (31 – 60)	0,067
clso				0,036
Oui	2 658 744 (52%)	2 602 032 (44%)	5 260 776 (48%)	
Non	2 418 188 (47%)	3 242 389 (55%)	5 660 577 (51%)	
Ne sait pas	72 451 (1,4%)	77 423 (1,3%)	149 874 (1,4%)	
peche.chasse				<0,001
Non	4 101 242 (80%)	5 615 441 (95%)	9 716 683 (88%)	
Oui	1 048 141 (20%)	306 403 (5,2%)	1 354 544 (12%)	

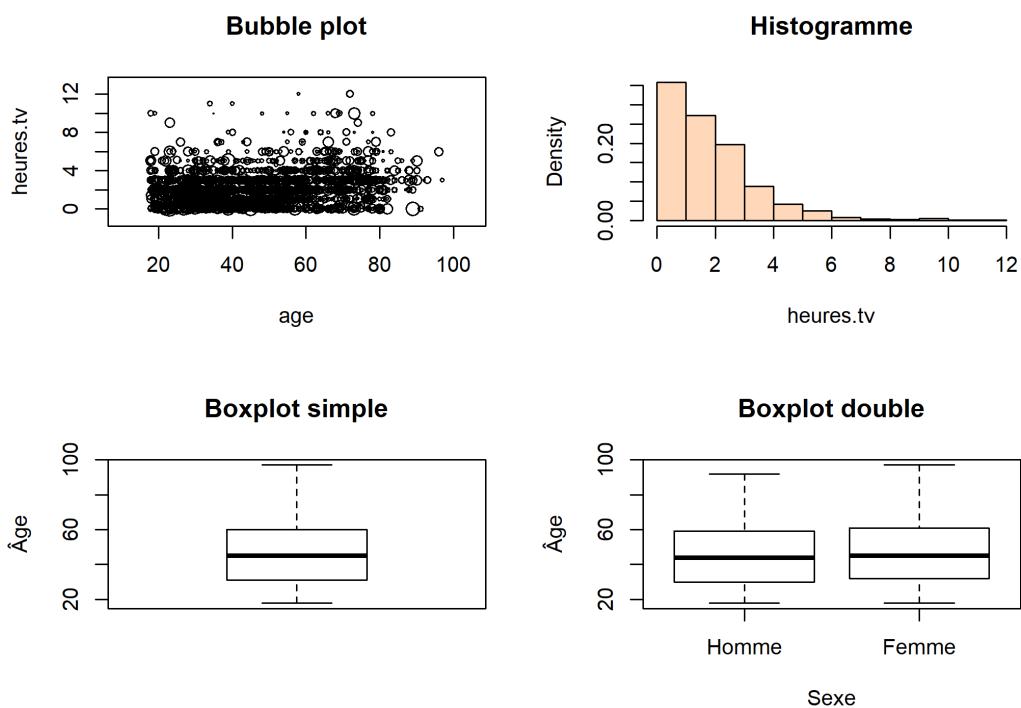
<sup>1</sup> Statistique présentée: Médiane (El); n (%)

<sup>2</sup> Test statistique réalisé: Wilcoxon rank-sum test for complex survey samples; chi-squared test with Rao & Scott's second-order correction

## Graphiques natifs avec survey

[survey](#) est également capable de produire des graphiques à partir des données pondérées. Quelques exemples :

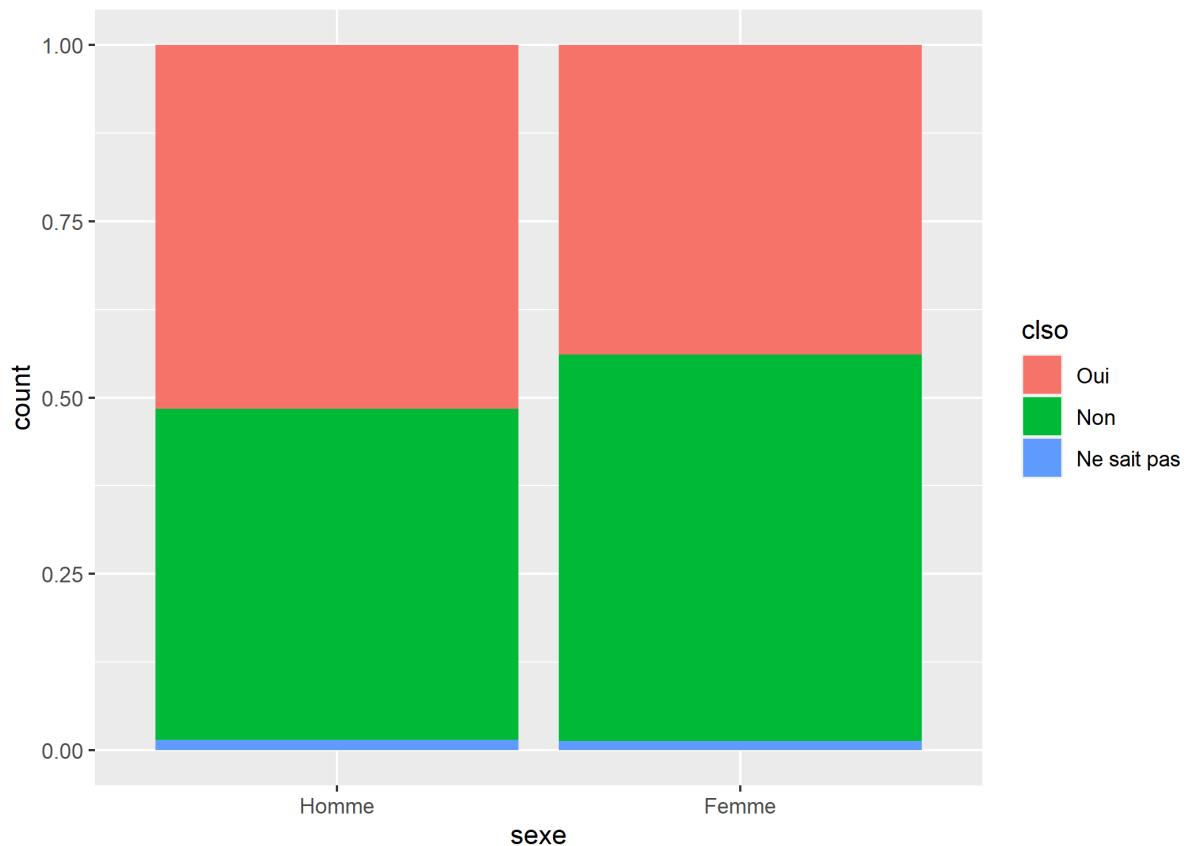
```
R> par(mfrow = c(2, 2))
svyplot(~ age + heures.tv, dw, col = "red", main = "Bubble plot")
svyhist(~heures.tv, dw, col = "peachpuff", main = "Histogramme")
svyboxplot(age ~ 1, dw, main = "Boxplot simple", ylab = "Âge")
svyboxplot(age ~ sexe, dw, main = "Boxplot double", ylab = "Âge", xlab =
"Sexe")
```



## Graphiques ggplot2

`ggplot2` accepte une esthétique `weight` pour indiquer des poids à prendre en compte dans les différents graphiques. La fonction `weights` permet justement de récupérer les poids d'un objet `survey`. La fonction `ggplot` n'accepte pas d'objet `survey` mais a besoin d'un tableau de données. Ce dernier peut être récupéré avec `$variables`.

```
R> library(ggplot2)
ggplot(dw$variables) +
  aes(weight = weights(dw), x = sexe, fill = cuso) +
  geom_bar(position = "fill")
```

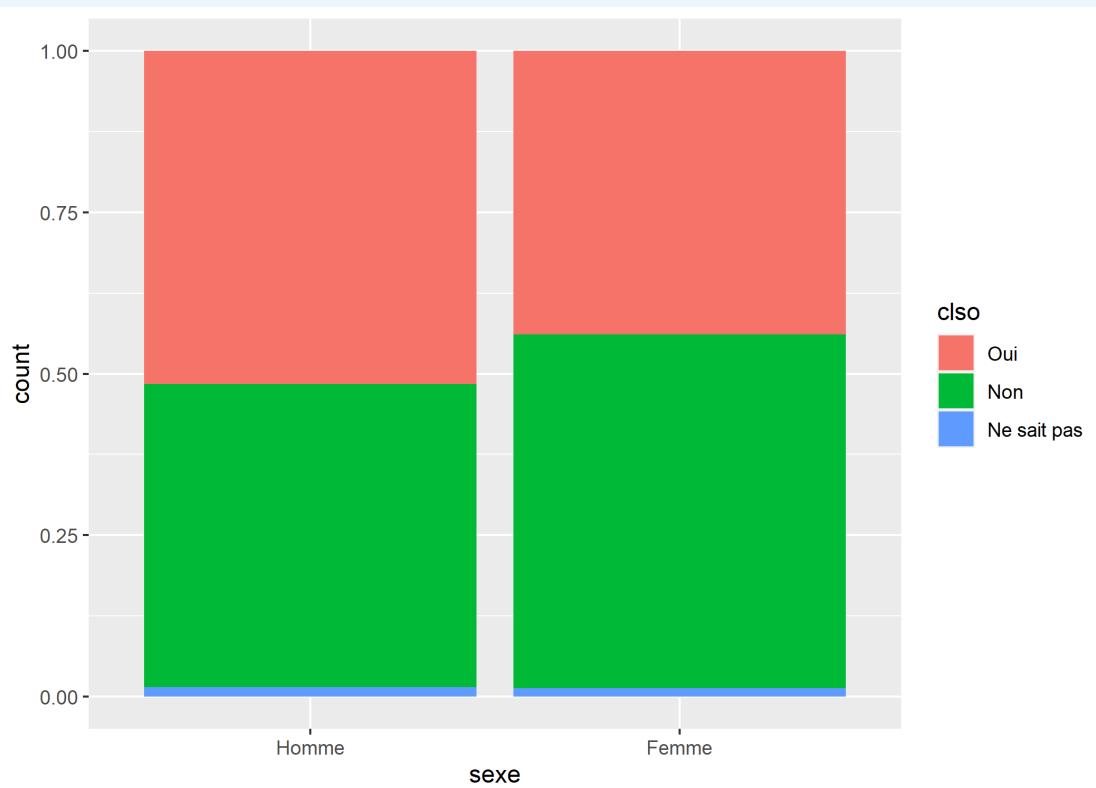


**ATTENTION :** les graphiques obtenus ne sont corrects qu'à la condition que seuls les poids soient nécessaires pour les construire, ce qui est le cas d'un nuage de points ou d'un diagramme en barres. Par contre, si le calcul du graphique implique le calcul de variance, la représentation sera incorrecte. Par exemple, avec `geom_smooth`, les aires de confiance affichées ne prendront pas correctement en compte le plan d'échantillonnage.

### NOTE

L’extension `questionr` propose, dans sa version de développement, une fonction `ggsurvey` pour faciliter les choses. Elle prend un objet `survey`, extrait le tableau de données et les poids, associe les poids à l’esthétique correspondante et appelle `ggplot`.

```
R> library(questionr)
  ggsurvey(dw) +
    aes(x = sexe, fill = cuso) +
    geom_bar(position = "fill")
```



Pour installer la version de développement de `questionr`, on pourra utiliser la commande suivante :

```
R> if (!require(devtools)) {
  install.packages("devtools")
  library(devtools)
}
install_github("juba/questionr")
```

## Extraire un sous-échantillon

Si l'on souhaite travailler sur un sous-échantillon tout en gardant les informations d'échantillonnage, on utilisera la fonction `subset` présentée en détail dans le chapitre Sous-ensembles, page 261.

```
R> sous <- subset(dw, sexe == "Femme" & age >= 40)
```

## Modèles logistiques

Pour réaliser des modèles logistiques (binaires, multinomiaux ou ordinaux) avec prise en compte d'un plan d'échantillonnage, on pourra se référer à la sous-section dédiée du chapitre Régression logistique, page 0.

## dplyr et survey

L'extension `srvyr` vise à permettre d'utiliser les verbes de `dplyr` avec `survey`. Le fonctionnement de cette extension est expliqué dans une vignette dédiée : <https://cran.r-project.org/web/packages/srvyr/vignettes/srvyr-vs-survey.html>.

## Conclusion

Si, la gestion de la pondération sous R n'est sans doute pas ce qui se fait de plus pratique et de plus simple, on pourra quand même donner les conseils suivants :

- utiliser les options de pondération des fonctions usuelles ou les fonctions d'extensions comme `questionr` pour les cas les plus simples ;
- si on utilise `survey`, effectuer autant que possible tous les recodages et manipulations sur les données non pondérées ;
- une fois les recodages effectués, on déclare le design et on fait les analyses en tenant compte de la pondération ;
- surtout ne jamais modifier les variables du design. Toujours effectuer recodages et manipulations sur les données non pondérées, puis redéclarer le design pour que les mises à jour effectuées soient disponibles pour l'analyse.



# Intervalles de confiance

Intervalle de confiance d'une moyenne . . . . .	477
Intervalle de confiance d'une proportion . . . . .	478
Données pondérées et l'extension survey . . . . .	482

Nous utiliserons dans ce chapitre les données de l'enquête *Histoire de vie 2003* fournies avec l'extension **questionr**.

```
R> library(questionr)
  data("hdv2003")
  d <- hdv2003
```

## Intervalle de confiance d'une moyenne

L'intervalle de confiance d'une moyenne peut être calculé avec la fonction **t.test** (fonction qui permet également de réaliser un test t de Student comme nous le verrons dans le chapitre dédié aux comparaisons de moyennes, page 485) :

```
R> t.test(d$heures.tv)
```

```
One Sample t-test

data: d$heures.tv
t = 56.505, df = 1994, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
2.168593 2.324540
sample estimates:
mean of x
2.246566
```

Le niveau de confiance peut être précisé via l’argument `conf.level` :

```
R> t.test(d$heures.tv, conf.level = 0.9)
```

```
One Sample t-test

data: d$heures.tv
t = 56.505, df = 1994, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
90 percent confidence interval:
2.181138 2.311995
sample estimates:
mean of x
2.246566
```

Le nombre d’heures moyennes à regarder la télévision parmi les enquêtés s’avère être de 2,2 heures, avec un intervalle de confiance à 95 % de [2,17 - 2,33] et un intervalle de confiance à 90 % de [2,18 - 2,31].

## Intervalle de confiance d’une proportion

La fonction `prop.test` permet de calculer l’intervalle de confiance d’une proportion. Une première possibilité consiste à lui transmettre une table à une dimension et deux entrées. Par exemple, si l’on s’intéresse à la proportion de personnes ayant pratiqué une activité physique au cours des douze derniers mois :

```
R> freq(d$sport)
```

```
R> prop.test(table(d$sport))
```

```
1-sample proportions test with continuity correction

data: table(d$sport), null probability 0.5
X-squared = 152.9, df = 1, p-value < 2.2e-16
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
0.6169447 0.6595179
sample estimates:
p
0.6385
```

On remarquera que la fonction a calculé l’intervalle de confiance correspondant à la première entrée du

tableau, autrement dit celui de la proportion d'enquêtés n'ayant pas pratiqué une activité sportive. Or, nous sommes intéressé par la proportion complémentaire, à savoir celle d'enquêtés ayant pratiqué une activité sportive. On peut dès lors modifier l'ordre de la table en indiquant notre modalité d'intérêt avec la fonction `relevel` ou bien indiquer à `prop.test` d'abord le nombre de succès puis l'effectif total :

```
R> prop.test(table(relevel(d$sport, "Oui")))
```

```
1-sample proportions test with continuity correction
```

```
data: table(relevel(d$sport, "Oui")), null probability 0.5
X-squared = 152.9, df = 1, p-value < 2.2e-16
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.3404821 0.3830553
sample estimates:
      p
 0.3615
```

```
R> prop.test(sum(d$sport == "Oui"), length(d$sport))
```

```
1-sample proportions test with continuity correction
```

```
data: sum(d$sport == "Oui") out of length(d$sport), null probability 0.5
X-squared = 152.9, df = 1, p-value < 2.2e-16
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.3404821 0.3830553
sample estimates:
      p
 0.3615
```

Enfin, le niveau de confiance peut être modifié via l'argument `conf.level` :

```
R> prop.test(table(relevel(d$sport, "Oui")), conf.level = 0.9)
```

```
1-sample proportions test with continuity correction
```

```
data: table(relevel(d$sport, "Oui")), null probability 0.5
X-squared = 152.9, df = 1, p-value < 2.2e-16
alternative hypothesis: true p is not equal to 0.5
90 percent confidence interval:
 0.3437806 0.3795989
sample estimates:
```

p  
0.3615

**NOTE**

Il existe de nombreuses manières de calculer un intervalle de confiance pour une proportion. En l’occurrence, l’intervalle calculé par `prop.test` correspond dans le cas présent à un intervalle bilatéral selon la méthode des scores de Wilson avec correction de continuité. Pour plus d’information, on pourra lire <http://joseph.larmarange.net/?Intervalle-de-confiance-bilateral>.

**NOTE**

Pour se simplifier un peu la vie, le package **JLutils** propose une fonction **prop.ci** (et ses deux variantes **prop.ci.lower** et **prop.ci.upper**) permettant d'appeler plus facilement **prop.test** et renvoyant directement l'intervalle de confiance.

**JLutils** n'étant disponible que sur [GitHub](#), on aura recours au package **devtools** et à sa fonction **install\_github** pour l'installer :

```
R> library(devtools)
    install_github("larmarange/JLutils")
```

**prop.ci** fonction accepte directement un tri à plat obtenu avec **table**, un vecteur de données, un vecteur logique (issu d'une condition), ou bien le nombre de succès et le nombre total d'essais. Voir les exemples ci-après :

```
R> library(JLutils)
    freq(d$sport)
```

```
R> prop.ci(d$sport)
```

```
[1] 0.6169447 0.6595179
```

```
R> prop.ci.lower(d$sport)
```

```
[1] 0.6169447
```

```
R> prop.ci.upper(d$sport)
```

```
[1] 0.6595179
```

```
R> prop.ci(d$sport, conf.level = 0.9)
```

```
[1] 0.6204011 0.6562194
```

```
R> prop.ci(table(d$sport))
```

```
[1] 0.6169447 0.6595179
```

```
R> prop.ci(d$sport == "Non")
```

```
[1] 0.6169447 0.6595179
```

```
R> prop.ci(d$sport == "Oui")
```

```
[1] 0.3404821 0.3830553
```

```
R> prop.ci.lower(c(1277, 723), n = 2000)
```

```
[1] 0.6169447 0.3404821
```

```
R> prop.ci.upper(c(1277, 723), n = 2000)
```

```
[1] 0.6595179 0.3830553
```

## Données pondérées et l’extension survey

Lorsque l’on utilise des données pondérées définies à l’aide de l’extension [survey](#)<sup>1</sup>, l’intervalle de confiance d’une moyenne s’obtient avec `confint` et celui d’une proportion avec `svyciprop`.

Quelques exemples :

---

1. Voir le chapitre dédié aux données pondérées, page 466.

```
R> library(survey)
dw <- svydesign(ids = ~1, data = d, weights = ~poids)
svymean(~age, dw)
```

```
mean      SE
age 46.347 0.5284
```

```
R> confint(svymean(~age, dw)) # Intervalle de confiance d'une moyenne
```

```
2.5 %   97.5 %
age 45.3117 47.38282
```

```
R> confint(svyby(~age, ~sexe, dw, svymean)) # Intervalles de confiance pour chaque sexe
```

```
2.5 %   97.5 %
Homme 43.74781 46.65618
Femme 45.88867 48.79758
```

```
R> freq(svytable(~sexe, dw))
```

```
R> svyciprop(~sexe, dw) # Intervalle de confiance d'une proportion
```

```
2.5% 97.5%
sexe 0.535 0.507 0.56
```



# Comparaisons (moyennes et proportions)

Comparaison de moyennes . . . . .	485
Comparaison de proportions . . . . .	490
<sup>2</sup> et dérivés . . . . .	493
Données pondérées et l'extension survey . . . . .	495
Tests dans les tableaux de <code>gtsummary</code> . . . . .	497

## WEBIN-R

Ce chapitre est évoqué dans le webin-R #03 (statistiques descriptives avec `gtsummary` et `esquisse`) sur [YouTube](#).

Nous utiliserons dans ce chapitre les données de l'enquête *Histoire de vie 2003* fournies avec l'extension `questionr`.

```
R> library(questionr)
  data("hdv2003")
  d <- hdv2003
```

## Comparaison de moyennes

On peut calculer la moyenne d'âge des deux groupes en utilisant la fonction `tapply`<sup>1</sup>:

---

1. La fonction `tapply` est présentée plus en détails dans le chapitre [Manipulation de données](#).

```
R> tapply(d$age, d$hard.rock, mean)
```

Non	Oui
48.30211	27.57143

L’écart est important. Est-il statistiquement significatif ? Pour cela on peut faire un test t de Student de comparaison de moyennes à l’aide de la fonction `t.test` :

```
R> t.test(age ~ hard.rock, data = d)
```

```
Welch Two Sample t-test

data: age by hard.rock
t = 9.6404, df = 13.848, p-value = 1.611e-07
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
16.11379 25.34758
sample estimates:
mean in group Non mean in group Oui
48.30211           27.57143
```

Le test est extrêmement significatif. L’intervalle de confiance à 95 % de la différence entre les deux moyennes va de 16,1 ans à 25,3 ans.

#### NOTE

La valeur affichée pour  $p$  est de `1.611e-07`. Cette valeur peut paraître étrange pour les non avertis. Cela signifie tout simplement 1,611 multiplié par 10 à la puissance -7, autrement dit 0,0000001611. Cette manière de représenter un nombre est couramment appelée notation scientifique.

Pour plus de détails, voir [http://fr.wikipedia.org/wiki/Notation\\_scientifique](http://fr.wikipedia.org/wiki/Notation_scientifique).

Il est possible de désactiver la notation scientifique avec la commande :

```
R> options(scipen = 999)
```

Pour rétablir la notation scientifique :

```
R> options(scipen = 0)
```

Nous sommes cependant allés un peu vite en besogne, car nous avons négligé une hypothèse

fondamentale du test  $t$  : les ensembles de valeur comparés doivent suivre approximativement une loi normale et être de même variance<sup>2</sup>.

#### NOTE

Dans le test de Student, on suppose l'égalité des variances parentes, ce qui permet de former une estimation commune de la variance des deux échantillons (on parle de *pooled variance*), qui revient à une moyenne pondérée des variances estimées à partir des deux échantillons. Dans le cas où l'on souhaite relaxer cette hypothèse, le test de Welch ou la correction de Satterthwaite reposent sur l'idée que l'on utilise les deux estimations de variance séparément, suivie d'une approximation des degrés de liberté pour la somme de ces deux variances. Le même principe s'applique dans le cas de l'analyse de variance à un facteur (cf. `oneway.test`).

Comment vérifier que l'hypothèse de normalité est acceptable pour ces données ? D'abord avec un petit graphique composé de deux histogrammes :

---

2. Concernant cette seconde condition, `t.test` utilise par défaut un test de Welch qui ne suppose pas l'égalité des variances parentes ; il est toutefois possible d'utiliser le test classique de Student en spécifiant l'option `var.equal = TRUE`.

```
R> par(mfrow = c(1, 2))
hist(d$age[d$hard.rock == "Oui"], main = "Hard rock", col = "red")
hist(d$age[d$hard.rock == "Non"], main = "Sans hard rock", col = "red")
```

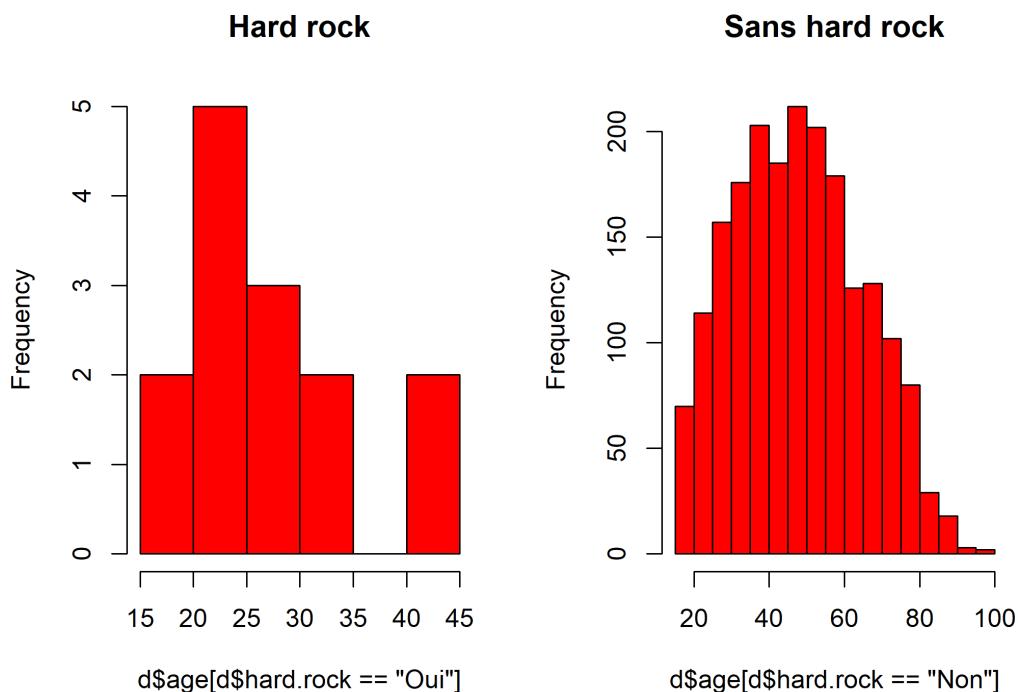


Figure 1. Distribution des âges pour appréciation de la normalité

Une alternative consisterait à utiliser des graphiques de type QQ-plot, à l'aide de la fonction `qnorm`, même si leur utilisation et leur interprétation ne sera pas détaillée ici.

#### NOTE

La fonction `par` permet de modifier de nombreux paramètres graphiques. Ici, l'instruction `par(mfrow = c(1, 2))` sert à indiquer que l'on souhaite afficher deux graphiques sur une même fenêtre, plus précisément que la fenêtre doit comporter une ligne et deux colonnes.

Ça a l'air à peu près bon pour les « Sans hard rock », mais un peu plus limite pour les fans de Metallica, dont les effectifs sont d'ailleurs assez faibles. Si on veut en avoir le cœur net on peut utiliser le test de normalité de Shapiro-Wilk avec la fonction `shapiro.test` :

```
R> shapiro.test(d$age[d$hard.rock == "Oui"])
```

Shapiro-Wilk normality test

```
data: d$age[d$hard.rock == "Oui"]
W = 0.86931, p-value = 0.04104
```

```
R> shapiro.test(d$age[d$hard.rock == "Non"])
```

Shapiro-Wilk normality test

```
data: d$age[d$hard.rock == "Non"]
W = 0.98141, p-value = 2.079e-15
```

Visiblement, le test estime que les distributions ne sont pas suffisamment proches de la normalité dans les deux cas.

Et concernant l'égalité des variances ?

```
R> tapply(d$age, d$hard.rock, var)
```

	Non	Oui
285.62858	62.72527	

L'écart n'a pas l'air négligeable. On peut le vérifier avec le test d'égalité des variances fourni par la fonction

`var.test` :

```
R> var.test(age ~ hard.rock, data = d)
```

F test to compare two variances

```
data: age by hard.rock
F = 4.5536, num df = 1985, denom df = 13, p-value =
0.003217
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
1.751826 8.694405
sample estimates:
ratio of variances
4.553644
```

La différence est très significative. En toute rigueur le test t n'aurait donc pas pu être utilisé. Cela dit, il convient de rappeler que ce test statistique (1) suppose la normalité des distributions et (2) considère comme hypothèse nulle l'égalité des variances (parentes) – ce que l'on souhaiterait vérifier alors qu'on ne peut pas accepter l'hypothèse nulle dans un cadre d'inférence fréquentiste – sans que l'on définisse réellement ce que signifie des variances différentes sur le plan pratique. Est-ce qu'une variation de la variance du simple au double est pertinente au regard du domaine d'étude, ou bien faut-il décider qu'à partir d'un rapport de 4 on peut considérer qu'il y a bien une différence importante entre deux variances ? Sans avoir fixé au préalable cette hypothèse alternative, on ne peut guère conclure à partir de ce test. Une alternative consiste à comparer la forme des distributions à l'aide, par exemple, de diagrammes de type boîtes à moustaches.

*Damned ! Ces maudits tests statistiques vont-ils nous empêcher de faire connaître au monde entier notre fabuleuse découverte sur l'âge des fans de Sepultura ? Non ! Car voici qu'approche à l'horizon un nouveau test, connu sous le nom de Wilcoxon/Mann-Whitney. Celui-ci a l'avantage d'être non-paramétrique, c'est à dire de ne faire aucune hypothèse sur la distribution des échantillons comparés, à l'exception que celles-ci ont des formes à peu près comparables (essentiellement en termes de variance). Attention, il ne s'agit pas d'un test comparant les différences de médianes (pour cela il existe le test de Mood) mais d'un test reposant sur la somme des rangs des observations, au lieu des valeurs brutes, dans les deux groupes :*

```
R> wilcox.test(age ~ hard.rock, data = d)
```

```
Wilcoxon rank sum test with continuity correction

data: age by hard.rock
W = 23980, p-value = 2.856e-06
alternative hypothesis: true location shift is not equal to 0
```

Ouf ! La différence est hautement significative<sup>3</sup>. Nous allons donc pouvoir entamer la rédaction de notre article pour la *Revue française de sociologie*.

## Comparaison de proportions

La fonction `prop.test`, que nous avons déjà rencontrée pour calculer l'intervalle de confiance d'une proportion (voir le chapitre dédié aux intervalles de confiance, page 477) permet également d'effectuer un test de comparaison de deux proportions.

Supposons que l'on souhaite comparer la proportion de personnes faisant du sport entre ceux qui lisent des bandes dessinées et les autres :

---

3. Ce test peut également fournir un intervalle de confiance avec l'option `conf.int=TRUE`.

```
R> tab <- xtabs(~ lecture.bd + sport, data = d)
lprop(tab)
```

		sport		
		lecture.bd	Oui	Total
		Non	64.2	35.8
		Oui	48.9	51.1
		Ensemble	63.8	36.1
				100.0

Une représentation graphique sous forme de diagramme en barres peut être définie comme suit :

```
R> barplot(prop.table(tab, margin = 1) * 100, beside = TRUE, ylim = c(0, 100),
0), xlab = "Sport", legend.text = c("Lecture : non", "Lecture : oui"))
```

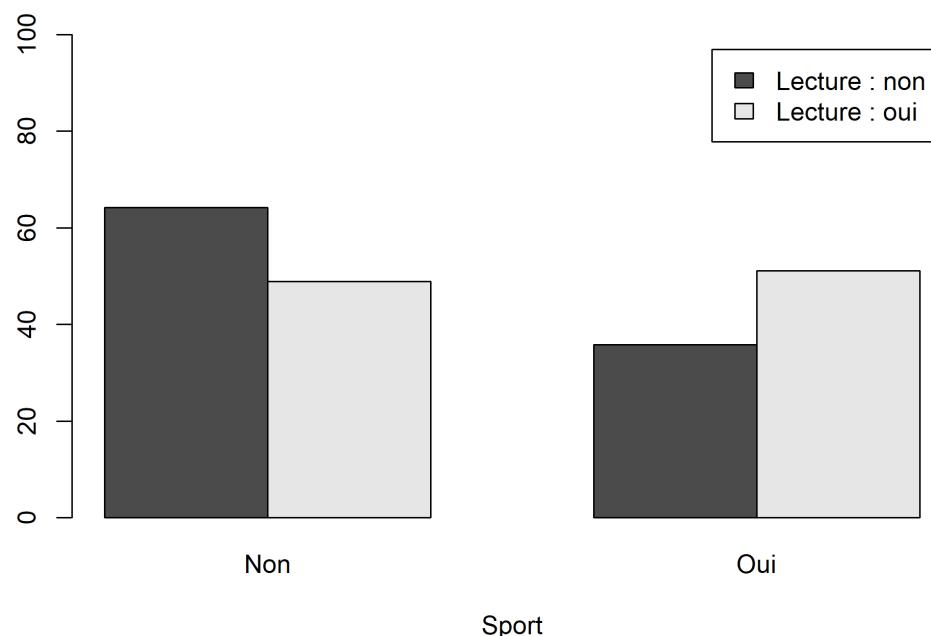


Figure 2. Répartition des individus (en %) selon les variables sport et lecture

Il suffit de transmettre notre tableau croisé (à 2×2 dimensions) à `prop.test` :

```
R> prop.test(tab)
```

```
2-sample test for equality of proportions with  
continuity correction  
  
data: tab  
X-squared = 4, df = 1, p-value = 0.0455  
alternative hypothesis: two.sided  
95 percent confidence interval:  
-0.002652453 0.308107236  
sample estimates:  
prop 1     prop 2  
0.6420891 0.4893617
```

On pourra également avoir recours à la fonction `fisher.test` qui renverra notamment l'odds ratio et son intervalle de confiance correspondant :

```
R> fisher.test(tab)
```

```
Fisher's Exact Test for Count Data  
  
data: tab  
p-value = 0.0445  
alternative hypothesis: true odds ratio is not equal to 1  
95 percent confidence interval:  
1.003372 3.497759  
sample estimates:  
odds ratio  
1.871433
```

#### NOTE

Formellement, le test de Fisher suppose que les marges du tableau (totaux lignes et colonnes) sont fixées, puisqu'il repose sur une loi hypergéométrique, et donc celui-ci se prête plus au cas des situations expérimentales (plans d'expérience, essais cliniques) qu'au cas des données tirées d'études observationnelles.

On pourra aussi avoir recours à la fonction `odds.ratio` de l'extension `questionr` qui réalise le même calcul mais présente le résultat légèrement différemment :

```
R> odds.ratio(tab)
```

Note : pour le calcul du risque relatif, on pourra regarder du côté de la fonction `relrisk` de l'extension `mosaic`.

## $\chi^2$ et dérivés

Dans le cadre d'un tableau croisé, on peut tester l'existence d'un lien entre les modalités de deux variables, avec le très classique test du  $\chi^2$  de Pearson<sup>4</sup>. Celui-ci s'obtient grâce à la fonction `chisq.test`, appliquée au tableau croisé obtenu avec `table` ou `xtabs`<sup>5</sup> :

```
R> d$qualreg <- as.character(d$qualif)
d$qualreg[d$qualif %in% c("Ouvrier specialise", "Ouvrier qualifie")] <- "Ouvrier"
d$qualreg[d$qualif %in% c(
  "Profession intermediaire",
  "Technicien"
)] <- "Intermediaire"

tab <- table(d$sport, d$qualreg)
tab
```

	Autre	Cadre	Employe	Intermediaire	Ouvrier
Non	38	117	401	127	381
Oui	20	143	193	119	114

```
R> chisq.test(tab)
```

```
Pearson's Chi-squared test
```

```
data: tab
X-squared = 96.798, df = 4, p-value < 2.2e-16
```

Le test est hautement significatif : on ne peut donc pas considérer qu'il y a indépendance entre les lignes et les colonnes du tableau.

4. On ne donnera pas plus d'indications sur le test du  $\chi^2$  ici. Les personnes désirant une présentation plus détaillée pourront se reporter (attention, séance d'autopromotion !) à la page suivante : <http://alea.fr.eu.org/pages/khi2>.

5. On peut aussi appliquer directement le test en spécifiant les deux variables à croiser via `chisq.test(d$qualreg, d$sport)`.

**NOTE**

Notons que l'agrégation des niveaux d'une variable catégorielle peut être réalisée d'une manière différente en utilisant les fonctions de gestion des niveaux d'un facteur. Les expressions précédentes sont donc équivalentes à l'approche ci-après, qui ne nécessite pas de convertir `d$qualif` en chaîne de caractères :

```
R> d$qualreg <- d$qualif
  levels(d$qualreg)[1:2] <- "Ouvrier"
  levels(d$qualreg)[2:3] <- "Intermédiaire"
  tab <- table(d$sport, d$qualreg)
```

On peut affiner l'interprétation du test en déterminant dans quelle cas l'écart à l'indépendance est le plus significatif en utilisant les résidus du test. Ceux-ci sont notamment affichables avec la fonction `chisq.residuals` de `questionr` :

```
R> chisq.residuals(tab)
```

	Autre	Cadre	Employe	Intermediaire	Ouvrier
Non	0.11	-3.89	0.95	-2.49	3.49
Oui	-0.15	5.23	-1.28	3.35	-4.70

Les cases pour lesquelles l'écart à l'indépendance est significatif ont un résidu dont la valeur est supérieure à 2 ou inférieure à -2 (le fameux nombre 2 issu de la loi normale, au-delà duquel on s'attend à observer au maximum 2,5 % des observations). Ici on constate que la pratique d'un sport est sur-représentée parmi les cadres et, à un niveau un peu moindre, parmi les professions intermédiaires, tandis qu'elle est sous-représentée chez les ouvriers.

Enfin, on peut calculer le coefficient de contingence de Cramer du tableau, qui présente l'avantage de pouvoir être comparé par la suite à celui calculé sur d'autres tableaux croisés. On peut pour cela utiliser la fonction `cramer.v` de `questionr` :

```
R> cramer.v(tab)
```

```
[1] 0.24199
```

**NOTE**

Pour un tableau à  $2 \times 2$  entrées, comme discuté plus haut, il est également possible de calculer le test exact de Fisher avec la fonction `fisher.test`. On peut soit lui passer le résultat de `table` ou `xtabs`, soit directement les deux variables à croiser.

```
R> lprop(table(d$sex, d$cuisine))
```

	Non	Oui	Total
Homme	70.0	30.0	100.0
Femme	44.5	55.5	100.0
Ensemble	56.0	44.0	100.0

```
R> fisher.test(table(d$sex, d$cuisine))
```

```
Fisher's Exact Test for Count Data

data: table(d$sex, d$cuisine)
p-value < 2.2e-16
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
2.402598 3.513723
sample estimates:
odds ratio
2.903253
```

Le test du  $\chi^2$  de Pearson étant assez robuste quant aux déviations par rapport aux hypothèses d'applications du test (effectifs théoriques tous = 5), le test de Fisher présente en général peu d'intérêt dans le cas de l'analyse des tableaux de contingence.

## Données pondérées et l'extension survey

Lorsque l'on utilise des données pondérées, on aura recours à l'extension `survey`<sup>6</sup>.

Préparons des données d'exemple :

---

6. Voir le chapitre dédié aux données pondérées, page 466.

```
R> library(survey)
dw <- svydesign(ids = ~1, data = d, weights = ~poids)
```

Pour comparer deux moyennes à l'aide d'un test  $t$  on aura recours à `svyttest` :

```
R> svyttest(age ~ sexe, dw)
```

Design-based t-test

```
data: age ~ sexe
t = 2.0404, df = 1998, p-value = 0.04144
alternative hypothesis: true difference in mean is not equal to 0
95 percent confidence interval:
0.08440815 4.19785019
sample estimates:
difference in mean
2.141129
```

Pour le test de Wilcoxon/Mann-Whitney, on pourra avoir recours à `svyranktest` :

```
R> svyranktest(age ~ hard.rock, dw)
```

Design-based KruskalWallis test

```
data: age ~ hard.rock
t = -11.12, df = 1998, p-value < 2.2e-16
alternative hypothesis: true difference in mean rank score is not equal to 0
sample estimates:
difference in mean rank score
-0.3636859
```

On ne peut pas utiliser `chisq.test` directement sur un tableau généré par `svytable`. Les effectifs étant extrapolés à partir de la pondération, les résultats du test seraient complètement faussés. Si on veut faire un test du  $\chi^2$  sur un tableau croisé pondéré, il faut utiliser `svychisq` :

```
R> rprop(svytable(~ sexe + cuso, dw))
```

		cuso			Total
sexe		Oui	Non	Ne sait pas	
Homme		51.6	47.0	1.4	100.0
Femme		43.9	54.8	1.3	100.0
Ensemble		47.5	51.1	1.4	100.0

```
R> svychisq(~ sexe + clso, dw)
```

Pearson's  $\chi^2$ : Rao & Scott adjustment

```
data: svychisq(~sexe + clso, dw)
F = 3.3331, ndf = 1.9734, ddf = 3944.9024, p-value =
0.03641
```

L'extension `survey` ne propose pas de version adaptée du test exact de Fisher. Pour comparer deux proportions, on aura donc recours au test du  $\chi^2$ :

```
R> rprop(svytable(~ lecture.bd + sport, dw))
```

		sport			
		lecture.bd	Non	Oui	Total
		Non	61.0	39.0	100.0
		Oui	46.8	53.2	100.0
		Ensemble	60.7	39.3	100.0

```
R> svychisq(~ lecture.bd + sport, dw)
```

Pearson's  $\chi^2$ : Rao & Scott adjustment

```
data: svychisq(~lecture.bd + sport, dw)
F = 2.6213, ndf = 1, ddf = 1999, p-value = 0.1056
```

## Tests dans les tableaux de `gtsummary`

Lorsque l'on réalise un tableau croisé avec `tbl_summary` ou `tbl_svysummary` de l'extension `gtsummary`, il est possible d'ajouter des tests de comparaison avec `add_p`.

```
R> library(gtsummary)
theme_gtsummary_language("fr", decimal.mark = ",", big.mark = " ")
```

Setting `language: fr` theme

```
R> d %>%
 tbl_summary(
  include = c("hard.rock", "age", "sport"),
  by = "hard.rock"
) %>%
add_p()
```

<b>Caractéristique</b>	<b>Non, N = 1 986<sup>1</sup></b>	<b>Oui, N = 14<sup>1</sup></b>	<b>p-value<sup>2</sup></b>
age	48 (35 – 60)	26 (22 – 30)	<0,001
sport			>0,9
Non	1 268 (64%)	9 (64%)	
Oui	718 (36%)	5 (36%)	

<sup>1</sup> Statistique présentée: Médiane (EI); n (%)

<sup>2</sup> Test statistique réalisé: test de Wilcoxon-Mann-Whitney; test du khi-deux d'indépendance

Il est possible de préciser le type de test à utiliser.

```
R> d %>%
 tbl_summary(
  include = c("hard.rock", "age"),
  by = "hard.rock"
) %>%
add_p(test = list(all_continuous() ~ "wilcox.test"))
```

<b>Caractéristique</b>	<b>Non, N = 1 986<sup>1</sup></b>	<b>Oui, N = 14<sup>1</sup></b>	<b>p-value<sup>2</sup></b>
age	48 (35 – 60)	26 (22 – 30)	<0,001

<sup>1</sup> Statistique présentée: Médiane (EI)

<sup>2</sup> Test statistique réalisé: test de Wilcoxon-Mann-Whitney

Cela fonctionne également avec les données pondérées et les plans d'échantillonnage complexe.

```
R> dw %>%
 tbl_svysummary(
  include = c("hard.rock", "age", "sport"),
  by = "hard.rock"
) %>%
add_p()
```

Caractéristique	Non, N = 10 981 963 <sup>1</sup>	Oui, N = 89 264 <sup>1</sup>	p-value <sup>2</sup>
age	45 (31 – 60)	23 (19 – 28)	<0,001
sport			0,8
Non	6 656 841 (61%)	57 920 (65%)	
Oui	4 325 122 (39%)	31 344 (35%)	

<sup>1</sup> Statistique présentée: Médiane (EI); n (%)

<sup>2</sup> Test statistique réalisé: Wilcoxon rank-sum test for complex survey samples; chi-squared test with Rao & Scott's second-order correction



# Définir un plan d'échantillonnage complexe

Différents types d'échantillonnage .....	501
Les options de svydesign .....	502
Quelques exemples .....	503
Extraire un sous-échantillon .....	504

L'extension `survey` ne permet pas seulement d'indiquer une variable de pondération mais également de prendre les spécificités du plan d'échantillonnage (strates, grappes, ...). Le plan d'échantillonnage ne joue pas seulement sur la pondération des données, mais influence le calcul des variances et par ricochet tous les tests statistiques. Deux échantillons identiques avec la même variable de pondération mais des designs différents produiront les mêmes moyennes et proportions mais des intervalles de confiance différents.

Le site officiel (en anglais) comporte beaucoup d'informations, mais pas forcément très accessibles :  
<http://r-survey.r-forge.r-project.org/>.

## Différents types d'échantillonnage

L'échantillonnage aléatoire simple ou échantillonnage équiprobable est une méthode pour laquelle tous les échantillons possibles (de même taille) ont la même probabilité d'être choisis et tous les éléments de la population ont une chance égale de faire partie de l'échantillon. C'est l'échantillonnage le plus simple : chaque individu à la même probabilité d'être sélectionné.

L'échantillonnage stratifié est une méthode qui consiste d'abord à subdiviser la population en groupes homogènes (*strates*) pour ensuite extraire un échantillon aléatoire de chaque strate. Cette méthode suppose la connaissance de la structure de la population. Pour estimer les paramètres, les résultats doivent être pondérés par l'importance relative de chaque strate dans la population.

L'échantillonnage par grappes est une méthode qui consiste à choisir un échantillon aléatoire d'unités qui sont elles-mêmes des sous-ensembles de la population (*grappes* ou *clusters* en anglais). Cette méthode

suppose que les unités de chaque grappe sont représentatives. Elle possède l’avantage d’être souvent plus économique.

Il est possible de combiner plusieurs de ces approches. Par exemple, les *Enquêtes Démographiques et de Santé*<sup>1</sup> (EDS) sont des enquêtes stratifiées en grappes à deux degrés. Dans un premier temps, la population est divisée en strates par région et milieu de résidence. Dans chaque strate, des zones d’enquêtes, correspondant à des unités de recensement, sont tirées au sort avec une probabilité proportionnelle au nombre de ménages de chaque zone au dernier recensement de population. Enfin, au sein de chaque zone d’enquête sélectionnée, un recensement de l’ensemble des ménages est effectué puis un nombre identique de ménages par zone d’enquête est tiré au sort de manière aléatoire simple.

## Les options de svydesign

La fonction `svydesign` accepte plusieurs arguments décrits sur sa page d’aide (obtenue avec la commande `?svydesign`).

L’argument `data` permet de spécifier le tableau de données contenant les observations.

L’argument `ids` est obligatoire et spécifie sous la forme d’une formule les identifiants des différents niveaux d’un tirage en grappe. S’il s’agit d’un échantillon aléatoire simple, on entrera `ids=~1`. Autre situation : supposons une étude portant sur la population française. Dans un premier temps, on a tiré au sort un certain nombre de départements français. Dans un second temps, on tire au sort dans chaque département des communes. Dans chaque commune sélectionnée, on tire au sort des quartiers. Enfin, on interroge de manière exhaustive toutes les personnes habitant les quartiers enquêtés. Notre fichier de données devra donc comporter pour chaque observation les variables `id_departement`, `id_commune` et `id_quartier`. On écrira alors pour l’argument `ids` la valeur suivante :

`ids=~id_departement+id_commune+id_quartier`.

Si l’échantillon est stratifié, on spécifiera les strates à l’aide de l’argument `strata` en spécifiant la variable contenant l’identifiant des strates. Par exemple : `strata=~id_strate`.

Il faut encore spécifier les probabilités de tirage de chaque cluster ou bien la pondération des individus. Si l’on dispose de la probabilité de chaque observation d’être sélectionnée, on utilisera l’argument `probs`. Si, par contre, on connaît la pondération de chaque observation (qui doit être proportionnelle à l’inverse de cette probabilité), on utilisera l’argument `weights`.

Si l’échantillon est stratifié, qu’au sein de chaque strate les individus ont été tirés au sort de manière aléatoire et que l’on connaît la taille de chaque strate, il est possible de ne pas avoir à spécifier la probabilité de tirage ou la pondération de chaque observation. Il est préférable de fournir une variable contenant la taille de chaque strate à l’argument `fpc`. De plus, dans ce cas-là, une petite correction sera appliquée au modèle pour prendre en compte la taille finie de chaque strate.

---

1. Vaste programme d’enquêtes réalisées à intervalles réguliers dans les pays du Sud, disponibles sur <http://www.dhsprogram.com/>.

## Quelques exemples

```
R> # Échantillonnage aléatoire simple
plan <- svydesign(ids = ~1, data = donnees)

# Échantillonnage stratifié à un seul niveau (la taille de chaque strate est connue)
plan <- svydesign(ids = ~1, data = donnees, fpc = ~taille)

# Échantillonnage en grappes avec tirages à quatre degrés (departement, commune, quartier, individus). La probabilité de tirage de chaque niveau de cluster est connue.
plan <- svydesign(ids = ~id_departement + id_commune + id_quartier, data = donnees, probs = ~proba_departement + proba_commune + proba_quartier)

# Échantillonnage stratifié avec tirage à deux degrés (clusters et individus). Le poids statistiques de chaque observation est connu.
plan <- svydesign(ids = ~id_cluster, data = donnees, strata = ~id_strate, weights = ~poids)
```

Prenons l'exemple d'une *Enquête Démographique et de Santé*. Le nom des différentes variables est standardisé et commun quelle que soit l'enquête. Nous supposerons que vous avez importé le fichier *individus* dans un tableau de données nommés `eds`. Le poids statistique de chaque individu est fourni par la variable *V005* qui doit au préalable être divisée par un million. Les grappes d'échantillonnage au premier degré sont fournies par la variable *V021* (*primary sample unit*). Si elle n'est pas renseignée, on pourra utiliser le numéro de grappe *V001*. Enfin, le milieu de résidence (urbain / rural) est fourni par *V025* et la région par *V024*. Pour rappel, l'échantillon a été stratifié à la fois par région et par milieu de résidence. Certaines enquêtes fournissent directement un numéro de strate via *V022*. Si tel est le cas, on pourra préciser le plan d'échantillonnage ainsi :

```
R> eds$poids <- eds$V005 / 1000000
design.eds <- svydesign(ids = ~V021, data = eds, strata = ~V022, weights = ~poids)
```

Si *V022* n'est pas fourni mais que l'enquête a bien été stratifiée par région et milieu de résidence (vérifiez toujours le premier chapitre du rapport d'enquête), on pourra créer une variable strate ainsi<sup>2</sup> :

---

2. L'astuce consiste à utiliser `as.integer` pour obtenir le code des facteurs et non leur valeur textuelle. L'addition des deux valeurs après multiplication du code de la région par 10 permet d'obtenir une valeur unique pour chaque combinaison des deux variables. On retransforme le résultat en facteurs puis on modifie les étiquettes des modalités.

```
R> eds$strate <- as.factor(as.integer(eds$V024) * 10 + as.integer(eds$V025))
  levels(eds$strate) <- c(paste(levels(eds$V024), "Urbain"), paste(levels(eds$V024), "Rural"))
  design.eds <- svydesign(ids = ~V021, data = eds, strata = ~strate, weights =
~poids)
```

### IMPORTANT

Il n'est pas ais  de modifier des variables dans un objet `survey.design`. Il est donc pr f rable de proc der   l'ensemble des nettoyages, recodages de variables (et au besoin transformation des vecteurs labellis s en facteur), avant de convertir le tableau de donn es en objet `survey` et de proc der aux analyses.

Une autre possibilit  et d'utiliser l'extension `srvyr` qui permet d'utiliser les verbes de `dplyr` avec `survey`. Le fonctionnement de cette extension est expliqu  dans une vignette d di e : <https://cran.r-project.org/web/packages/srvyr/vignettes/srvyr-vs-survey.html>.

## Extraire un sous- chantillon

Si l'on souhaite travailler sur un sous- chantillon tout en gardant les informations d' chantillonnage, on utilisera la fonction `subset` pr sent e en d tail dans le chapitre [Manipulation de donn es](#).

```
R> sous <- subset(plan, sexe == "Femme" & age >= 40)
```

# Régression linéaire

## IMPORTANT

Ce chapitre est en cours d'écriture.

On pourra se référer au chapitre «6 Régressions linéaires avec R» du support de cours d'Ewen Gallic intitulé *Logiciel R et programmation* ([http://egallic.fr/Enseignement/R/m1\\_stat\\_eco\\_logiciel\\_R.pdf](http://egallic.fr/Enseignement/R/m1_stat_eco_logiciel_R.pdf)).



# Régression logistique binaire, multinomiale et ordinale

Préparation des données .....	508
Régression logistique binaire .....	511
Coefficients du modèle .....	514
Représentation graphique du modèle .....	520
Représentation graphique des effets .....	521
Matrice de confusion .....	524
Identifier les variables ayant un effet significatif .....	525
Sélection de modèles .....	528
Tableaux «all-in-one» .....	533
Effets d'interaction dans le modèle .....	555
Multicolinéarité .....	555
Régression logistique multinomiale .....	0
Régression logistique ordinale .....	0
Données pondérées et l'extension survey .....	0
Régression logistique binaire .....	0
Régression multinomiale .....	0
Régression ordinale .....	0

## NOTE

La version originale de ce chapitre a été écrite par Joseph Larmarange dans le cadre du support de cours [Introduction à l'analyse d'enquêtes avec R](#).

La régression logistique est fréquemment utilisée en sciences sociales car elle permet d'effectuer un raisonnement dit *toutes choses étant égales par ailleurs*. Plus précisément, la régression logistique a pour but d'isoler les effets de chaque variable, c'est-à-dire d'identifier les effets résiduels d'une variable

explicative sur une variable d'intérêt, une fois pris en compte les autres variables explicatives introduites dans le modèle. La régression logistique est ainsi prisée en épidémiologie pour identifier les facteurs associés à telle ou telle pathologie.

La régression logistique ordinaire ou régression logistique binaire vise à expliquer une variable d'intérêt binaire (c'est-à-dire de type « oui / non » ou « vrai / faux »). Les variables explicatives qui seront introduites dans le modèle peuvent être quantitatives ou qualitatives.

La régression logistique multinomiale est une extension de la régression logistique aux variables qualitatives à trois modalités ou plus, la régression logistique ordinaire aux variables qualitatives à trois modalités ou plus qui sont ordonnées hiérarchiquement.

## Préparation des données

Dans ce chapitre, nous allons encore une fois utiliser les données de l'enquête *Histoire de vie*, fournies avec l'extension `questionr`.

```
R> library(questionr)
  data(hdv2003)
  d <- hdv2003
```

À titre d'exemple, nous allons étudier l'effet de l'âge, du sexe, du niveau d'étude, de la pratique religieuse et du nombre moyen d'heures passées à regarder la télévision par jour sur le fait de pratiquer un sport.

En premier lieu, il importe de vérifier que notre variable d'intérêt (ici *sport*) est correctement codée. Une possibilité consiste à créer une variable booléenne (vrai / faux) selon que l'individu a pratiqué du sport ou non :

```
R> d$sport2 <- FALSE
  d$sport2[d$sport == "Oui"] <- TRUE
```

Dans le cas présent, cette variable n'a pas de valeur manquante. Mais, le cas échéant, il importe de bien coder les valeurs manquantes en `NA`, les individus en question étant alors exclu de l'analyse.

Il n'est pas forcément nécessaire de transformer notre variable d'intérêt en variable booléenne. En effet, R accepte sans problème une variable de type facteur. Cependant, l'ordre des valeurs d'un facteur a de l'importance. En effet, R considère toujours la première modalité comme étant la modalité de référence. Dans le cas de la variable d'intérêt, la modalité de référence correspond au fait de ne pas remplir le critère étudié, dans notre exemple au fait de ne pas avoir eu d'activité sportive au cours des douze derniers mois.

Pour connaître l'ordre des modalités d'une variable de type facteur, on peut utiliser la fonction `levels` ou bien encore tout simplement la fonction `freq` de l'extension `questionr` :

```
R> levels(d$sport)
```

```
[1] "Non" "Oui"
```

```
R> freq(d$sport)
```

Dans notre exemple, la modalité « Non » est déjà la première modalité. Il n'y a donc pas besoin de modifier notre variable. Si ce n'est pas le cas, il faudra modifier la modalité de référence avec la fonction `relevel` comme nous allons le voir un peu plus loin.

### IMPORTANT

Il est possible d'indiquer un facteur à plus de deux modalités. Dans une telle situation, R considérera que tous les modalités, sauf la modalité de référence, est une réalisation de la variable d'intérêt. Cela serait correct, par exemple, si notre variable *sport* était codée ainsi : « Non », « Oui, toutes les semaines », « Oui, au moins une fois par mois », « Oui, moins d'une fois par mois ». Cependant, afin d'éviter tout risque d'erreur ou de mauvaise interprétation, il est vivement conseillé de recoder au préalable sa variable d'intérêt en un facteur à deux modalités.

La notion de modalité de référence s'applique également aux variables explicatives qualitatives. En effet, dans un modèle, tous les coefficients sont calculés par rapport à la modalité de référence. Il importe de choisir une modalité de référence qui fasse sens afin de faciliter l'interprétation. Par ailleurs, ce choix peut également dépendre de la manière dont on souhaite présenter les résultats. De manière générale on évitera de choisir comme référence une modalité peu représentée dans l'échantillon ou bien une modalité correspondant à une situation atypique.

Prenons l'exemple de la variable *sexe*. Souhaite-t-on connaître l'effet d'être une femme par rapport au fait d'être un homme ou bien l'effet d'être un homme par rapport au fait d'être une femme ? Si l'on opte pour le second, alors notre modalité de référence sera le sexe féminin. Comme est codée cette variable ?

```
R> freq(d$sexe)
```

La modalité « Femme » s'avère ne pas être la première modalité. Nous devons appliquer la fonction `relevel` :

```
R> d$sexe <- relevel(d$sexe, "Femme")
freq(d$sexe)
```

## IMPORTANT

### Données labellisées

Si l’on utilise des données labellisées (voir le chapitre dédié, page 123), nos variables catégorielles seront stockées sous la forme d’un vecteur numérique avec des étiquettes. Il sera donc nécessaire de convertir ces variables en facteurs, tout simplement avec la fonction `to_factor` de l’extension `labelled` qui pourra utiliser les étiquettes de valeurs comme modalités du facteur.

Les variables `age` et `heures.tv` sont des variables quantitatives. Il importe de vérifier qu’elles sont bien enregistrées en tant que variables numériques. En effet, il arrive parfois que dans le fichier source les variables quantitatives soient renseignées sous forme de valeur textuelle et non sous forme numérique.

```
R> str(d$age)
```

```
int [1:2000] 28 23 59 34 71 35 60 47 20 28 ...
```

```
R> str(d$heures.tv)
```

```
num [1:2000] 0 1 0 2 3 2 2.9 1 2 2 ...
```

Nos deux variables sont bien renseignées sous forme numérique.

Cependant, l’effet de l’âge est rarement linéaire. Un exemple trivial est par exemple le fait d’occuper un emploi qui sera moins fréquent aux jeunes âges et aux âges élevés. Dès lors, on pourra transformer la variable `age` en groupe d’âges avec la fonction `cut` (voir le chapitre [Manipulation de données](#)) :

```
R> d$grpage <- cut(d$age, c(16, 25, 45, 65, 99), right = FALSE, include.lowest = TRUE)
R> freq(d$grpage)
```

Jetons maintenant un oeil à la variable `nivetud` :

```
R> freq(d$nivetud)
```

En premier lieu, cette variable est détaillée en pas moins de huit modalités dont certaines sont peu représentées (seulement 39 individus soit 2 % n’ont jamais fait d’études par exemple). Afin d’améliorer notre modèle logistique, il peut être pertinent de regrouper certaines modalités (voir le chapitre [Manipulation de données](#)) :

```
R> d$etud <- d$nivetud
  levels(d$etud) <- c(
    "Primaire", "Primaire", "Primaire",
    "Secondaire", "Secondaire", "Technique/Professionnel",
    "Technique/Professionnel", "Supérieur"
  )
  freq(d$etud)
```

Notre variable comporte également 112 individus avec une valeur manquante. Si nous conservons cette valeur manquante, ces 112 individus seront, par défaut, exclus de l'analyse. Ces valeurs manquantes n'étant pas négligeable (5,6 %), nous pouvons également faire le choix de considérer ces valeurs manquantes comme une modalité supplémentaire. Auquel cas, nous utiliserons la fonction `addNAstr` fournie par `questionr` :

```
R> levels(d$etud)
```

```
[1] "Primaire"           "Secondaire"
[3] "Technique/Professionnel" "Supérieur"
```

```
R> d$etud <- addNAstr(d$etud, "manquant")
  levels(d$etud)
```

```
[1] "Primaire"           "Secondaire"
[3] "Technique/Professionnel" "Supérieur"
[5] "manquant"
```

## Régression logistique binaire

La fonction `glm` (pour *generalized linear models* soit modèle linéaire généralisé en français) permet de calculer une grande variété de modèles statistiques. La régression logistique ordinaire correspond au modèle *logit* de la famille des modèles binomiaux, ce que l'on indique à `glm` avec l'argument `family=binomial(logit)`.

Le modèle proprement dit sera renseigné sous la forme d'une formule (que nous avons déjà rencontrée dans le chapitre sur la statistique bivariée, page 365 et présentée plus en détails dans un chapitre dédié, page 929). On indiquera d'abord la variable d'intérêt, suivie du signe `~` (que l'on obtient en appuyant sur les touches `Alt Gr` et `3` sur un clavier de type PC) puis de la liste des variables explicatives séparées par un signe `+`. Enfin, l'argument `data` permettra d'indiquer notre tableau de données.

```
R> reg <- glm(sport ~ sexe + grpage + etud + relig + heures.tv, data = d, family = binomial(logit))
reg
```

```
Call: glm(formula = sport ~ sexe + grpage + etud + relig + heures.tv,
family = binomial(logit), data = d)
```

Coefficients:

```
(Intercept)           -0.798368
sexeHomme            0.439694
grpage[25,45]         -0.420448
grpage[45,65]          -1.085434
grpage[65,99]          -1.381353
etudSecondaire        0.950571
etudTechnique/Professionnel 1.049253
etudSupérieur          1.891667
etudmanquant           2.150428
religPratiquant occasionnel -0.021904
religAppartenance sans pratique -0.006696
religNi croyance ni appartenance -0.215389
religRejet              -0.383543
religNSP ou NVPR          -0.083789
heures.tv                -0.120911
```

Degrees of Freedom: 1994 Total (i.e. Null); 1980 Residual

(5 observations deleted due to missingness)

Null Deviance: 2609

Residual Deviance: 2206 AIC: 2236

**NOTE**

Il est possible de spécifier des modèles plus complexes. Par exemple, `x:y` permet d'indiquer l'interaction entre les variables `x` et `y`. `x * y` sera équivalent à `x + y + x:y`. Pour aller plus loin, voir <http://ww2.coastal.edu/kingw/statistics/R-tutorials/formulae.html>.

Une présentation plus complète des résultats est obtenue avec la méthode `summary` :

```
R> summary(reg)
```

```
Call:
glm(formula = sport ~ sexe + grpage + etud + relig + heures.tv,
     family = binomial(logit), data = d)

Deviance Residuals:
    Min      1Q   Median      3Q      Max 
-1.8784 -0.8865 -0.4808  1.0033  2.4222 

Coefficients:
                                         Estimate Std. Error
(Intercept)                         -0.798368  0.323903
sexeHomme                            0.439694  0.106062
grpage[25,45]                         -0.420448  0.228053
grpage[45,65]                          -1.085434  0.237716
grpage[65,99]                          -1.381353  0.273796
etudSecondaire                        0.950571  0.197442
etudTechnique/Professionnel          1.049253  0.189804
etudSupérieur                         1.891667  0.195218
etudmanquant                          2.150428  0.330229
religPratiquant occasionnel          -0.021904  0.189199
religAppartenance sans pratique      -0.006696  0.174737
religNi croyance ni appartenance    -0.215389  0.193080
religRejet                             -0.383543  0.285905
religNSP ou NVPR                      -0.083789  0.411028
heures.tv                             -0.120911  0.033591
                                         z value Pr(>|z|) 
(Intercept)                         -2.465  0.013708 *
sexeHomme                            4.146  3.39e-05 ***
grpage[25,45]                         -1.844  0.065236 .
grpage[45,65]                          -4.566  4.97e-06 ***
grpage[65,99]                          -5.045  4.53e-07 ***
etudSecondaire                        4.814  1.48e-06 ***
etudTechnique/Professionnel          5.528  3.24e-08 ***
```

```

etudSupérieur          9.690 < 2e-16 ***
etudmanquant           6.512 7.42e-11 ***
religPratiquant occasionnel -0.116 0.907833
religAppartenance sans pratique -0.038 0.969434
religNi croyance ni appartenance -1.116 0.264617
religRejet              -1.342 0.179756
religNSP ou NVPR         -0.204 0.838470
heures.tv               -3.599 0.000319 ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 2609.2 on 1994 degrees of freedom
Residual deviance: 2206.2 on 1980 degrees of freedom
(5 observations deleted due to missingness)
AIC: 2236.2

Number of Fisher Scoring iterations: 4

```

## Coefficients du modèle

Dans le cadre d'un modèle logistique, généralement on ne présente pas les coefficients du modèle mais leur valeur exponentielle, cette dernière correspondant en effet à des *odds ratio*, également appelés rapports des cotes. L'*odds ratio* diffère du risque relatif. Cependant son interprétation est similaire. Un *odds ratio* de 1 signifie l'absence d'effet. Un *odds ratio* largement supérieur à 1 correspond à une augmentation du phénomène étudié et un *odds ratio* largement inférieur à 1 correspond à une diminution du phénomène étudié.

La fonction `coef` permet d'obtenir les coefficients d'un modèle, `confint` leurs intervalles de confiance et `exp` de calculer l'exponentiel. Les *odds ratio* et leurs intervalles de confiance s'obtiennent ainsi :

```
R> exp(coef(reg))
```

```

(Intercept)      0.4500628
sexeHomme        1.5522329
grpage[25,45]    0.6567525
grpage[45,65]    0.3377552

```

```

grpage[65,99]
  0.2512383
etudSecondaire
  2.5871865
etudTechnique/Professionnel
  2.8555168
etudSupérieur
  6.6304155
etudmanquant
  8.5885303
religPratiquant occasionnel
  0.9783342
religAppartenance sans pratique
  0.9933267
religNi croyance ni appartenance
  0.8062276
religRejet
  0.6814428
religNSP ou NVPR
  0.9196256
heures.tv
  0.8861129

```

```
R> exp(confint(reg))
```

Waiting for profiling to be done...

	2.5 %	97.5 %
(Intercept)	0.2377004	0.8473181
sexeHomme	1.2614265	1.9120047
grpage[25,45)	0.4194391	1.0274553
grpage[45,65)	0.2115307	0.5380894
grpage[65,99]	0.1463860	0.4288023
etudSecondaire	1.7652682	3.8327896
etudTechnique/Professionnel	1.9804881	4.1729378
etudSupérieur	4.5517938	9.7950691
etudmanquant	4.5347799	16.5885323
religPratiquant occasionnel	0.6758807	1.4198530
religAppartenance sans pratique	0.7063000	1.4020242
religNi croyance ni appartenance	0.5524228	1.1782475
religRejet	0.3868800	1.1889781
religNSP ou NVPR	0.3996746	2.0215562
heures.tv	0.8290223	0.9457756

On pourra faciliter la lecture en combinant les deux :

```
R> exp(cbind(coef(reg), confint(reg)))
```

Waiting for profiling to be done...

		2.5 %
(Intercept)	0.4500628	0.2377004
sexeHomme	1.5522329	1.2614265
grpage[25,45]	0.6567525	0.4194391
grpage[45,65]	0.3377552	0.2115307
grpage[65,99]	0.2512383	0.1463860
etudSecondaire	2.5871865	1.7652682
etudTechnique/Professionnel	2.8555168	1.9804881
etudSupérieur	6.6304155	4.5517938
etudmanquant	8.5885303	4.5347799
religPratiquant occasionnel	0.9783342	0.6758807
religAppartenance sans pratique	0.9933267	0.7063000
religNi croyance ni appartenance	0.8062276	0.5524228
religRejet	0.6814428	0.3868800
religNSP ou NVPR	0.9196256	0.3996746
heures.tv	0.8861129	0.8290223
		97.5 %
(Intercept)	0.8473181	
sexeHomme	1.9120047	
grpage[25,45]	1.0274553	
grpage[45,65]	0.5380894	
grpage[65,99]	0.4288023	
etudSecondaire	3.8327896	
etudTechnique/Professionnel	4.1729378	
etudSupérieur	9.7950691	
etudmanquant	16.5885323	
religPratiquant occasionnel	1.4198530	
religAppartenance sans pratique	1.4020242	
religNi croyance ni appartenance	1.1782475	
religRejet	1.1889781	
religNSP ou NVPR	2.0215562	
heures.tv	0.9457756	

Pour savoir si un *odds ratio* diffère significativement de 1 (ce qui est identique au fait que le coefficient soit différent de 0), on pourra se référer à la colonne *Pr(>|z|)* obtenue avec `summary`.

Si vous disposez de l'extension `questionr`, la fonction `odds.ratio` permet de calculer directement les *odds ratio*, leur intervalles de confiance et les *p-value*:

```
R> library(questionr)
odds.ratio(reg)
```

Waiting for profiling to be done...

La fonction `tidy` de l'extension `broom` pour récupérer les coefficients du modèle sous la forme d'un tableau de données. On précisera `conf.int = TRUE` pour obtenir les intervalles de confiance et `exponentiate = TRUE` pour avoir les odds ratio plutôt que les coefficients bruts.

```
R> library(broom)
tidy(reg, conf.int = TRUE, exponentiate = TRUE)
```

L'extension `broom.helpers` fournit une fonction `tidy_plus_plus` qui permet d'améliorer le tableau renvoyé par `tidy` en y identifiant les variables utilisées, ajoutant les modalités de référence et en proposant des étiquettes plus explicites.

```
R> library(broom.helpers)
tidy_plus_plus(reg, exponentiate = TRUE)
```

Si l'on souhaite avoir des noms de variables plus explicites, il faut ajouter des étiquettes des variables avec `var_label` de l'extension `labelled` (voir le chapitre sur les vecteurs labellisés, page 0).

Par contre, cette étape doit avoir eu lieu avant le calcul de la régression linéaire.

```
R> library(labelled)
var_label(d$sport) <- "Pratique du sport ?"
var_label(d$sexe) <- "Sexe"
var_label(d$grpage) <- "Groupe d'âges"
var_label(d$etud) <- "Niveau d'étude"
var_label(d$relig) <- "Pratique religieuse"
var_label(d$heures.tv) <- "Heures de télévision / jour"
var_label(d$trav.imp) <- "Importance du travail"
reg <- glm(sport ~ sexe + grpage + etud + relig + heures.tv, data = d, family = binomial(logit))
```

```
R> tidy_plus_plus(reg, exponentiate = TRUE)
```

La fonction `tbl_regression` de l'extension `gtsummary`, qui a recours en interne à `broom.helpers`, permet d'obtenir un tableau plus propre. Comme nous souhaitons afficher les *odds ratios* plutôt que les coefficients du modèle, on indiquera `exponentiate = TRUE`.

```
R> library(gtsummary)
```

```
Attaching package: 'gtsummary'
```

```
The following objects are masked from 'package:broom.helpers':
```

```
all_categorical, all_continuous, all_dichotomous
```

```
R> tbl_regression(reg, exponentiate = TRUE)
```

Characteristic	OR <sup>1</sup>	95% CI <sup>1</sup>	p-value
<b>Sexe</b>			
Femme	—	—	
Homme	1.55	1.26, 1.91	<0.001
<b>Groupe d'âges</b>			
[16,25)	—	—	
[25,45)	0.66	0.42, 1.03	0.065
[45,65)	0.34	0.21, 0.54	<0.001
[65,99]	0.25	0.15, 0.43	<0.001
<b>Niveau d'étude</b>			
Primaire	—	—	
Secondaire	2.59	1.77, 3.83	<0.001
Technique/Professionnel	2.86	1.98, 4.17	<0.001
Supérieur	6.63	4.55, 9.80	<0.001
manquant	8.59	4.53, 16.6	<0.001
<b>Pratique religieuse</b>			
Pratiquant regulier	—	—	
Pratiquant occasionnel	0.98	0.68, 1.42	>0.9
Appartenance sans pratique	0.99	0.71, 1.40	>0.9
Ni croyance ni appartenance	0.81	0.55, 1.18	0.3
Rejet	0.68	0.39, 1.19	0.2
NSP ou NVPR	0.92	0.40, 2.02	0.8
Heures de télévision / jour	0.89	0.83, 0.95	<0.001

<sup>1</sup> OR = Odds Ratio, CI = Confidence Interval

On pourra se référer à la [vignette dédiée \(en anglais\)](#) pour découvrir les différentes options de `tbl_regression`.

## Représentation graphique du modèle

Il est possible de représenter graphiquement les différents odds ratios avec la fonction `ggcoef_model` de [GGally](#).

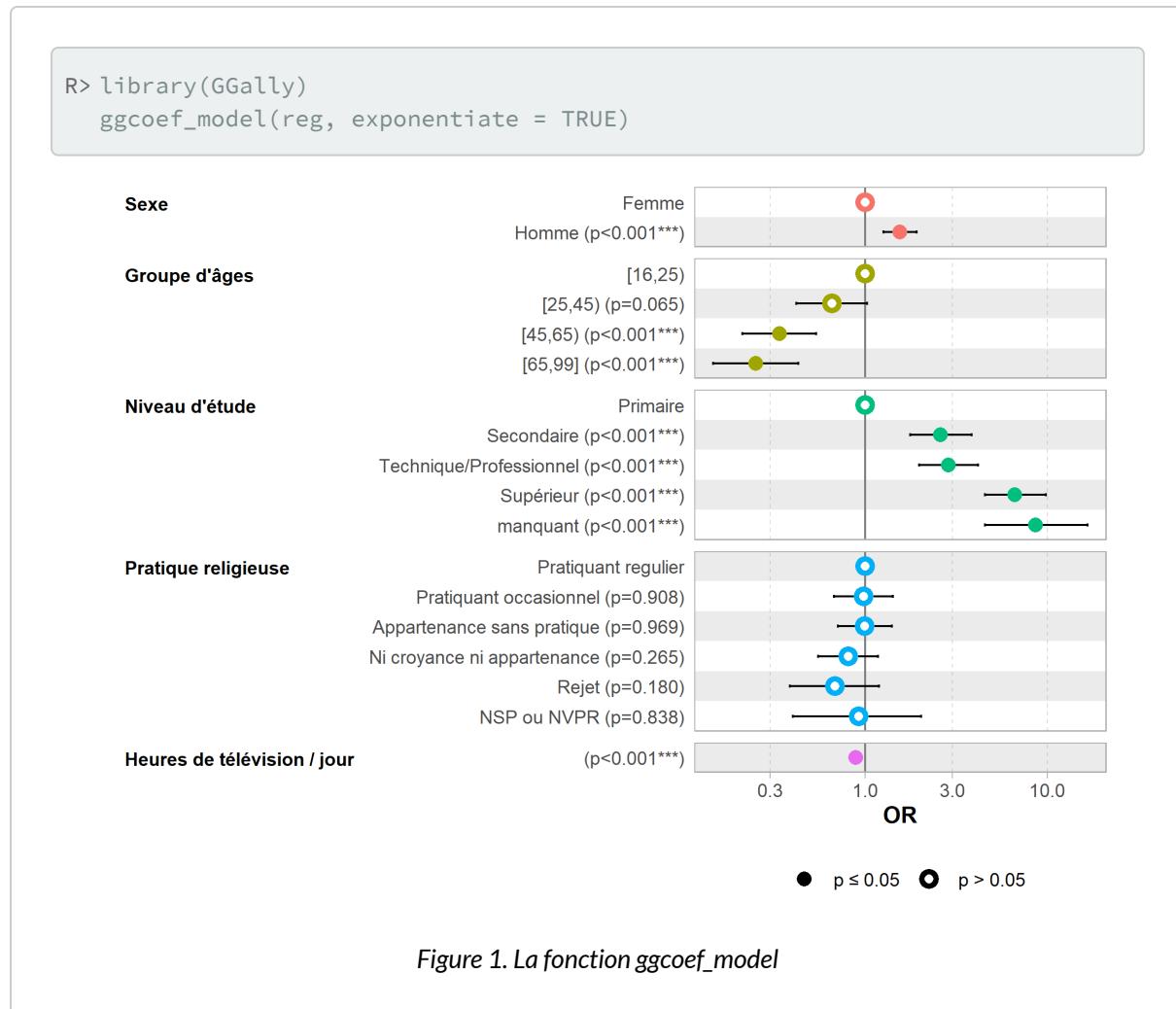


Figure 1. La fonction `ggcoef_model`

div class="note">> L'extension `forestmodel` propose de son côté une fonction `forest_model` qui, à partir d'un modèle, propose une représentation visuelle et tabulaire des coefficients.

```
R> library(forestmodel)
forest_model(reg)
```

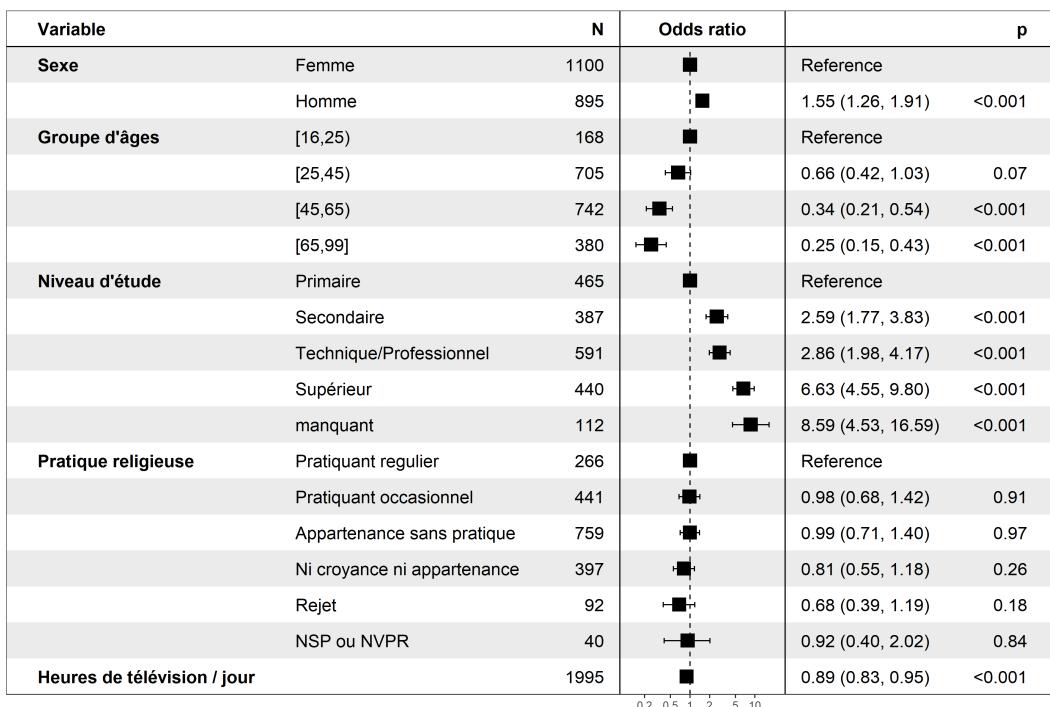


Figure 2. La fonction `forest_model`

A la différence de `ggcoef_model` qui fonctionne avec la plupart des types de modèles (dont les modèles à effets mixtes ou ceux réalisés avec un plan d'échantillonnage complexe), `forest_model` ne fonctionne qu'avec les modèles les plus courants.

## Représentation graphique des effets

L'extension `effects` propose une représentation graphique résumant les effets de chaque variable du modèle. Pour cela, il suffit d'appliquer la méthode `plot` au résultat de la fonction `allEffects`. Nous obtenons alors la figure ci-dessous, page 522.

```
R> library(effects)
```

Loading required package: carData

lattice theme set by effectsTheme()  
See ?effectsTheme for details.

```
R> plot(allEffects(reg))
```

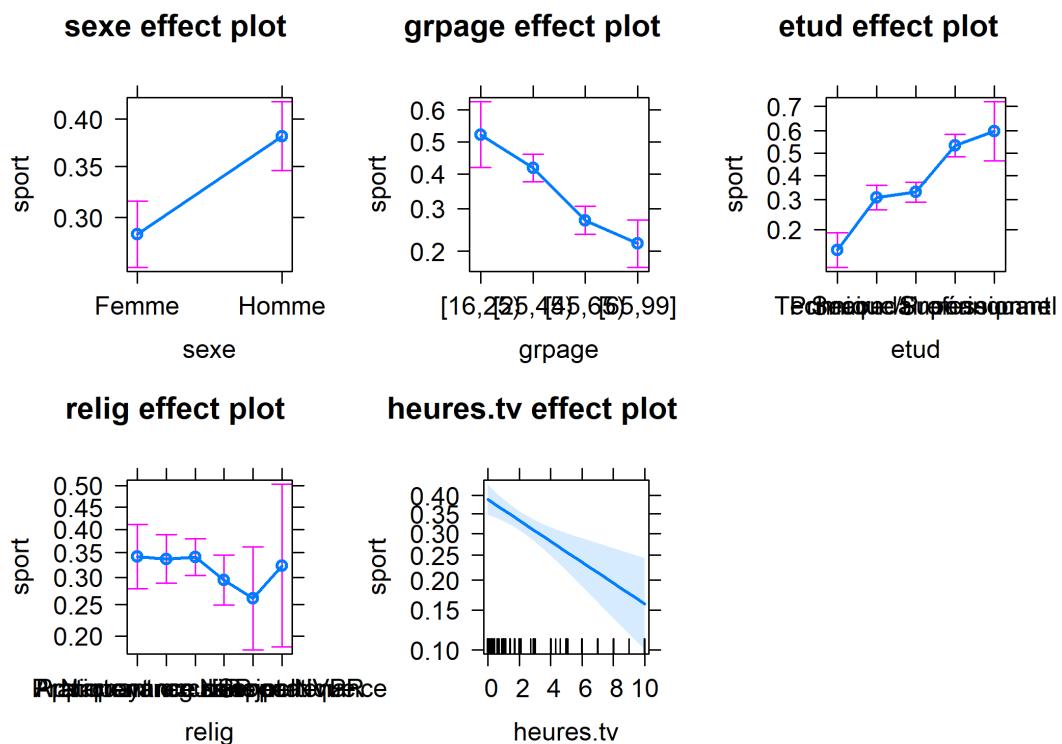


Figure 3. Représentation graphique de l'effet de chaque variable du modèle logistique

Nous pouvons alternativement avoir recours à l'extension `ggeffects` et sa fonction `ggeffect` qui permettent de récupérer les résultats de `effects` dans un format utilisable avec `ggplot2`.

Ainsi, la fonction `ggeffect`, quand on lui précise un terme spécifique, produit un tableau de données avec les effets marginaux pour cette variable.

```
R> library(ggeffects)
ggeffect(reg, "sexe")
```

En combinant ce résultat avec `plot`, on obtient un graphique `ggplot2` de l'effet en question.

```
R> plot(ggeffect(reg, "sexe"))
```

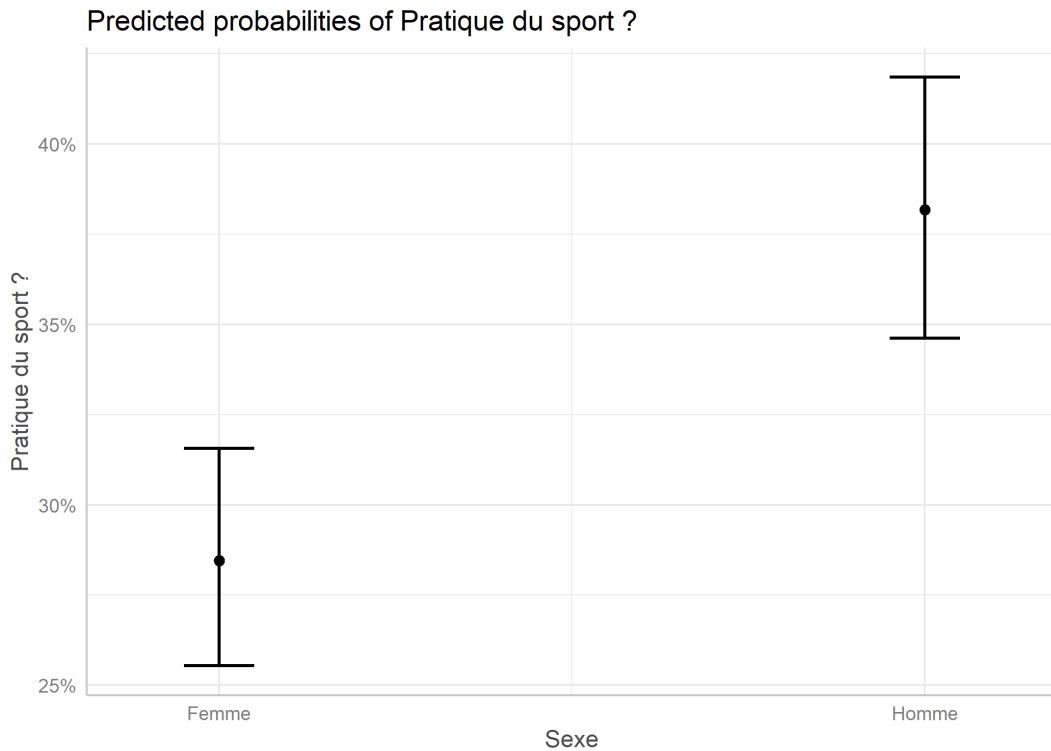


Figure 4. Effet du sexe représenté avec `ggeffect`

Si l'on ne précise pas de terme, `ggeffect(reg)` calcule les effets pour chaque variable du modèle et `plot(ggeffect(reg))` renvoie une liste de graphiques. Il faut donc utiliser la fonction `plot_grid` de `cowplot` pour combiner ces graphiques en un seul (voir le chapitre dédié, page 825).

```
R> library(ggeffects)
cowplot:::plot_grid(plotlist = plot(ggeffect(reg)))
```

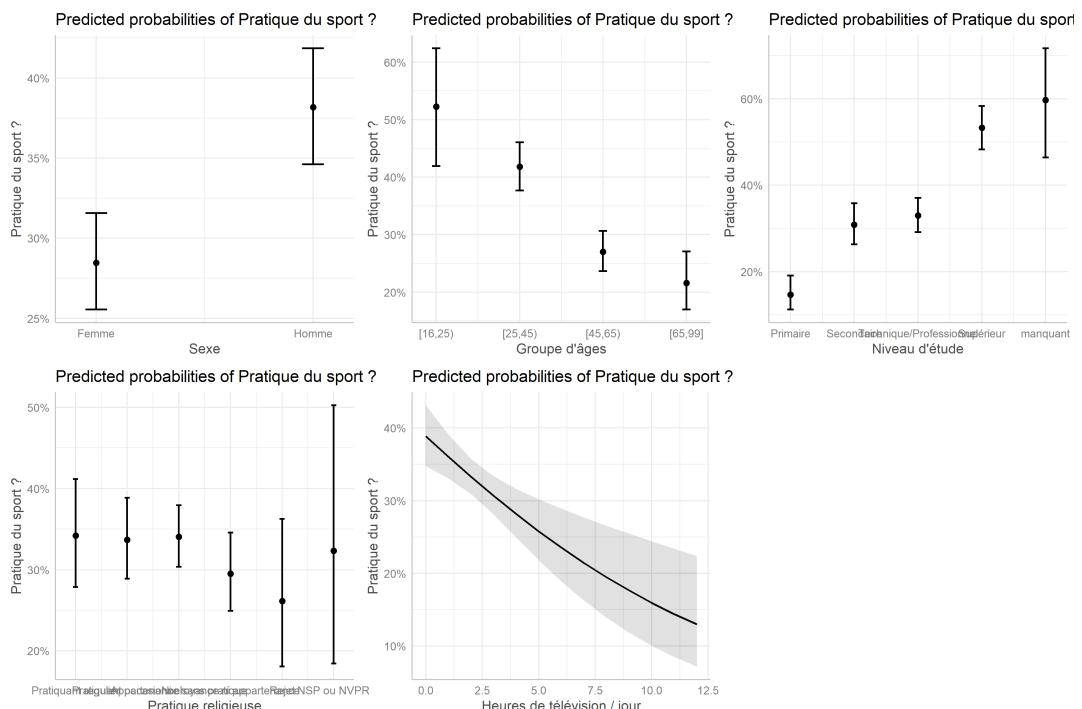


Figure 5. Effet du modèles représentés avec ggeffect

On pourra noter la prise en compte par **ggeffect** des étiquettes de variables définies plus haut.

## Matrice de confusion

Une manière de tester la qualité d'un modèle est le calcul d'une matrice de confusion, c'est-à-dire le tableau croisé des valeurs observées et celles des valeurs prédites en appliquant le modèle aux données d'origine.

La méthode **predict** avec l'argument **type="response"** permet d'appliquer notre modèle logistique à un tableau de données et renvoie pour chaque individu la probabilité qu'il ait vécu le phénomène étudié.

```
R> sport.pred <- predict(reg, type = "response", newdata = d)
head(sport.pred)
```

1	2	3	4	5
---	---	---	---	---

```
0.61241192 0.73414575 0.15982958 0.70350157 0.07293505
6
0.34824228
```

Or notre variable étudiée est de type binaire. Nous devons donc transformer nos probabilités prédictives en une variable du type « oui / non ». Usuellement, les probabilités prédictives seront réunies en deux groupes selon qu'elles soient supérieures ou inférieures à la moitié. La matrice de confusion est alors égale à :

```
R> table(sport.pred > 0.5, d$sport)
```

	Non	Oui
FALSE	1076	384
TRUE	199	336

Nous avons donc 583 (384+199) prédictions incorrectes sur un total de 1993, soit un taux de mauvais classement de 29,3 %.

## Identifier les variables ayant un effet significatif

Les p-values associées aux odds ratios nous indiquent si un odd ratio est significativement différent de 1, par rapport à la modalité de référence. Mais cela n'indique pas si globalement une variable a un effet significatif sur le modèle. Pour tester l'effet global sur un modèle, on peut avoir recours à la fonction `drop1`. Cette dernière va tour à tour supprimer chaque variable du modèle et réaliser une analyse de variance (ANOVA, voir fonction `anova`) pour voir si la variance change significativement.

```
R> drop1(reg, test = "Chisq")
```

Ainsi, dans le cas présent, la suppression de la variable `relig` ne modifie significativement pas le modèle, indiquant l'absence d'effet de cette variable.

Une fonction plus générique (i.e. fonctionnant avec une plus grande variété de modèles) est la fonction `Anova` de l'extension `car`.

```
R> library(car)
Anova(reg)
```

Si l'on a recours à `tbl_regression`, on peut facilement ajouter les p-valeurs globales avec `add_global_p`.

```
R> reg %>%
 tbl_regression(exponentiate = TRUE) %>%
add_global_p(type = "II")
```

```
add_global_p: Global p-values for variable(s) `add_global_p(include = c("sexe",
"grpage", "etud", "relig", "heures.tv"))` were calculated with
`car::Anova(x$model_obj, type = "II")`
```

<b>Characteristic</b>	<b>OR<sup>1</sup></b>	<b>95% CI<sup>1</sup></b>	<b>p-value</b>
Sexe			<0.001
Femme	—	—	
Homme	1.55	1.26, 1.91	
Groupe d'âges			<0.001
[16,25)	—	—	
[25,45)	0.66	0.42, 1.03	
[45,65)	0.34	0.21, 0.54	
[65,99]	0.25	0.15, 0.43	
Niveau d'étude			<0.001
Primaire	—	—	
Secondaire	2.59	1.77, 3.83	
Technique/Professionnel	2.86	1.98, 4.17	
Supérieur	6.63	4.55, 9.80	
manquant	8.59	4.53, 16.6	
Pratique religieuse			0.5
Pratiquant regulier	—	—	
Pratiquant occasionnel	0.98	0.68, 1.42	
Appartenance sans pratique	0.99	0.71, 1.40	
Ni croyance ni appartenance	0.81	0.55, 1.18	
Rejet	0.68	0.39, 1.19	
NSP ou NVPR	0.92	0.40, 2.02	
Heures de télévision / jour	0.89	0.83, 0.95	<0.001

<sup>1</sup> OR = Odds Ratio, CI = Confidence Interval

#### NOTE

Concernant le test réalisé dans le cadre d'une Anova, il existe trois tests différents que l'on présente comme le type 1, le type 2 et le type 3 (ou I, II et III). Pour une explication sur ces différents types, on pourra se référer (en anglais) à <https://mcfromnz.wordpress.com/2011/03/02/anova-type-iiiiii-ss-explained/> ou encore [http://md.psych.bio.uni-goettingen.de/mv/unit/lm\\_cat/lm\\_cat\\_unbal\\_ss\\_explained.html](http://md.psych.bio.uni-goettingen.de/mv/unit/lm_cat/lm_cat_unbal_ss_explained.html).

Le type I n'est pas recommandé dans le cas présent car il dépend de l'ordre dans lequel les différentes variables sont testées.

Lorsqu'il n'y a pas d'interaction dans un modèle (voir le chapitre sur les interactions, page 617), le type II serait à privilégier car plus puissant.

En présence d'interactions, il est conseillé d'avoir plutôt recours au type III. Cependant, en toute rigueur, pour utiliser le type III, il faut que les variables catégorielles soient codées en utilisant un contrastes dont la somme est nulle (un contrast de type somme ou polynomial). Or, par défaut, les variables catégorielles sont codées avec un contraste de type «traitement» (pour une explication sur les contrastes dans les modèles, voir en anglais [cette page](#) ou [celle-ci](#)).

Par défaut, `Anova` utilise le type II et `add_global_p` le type III. Dans les deux cas, il est possible de préciser le type de test avec `type = "II"` ou `type = "III"`.

Dans le cas de notre exemple, un modèle simple sans interaction, le type de test ne change pas les résultats.

```
R> Anova(reg, type = "II")
```

```
R> Anova(reg, type = "III")
```

## Sélection de modèles

Il est toujours tentant lorsque l'on recherche les facteurs associés à un phénomène d'inclure un nombre important de variables explicatives potentielles dans un modèle logistique. Cependant, un tel modèle n'est pas forcément le plus efficace et certaines variables n'auront probablement pas d'effet significatif sur la variable d'intérêt.

La technique de sélection descendante pas à pas est une approche visant à améliorer son modèle explicatif. On réalise un premier modèle avec toutes les variables spécifiées, puis on regarde s'il est possible d'améliorer le modèle en supprimant une des variables du modèle. Si plusieurs variables permettent d'améliorer le modèle, on supprimera la variable dont la suppression améliorera le plus le

modèle. Puis on recommence le même procédé pour voir si la suppression d'une seconde variable peut encore améliorer le modèle et ainsi de suite. Lorsque le modèle ne peut plus être améliorer par la suppression d'une variable, on s'arrête.

Il faut également définir un critère pour déterminer la qualité d'un modèle. L'un des plus utilisés est le Akaike Information Criterion ou AIC. Plus l'AIC sera faible, meilleure sera le modèle.

La fonction `step` permet justement de sélectionner le meilleur modèle par une procédure pas à pas descendante basée sur la minimisation de l'AIC. La fonction affiche à l'écran les différentes étapes de la sélection et renvoie le modèle final.

```
R> reg2 <- step(reg)
```

```
Start: AIC=2236.17
sport ~ sexe + grpage + etud + relig + heures.tv
```

	Df	Deviance	AIC
- relig	5	2210.4	2230.4
<none>		2206.2	2236.2
- heures.tv	1	2219.6	2247.6
- sexe	1	2223.5	2251.5
- grpage	3	2259.0	2283.0
- etud	4	2330.0	2352.0

```
Step: AIC=2230.4
sport ~ sexe + grpage + etud + heures.tv
```

	Df	Deviance	AIC
<none>		2210.4	2230.4
- heures.tv	1	2224.0	2242.0
- sexe	1	2226.4	2244.4
- grpage	3	2260.6	2274.6
- etud	4	2334.3	2346.3

Le modèle initial a un AIC de 2235,9. À la première étape, il apparait que la suppression de la variable religion permet diminuer l'AIC à 2230,2. Lors de la seconde étape, toute suppression d'une autre variable ferait augmenter l'AIC. La procédure s'arrête donc.

Pour obtenir directement l'AIC d'un modèle donné, on peut utiliser la fonction `AIC`.

```
R> AIC(reg)
```

```
[1] 2236.173
```

```
R> AIC(reg2)
```

```
[1] 2230.404
```

On peut effectuer une analyse de variance ou ANOVA pour comparer les deux modèles avec la fonction anova.

```
R> anova(reg, reg2, test = "Chisq")
```

Il n'y a pas de différences significatives entre nos deux modèles. Autrement dit, notre second modèle explique tout autant de variance que notre premier modèle, tout en étant plus parcimonieux.

**NOTE**

Une alternative à la fonction `step` est la fonction `stepAIC` de l'extension **MASS** qui fonctionne de la même manière. Si cela ne change rien aux régressions logistiques classiques, il arrive que pour certains types de modèle la méthode `step` ne soit pas disponible, mais que `stepAIC` puisse être utilisée à la place.

```
R> library(MASS)
reg2bis <- stepAIC(reg)
```

```
Start: AIC=2236.17
sport ~ sexe + grpage + etud + relig + heures.tv
```

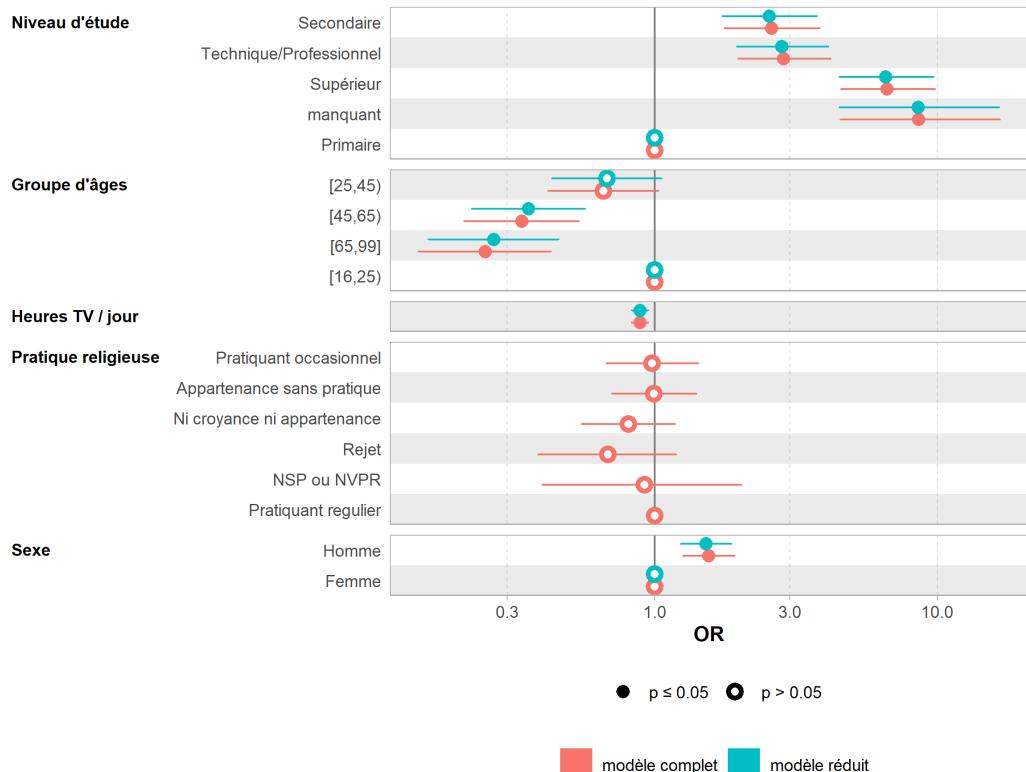
	Df	Deviance	AIC
- relig	5	2210.4	2230.4
<none>		2206.2	2236.2
- heures.tv	1	2219.6	2247.6
- sexe	1	2223.5	2251.5
- grpage	3	2259.0	2283.0
- etud	4	2330.0	2352.0

```
Step: AIC=2230.4
sport ~ sexe + grpage + etud + heures.tv
```

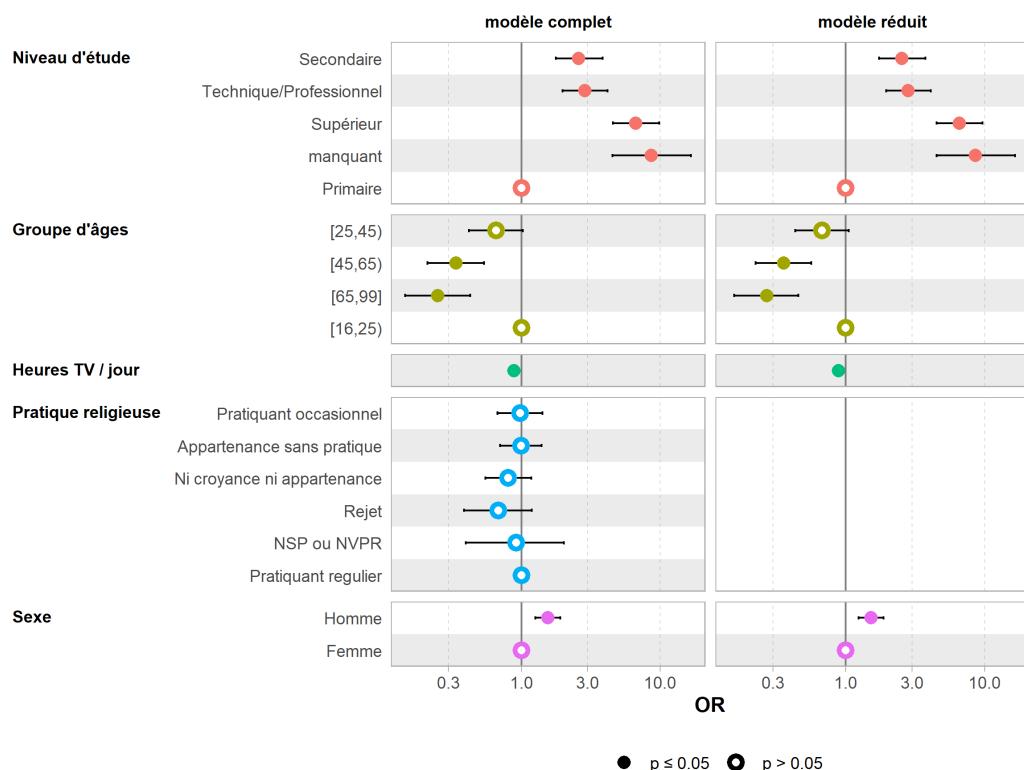
	Df	Deviance	AIC
<none>		2210.4	2230.4
- heures.tv	1	2224.0	2242.0
- sexe	1	2226.4	2244.4
- grpage	3	2260.6	2274.6
- etud	4	2334.3	2346.3

On peut facilement comparer visuellement deux modèles avec `ggcoef_compare` de **GGally**.

```
R> library(GGally)
ggcoef_compare(
  list("modèle complet" = reg, "modèle réduit" = reg2),
  exponentiate = TRUE,
  variable_labels = c(heures.tv = "Heures TV / jour")
)
```



```
R> ggcoef_compare(
  list("modèle complet" = reg, "modèle réduit" = reg2),
  type = "faceted",
  exponentiate = TRUE,
  variable_labels = c(heures.tv = "Heures TV / jour")
)
```

Figure 6. La fonction `ggcoef_compare`

## Tableaux «all-in-one»

### gtsummary

Nous avons déjà vu précédemment la fonction `tbl_regression` de l'extension `gtsummary`. Une vignette dédiée de l'extension explicite les possibilités de personnalisation des résultats. Par exemple `add_global_p` permet d'ajouter des p-valeurs globales pour chaque variable.

```
R> library(gtsummary)
  theme_gtsummary_language("fr", decimal.mark = ",",
                           big.mark = " ")
```

```
Setting `language: fr` theme
```

```
R> tbl_regression(reg, exponentiate = TRUE) %>%
  add_global_p(keep = TRUE)
```

```
add_global_p: Global p-values for variable(s) `add_global_p(include = c("sexe",
"grpage", "etud", "relig", "heures.tv"))` were calculated with
`car:::Anova(x$model_obj, type = "III")`
```

Caractéristique	OR <sup>1</sup>	95% CI <sup>1</sup>	p-value
Sexe	<0,001		
Femme	—	—	
Homme	1,55	1,26 – 1,91	<0,001
Groupe d'âges	<0,001		
[16,25)	—	—	
[25,45)	0,66	0,42 – 1,03	0,065
[45,65)	0,34	0,21 – 0,54	<0,001
[65,99]	0,25	0,15 – 0,43	<0,001
Niveau d'étude	<0,001		
Primaire	—	—	
Secondaire	2,59	1,77 – 3,83	<0,001
Technique/Professionnel	2,86	1,98 – 4,17	<0,001
Supérieur	6,63	4,55 – 9,80	<0,001
manquant	8,59	4,53 – 16,6	<0,001
Pratique religieuse	0,5		
Pratiquant régulier	—	—	
Pratiquant occasionnel	0,98	0,68 – 1,42	>0,9
Appartenance sans pratique	0,99	0,71 – 1,40	>0,9
Ni croyance ni appartenance	0,81	0,55 – 1,18	0,3
Rejet	0,68	0,39 – 1,19	0,2
NSP ou NVPR	0,92	0,40 – 2,02	0,8
Heures de télévision / jour	0,89	0,83 – 0,95	<0,001

<sup>1</sup> OR = rapport de cotes, CI = intervalle de confiance

On peut remarquer que `gtsummary` (comme d'autres extensions présentées précédemment) a tenu compte des étiquettes de variables définies plus haut avec `var_label` de l'extension `labelled` (voir le

chapitre sur les vecteurs labellisés, page 0).

Comme `tbl_regression` repose sur les extensions `broom` et `broom.helpers`, `gtsummary` peut en principe couvrir un plus grand nombre de modèles que `finalfit` (voir ci-dessous) ainsi qu'un plus grand nombre de cas particuliers (effets d'interactions, contrastes autres que traitement, paramètre polynomiaux définis avec `poly`).

`tbl_merge` permet de fusionner ensemble les résultats de plusieurs modèles.

```
R> tbl_merge(  
  tbls = list(tbl_regression(reg), tbl_regression(reg2)),  
  tab_spanner = c("Modèle complet", "Modèle réduit")  
)
```

Caractéristique	Modèle complet			Modèle réduit		
	log(OR) <sup>1</sup>	95% CI <sup>1</sup>	p-value	log(OR) <sup>1</sup>	95% CI <sup>1</sup>	p-value
<b>Sexe</b>						
Femme	—	—	—	—	—	—
Homme	0,44	0,23 – 0,65	<0,001	0,42	0,21 – 0,62	<0,001
<b>Groupe d'âges</b>						
[16,25)	—	—	—	—	—	—
[25,45)	-0,42	-0,87 – 0,03	0,065	-0,39	-0,84 – 0,05	0,084
[45,65)	-1,1	-1,6 – -0,62	<0,001	-1,0	-1,5 – -0,57	<0,001
[65,99]	-1,4	-1,9 – -0,85	<0,001	-1,3	-1,8 – -0,78	<0,001
<b>Niveau d'étude</b>						
Primaire	—	—	—	—	—	—
Secondaire	1,0	0,57 – 1,3	<0,001	0,93	0,55 – 1,3	<0,001
Technique/ Professionnel	1,0	0,68 – 1,4	<0,001	1,0	0,67 – 1,4	<0,001
Supérieur	1,9	1,5 – 2,3	<0,001	1,9	1,5 – 2,3	<0,001
manquant	2,2	1,5 – 2,8	<0,001	2,1	1,5 – 2,8	<0,001
<b>Pratique religieuse</b>						
Pratiquant régulier	—	—	—	—	—	—
Pratiquant occasionnel	-0,02	-0,39 – 0,35	>0,9	—	—	—
Appartenance sans	-0,01	-0,35 –	>0,9	—	—	—

<sup>1</sup> OR = rapport de cotes, CI = intervalle de confiance

Caractéristique	Modèle complet			Modèle réduit		
	log(OR) <sup>1</sup>	95% CI <sup>1</sup>	p-value	log(OR) <sup>1</sup>	95% CI <sup>1</sup>	p-value
pratique	0,34					
Ni croyance ni appartenance	-0,22	-0,59 – 0,16	0,3			
Rejet	-0,38	-0,95 – 0,17	0,2			
NSP ou NVPR	-0,08	-0,92 – 0,70	0,8			
Heures de télévision / jour	-0,12	-0,19 – -0,06	<0,001	-0,12	-0,19 – -0,06	<0,001

<sup>1</sup> OR = rapport de cotes, CI = intervalle de confiance

L'extension `gtsummary` fournit également la fonction `tbl_summary` pour effectuer des tris à plats et/ou un tri croisé.

```
R> d %>%
  dplyr::select(sport, sexe, grpage, etud, relig, heures.tv) %>%
 tbl_summary()
```

<b>Caractéristique</b>	<b>N = 2 000<sup>1</sup></b>
<b>Pratique du sport ?</b>	
Non	1 277 (64%)
Oui	723 (36%)
<b>Sexe</b>	
Femme	1 101 (55%)
Homme	899 (45%)
<b>Groupe d'âges</b>	
[16,25)	169 (8,5%)
[25,45)	706 (35%)
[45,65)	745 (37%)
[65,99]	380 (19%)
<b>Niveau d'étude</b>	
Primaire	466 (23%)
Secondaire	387 (19%)
Technique/Professionnel	594 (30%)
Supérieur	441 (22%)
manquant	112 (5,6%)
<b>Pratique religieuse</b>	
Pratiquant régulier	266 (13%)
Pratiquant occasionnel	442 (22%)
Appartenance sans pratique	760 (38%)
Ni croyance ni appartenance	399 (20%)
Rejet	93 (4,7%)

<sup>1</sup> n (%); Médiane (EI)

<b>Caractéristique</b>	<b>N = 2 000<sup>1</sup></b>
NSP ou NVPR	40 (2,0%)
Heures de télévision / jour	2,00 (1,00 – 3,00)
Manquant	5

<sup>1</sup> n (%) ; Médiane (EI)

```
R> d %>%
  dplyr::select(sport, sexe, grpage, etud, relig, heures.tv) %>%
 tbl_summary(by = "sport", percent = "row") %>%
add_overall(last = TRUE) %>%
add_p()
```

Caractéristique	Non, N = 1 277 <sup>1</sup>	Oui, N = 723 <sup>1</sup>	Total, N = 2 000 <sup>1</sup>	p- value <sup>2</sup>
Sexe				<0,001
Femme	747 (68%)	354 (32%)	1 101 (100%)	
Homme	530 (59%)	369 (41%)	899 (100%)	
Groupe d'âges				<0,001
[16,25)	58 (34%)	111 (66%)	169 (100%)	
[25,45)	359 (51%)	347 (49%)	706 (100%)	
[45,65)	541 (73%)	204 (27%)	745 (100%)	
[65,99]	319 (84%)	61 (16%)	380 (100%)	
Niveau d'étude				<0,001
Primaire	416 (89%)	50 (11%)	466 (100%)	
Secondaire	270 (70%)	117 (30%)	387 (100%)	
Technique/Professionnel	378 (64%)	216 (36%)	594 (100%)	
Supérieur	186 (42%)	255 (58%)	441 (100%)	
manquant	27 (24%)	85 (76%)	112 (100%)	
Pratique religieuse				0,14
Pratiquant régulier	182 (68%)	84 (32%)	266 (100%)	
Pratiquant occasionnel	295 (67%)	147 (33%)	442 (100%)	
Appartenance sans pratique	473 (62%)	287 (38%)	760 (100%)	
Ni croyance ni appartenance	239 (60%)	160 (40%)	399 (100%)	
Rejet	60 (65%)	33 (35%)	93 (100%)	
NSP ou NVPR	28 (70%)	12 (30%)	40 (100%)	

<sup>1</sup> n (%); Médiane (EI)<sup>2</sup> Pearson's Chi-squared test; Wilcoxon rank sum test

Caractéristique	Non, N = 1 277 <sup>1</sup>	Oui, N = 723 <sup>1</sup>	Total, N = 2 000 <sup>1</sup>	p- value <sup>2</sup>
Heures de télévision / jour	2,00 (1,00 – 3,00)	2,00 (1,00 – 3,00)	2,00 (1,00 – 3,00)	<0,001
Manquant	2	3	5	

<sup>1</sup> n (%); Médiane (EI)

<sup>2</sup> Pearson's Chi-squared test; Wilcoxon rank sum test

Il est à noter que `tbl_regression` sait prendre en compte les effets d'interactions (voir ci-après).

## finalfit

L'extension `finalfit` fournit une fonction `finalfit` du type "all-in-one" qui calcule un tableau avec les tris croisés, les odds ratios univariés et un modèle multivarié.

Il faut d'abord définir la variable dépendante et les variables explicatives.

```
R> dep <- "sport"
  vars <- c("sexe", "grpage", "etud", "relig", "heures.tv")
```

Une première fonction `summary_factorlist` fournit un tableau descriptif avec, si l'option `p = TRUE` est indiquée, des tests de comparaisons (ici des tests du Chi<sup>2</sup>).

```
R> library(finalfit)
  tab <- summary_factorlist(d, dep, vars, p = TRUE, add_dependent_label = TRUE)
  tab
```

On peut remarquer que `finalfit` a tenu compte des étiquettes de variables définies plus haut avec `var_label` de l'extension `labelled` (voir le chapitre sur les vecteurs labellisés, page 0).

On peut associer le résultat avec la fonction `kable` de `knitr` pour un rendu plus esthétique lorsque l'on produit un rapport `Rmarkdown` (voir le chapitre dédié aux rapports automatisés, page 979).

```
R> knitr::kable(tab, row.names = FALSE)
```

Dependent: Pratique du sport ?		Non	Oui	p
Sexe	Femme	747 (58.5)	354 (49.0)	<0.001
	Homme	530 (41.5)	369 (51.0)	
Groupe d'âges	[16,25)	58 (4.5)	111 (15.4)	<0.001
	[25,45)	359 (28.1)	347 (48.0)	
	[45,65)	541 (42.4)	204 (28.2)	
	[65,99]	319 (25.0)	61 (8.4)	
Niveau d'étude	Primaire	416 (32.6)	50 (6.9)	<0.001
	Secondaire	270 (21.1)	117 (16.2)	
	Technique/Professionnel	378 (29.6)	216 (29.9)	
	Supérieur	186 (14.6)	255 (35.3)	
	manquant	27 (2.1)	85 (11.8)	
Pratique religieuse	Pratiquant regulier	182 (14.3)	84 (11.6)	0.144
	Pratiquant occasionnel	295 (23.1)	147 (20.3)	
	Appartenance sans pratique	473 (37.0)	287 (39.7)	
	Ni croyance ni appartenance	239 (18.7)	160 (22.1)	

---

Dependent: Pratique du sport ?		Non	Oui	p
	Rejet	60 (4.7)	33 (4.6)	
	NSP ou NVPR	28 (2.2)	12 (1.7)	
Heures de télévision / jour	Mean (SD)	2.5 (1.9)	1.8 (1.4)	<0.001

La fonction `finalfit`, quant à elle, calcule à la fois les *odds ratios* univariés (modèles logistiques avec une seule variable inclue à la fois) et un modèle complet, présentant le tout dans un tableau synthétique.

```
R> tab <- finalfit(d, dep, vars)
  knitr::kable(tab, row.names = FALSE)
```

<b>Dependent:</b> <b>Pratique du sport</b>		<b>Non</b>	<b>Oui</b>	<b>OR (univariable)</b>	<b>OR (multivariable)</b>
<b>?</b>					
Sexe	Femme	747 (67.8)	354 (32.2)	-	-
	Homme	530 (59.0)	369 (41.0)	1.47 (1.22-1.77, p<0.001)	1.55 (1.26-1.91, p<0.001)
Groupe d'âges	[16,25)	58 (34.3)	111 (65.7)	-	-
	[25,45)	359 (50.8)	347 (49.2)	0.51 (0.35-0.71, p<0.001)	0.66 (0.42-1.03, p=0.065)
	[45,65)	541 (72.6)	204 (27.4)	0.20 (0.14-0.28, p<0.001)	0.34 (0.21-0.54, p<0.001)
[65,99]		319 (83.9)	61 (16.1)	-	-
Niveau d'étude	Primaire	416 (89.3)	50 (10.7)	-	-
	Secondaire	270 (69.8)	117 (30.2)	3.61 (2.52-5.23, p<0.001)	2.59 (1.77-3.83, p<0.001)
	Technique/ Professionnel	378 (63.6)	216 (36.4)	4.75 (3.42-6.72, p<0.001)	2.86 (1.98-4.17, p<0.001)
Supérieur		186 (42.2)	255 (57.8)	11.41 (8.11-16.31, p<0.001)	6.63 (4.55-9.80, p<0.001)
manquant		27 (24.1)	85 (75.9)	26.19 (15.74-44.90, p<0.001)	8.59 (4.53-16.59, p<0.001)
Pratique religieuse	Pratiquant regulier	182 (68.4)	84 (31.6)	-	-

<b>Dependent:</b> <b>Pratique du sport</b>		<b>Non</b>	<b>Oui</b>	<b>OR (univariable)</b>	<b>OR (multivariable)</b>
?					
Pratiquant occasionnel	295 (66.7)	147 (33.3)		1.08 (0.78-1.50, p=0.644)	0.98 (0.68-1.42, p=0.908)
Appartenance sans pratique	473 (62.2)	287 (37.8)		1.31 (0.98-1.78, p=0.071)	0.99 (0.71-1.40, p=0.969)
Ni croyance ni appartenance	239 (59.9)	160 (40.1)		1.45 (1.05-2.02, p=0.026)	0.81 (0.55-1.18, p=0.265)
Rejet	60 (64.5)	33 (35.5)		1.19 (0.72-1.95, p=0.489)	0.68 (0.39-1.19, p=0.180)
NSP ou NVPR	28 (70.0)	12 (30.0)		0.93 (0.44-1.88, p=0.841)	0.92 (0.40-2.02, p=0.838)
Heures de télévision / jour	Mean (SD)	2.5 (1.9)	1.8 (1.4)	0.79 (0.74-0.84, p<0.001)	0.89 (0.83-0.95, p<0.001)
NA	NA	NA	NA	0.10 (0.07-0.15, p<0.001)	0.25 (0.15-0.43, p<0.001)

Par défaut, toutes les variables explicatives fournies sont retenues dans le modèle affiché. Si on ne souhaite inclure que certaines variables dans le modèle multivarié (parce que l'on aura précédemment réalisé une procédure [step](#)), il faudra préciser séparément les variables du modèle multivarié.

```
R> vars_multi <- c("sexe", "grpage", "etud", "heures.tv")
  tab <- finalfit(d, dep, vars, explanatory_multi = vars_multi)
  knitr::kable(tab, row.names = FALSE)
```

<b>Dependent:</b> <b>Pratique du sport</b>		<b>Non</b>	<b>Oui</b>	<b>OR (univariable)</b>	<b>OR (multivariable)</b>
<b>?</b>					
Sexe	Femme	747 (67.8)	354 (32.2)	-	-
	Homme	530 (59.0)	369 (41.0)	1.47 (1.22-1.77, p<0.001)	1.52 (1.24-1.87, p<0.001)
Groupe d'âges	[16,25)	58 (34.3)	111 (65.7)	-	-
	[25,45)	359 (50.8)	347 (49.2)	0.51 (0.35-0.71, p<0.001)	0.68 (0.43-1.06, p=0.084)
	[45,65)	541 (72.6)	204 (27.4)	0.20 (0.14-0.28, p<0.001)	0.36 (0.23-0.57, p<0.001)
[65,99]		319 (83.9)	61 (16.1)	-	-
Niveau d'étude	Primaire	416 (89.3)	50 (10.7)	-	-
	Secondaire	270 (69.8)	117 (30.2)	3.61 (2.52-5.23, p<0.001)	2.54 (1.73-3.75, p<0.001)
	Technique/ Professionnel	378 (63.6)	216 (36.4)	4.75 (3.42-6.72, p<0.001)	2.81 (1.95-4.10, p<0.001)
Supérieur		186 (42.2)	255 (57.8)	11.41 (8.11-16.31, p<0.001)	6.55 (4.50-9.66, p<0.001)
manquant		27 (24.1)	85 (75.9)	26.19 (15.74-44.90, p<0.001)	8.54 (4.51-16.47, p<0.001)
Pratique religieuse	Pratiquant regulier	182 (68.4)	84 (31.6)	-	-

<b>Dependent:</b> <b>Pratique du sport</b>		<b>Non</b>	<b>Oui</b>	<b>OR (univariable)</b>	<b>OR (multivariable)</b>
?					
Pratiquant occasionnel	295 (66.7)	147 (33.3)		1.08 (0.78-1.50, p=0.644)	-
Appartenance sans pratique	473 (62.2)	287 (37.8)		1.31 (0.98-1.78, p=0.071)	-
Ni croyance ni appartenance	239 (59.9)	160 (40.1)		1.45 (1.05-2.02, p=0.026)	-
Rejet	60 (64.5)	33 (35.5)		1.19 (0.72-1.95, p=0.489)	-
NSP ou NVPR	28 (70.0)	12 (30.0)		0.93 (0.44-1.88, p=0.841)	-
Heures de télévision /jour	Mean (SD)	2.5 (1.9)	1.8 (1.4)	0.79 (0.74-0.84, p<0.001)	0.89 (0.83-0.95, p<0.001)
NA	NA	NA	NA	0.10 (0.07-0.15, p<0.001)	0.27 (0.16-0.46, p<0.001)

On pourra se référer à l'aide de la fonction `finalfit` pour d'autres exemples.

L'extension `finalfit` propose aussi une fonction `or_plot` pour présenter les odd ratios obtenus sous forme de graphique.

```
R> var_label(d$heures.tv) <- "Nombre d'heures\nde TV par jour"
or_plot(d, dep, vars_multi)
```

Waiting for profiling to be done...  
 Waiting for profiling to be done...  
 Waiting for profiling to be done...

Warning: Removed 4 rows containing missing values  
 (geom\_errorbarh).

Pratique du sport ?: OR (95% CI, p-value)

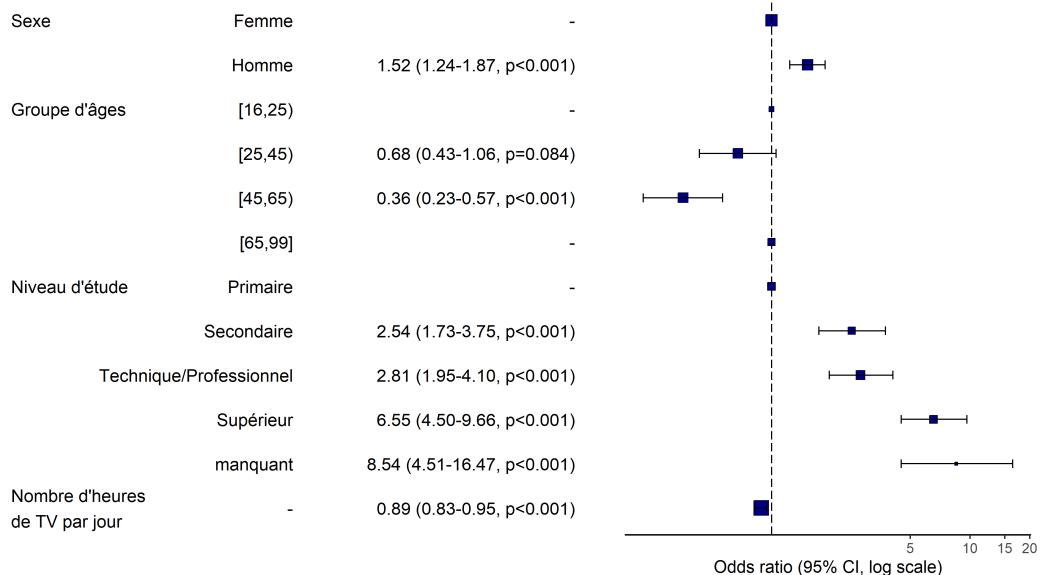


Figure 7. Graphique des odds ratios obtenu avec or\_plot

**ATTENTION:** `or_plot` n'est pas compatible avec les effets d'interaction (cf. ci-dessous).

## Effets d'interaction dans le modèle

Voir le chapitre dédié aux effets d'interaction, page 617.

## Multicolinéarité

Voir le chapitre dédié, page 645.



# Analyse des correspondances multiples (ACM)

Principe général . . . . .	558
ACM avec ade4 . . . . .	559
ACM avec FactoMineR . . . . .	580
Extensions complémentaires . . . . .	591
factoextra . . . . .	591
explor . . . . .	591
Factoshiny . . . . .	592
FactoInvestigate . . . . .	593
Voir aussi . . . . .	593

Il existe plusieurs techniques d'analyse factorielle dont les plus courantes sont l'[analyse en composante principale \(ACP\)](#) portant sur des variables quantitatives, l'[analyse factorielle des correspondances \(AFC\)](#) portant sur deux variables qualitatives et l'[analyse des correspondances multiples \(ACM\)](#) portant sur plusieurs variables qualitatives (il s'agit d'une extension de l'AFC). Pour combiner des variables à la fois quantitatives et qualitatives, on pourra avoir recours à l'analyse factorielle avec données mixtes.

Bien que ces techniques soient disponibles dans les extensions standards de R, il est souvent préférable d'avoir recours à deux autres extensions plus complètes, [ade4](#) et [FactoMineR](#), chacune ayant ses avantages et des possibilités différentes. Voici les fonctions les plus fréquentes :

Analyse	Variables	Fonction standard	Fonction ade4	Fonctions FactoMineR
ACP	plusieurs variables quantitatives	<code>princomp</code>	<code>dudi.pca</code>	<code>PCA</code>
AFC	deux variables qualitatives	<code>corresp</code>	<code>dudi.coa</code>	<code>CA</code>
ACM	plusieurs variables qualitatives	<code>mca</code>	<code>dudi.acm</code>	<code>MCA</code>
Analyse mixte	plusieurs variables quantitatives et/ou qualitatives	<code>FAMD</code>	<code>dudi.mix</code>	—

Dans la suite de ce chapitre, nous n'arboderons que l'analyse des correspondances multiples (ACM).

#### NOTE

On trouvera également de nombreux supports de cours en français sur l'analyse factorielle sur le site de François Gilles Carpentier : <http://geai.univ-brest.fr/~carpenti/>.

## Principe général

L'analyse des correspondances multiples est une technique descriptive visant à résumer l'information contenu dans un grand nombre de variables afin de faciliter l'interprétation des corrélations existantes entre ces différentes variables. On cherche à savoir quelles sont les modalités corrélées entre elles.

L'idée générale est la suivante<sup>1</sup>. L'ensemble des individus peut être représenté dans un espace à plusieurs dimensions où chaque axe représente les différentes variables utilisées pour décrire chaque individu. Plus précisément, pour chaque variable qualitative, il y a autant d'axes que de modalités moins un. Ainsi il faut trois axes pour décrire une variable à quatre modalités. Un tel nuage de points est aussi difficile à interpréter que de lire directement le fichier de données. On ne voit pas les corrélations qu'il peut y avoir entre modalités, par exemple qu'aller au cinéma est plus fréquent chez les personnes habitant en milieu urbain. Afin de mieux représenter ce nuage de points, on va procéder à un changement de systèmes de coordonnées. Les individus seront dès lors projetés et représentés sur un nouveau système d'axe. Ce nouveau système d'axes est choisi de telle manière que la majorité des variations soit concentrées sur les premiers axes. Les deux-trois premiers axes permettront d'expliquer la majorité des différences observées dans l'échantillon, les autres axes n'apportant qu'une faible part additionnelle d'information. Dès lors, l'analyse pourra se concentrer sur ses premiers axes qui constitueront un bon résumé des variations observables dans l'échantillon.

1. Pour une présentation plus détaillée, voir  
<http://www.math.univ-toulouse.fr/~baccini/zpedago/asdm.pdf>.

Avant toute ACM, il est indispensable de réaliser une analyse préliminaire de chaque variable, afin de voir si toutes les classes sont aussi bien représentées ou s'il existe un déséquilibre. L'ACM est sensible aux effectifs faibles, aussi il est préférable de regrouper les classes peu représentées le cas échéant.

## ACM avec ade4

Si l'extension **ade4** n'est pas présente sur votre PC, il vous faut l'installer :

```
R> install.packages("ade4", dep = TRUE)
```

Dans tous les cas, il faut penser à la charger en mémoire :

```
R> library(ade4)
```

Comme précédemment, nous utiliserons le fichier de données `hdv2003` fourni avec l'extension **questionr**.

```
R> library(questionr)
data(hdv2003)
d <- hdv2003
```

En premier lieu, comme dans le chapitre sur la régression logistique, page 507, nous allons créer une variable groupe d'âges et regrouper les modalités de la variable « niveau d'étude ».

```
R> d$grpage <- cut(d$age, c(16, 25, 45, 65, 93), right = FALSE, include.lowest =
TRUE)
d$etud <- d$nivetud
levels(d$etud) <- c(
  "Primaire", "Primaire", "Primaire", "Secondaire", "Secondaire",
  "Technique/Professionnel", "Technique/Professionnel", "Supérieur"
)
```

Ensuite, nous allons créer un tableau de données ne contenant que les variables que nous souhaitons prendre en compte pour notre analyse factorielle.

```
R> d2 <- d[, c("grpage", "sexe", "etud", "peche.chasse", "cinema", "cuisine", "bricol",
  "sport", "lecture.bd")]
```

Le calcul de l'ACM se fait tout simplement avec la fonction `dudi.acm`.

```
R> acm <- dudi.acm(d2)
```

Par défaut, la fonction affichera le graphique des valeurs propres de chaque axe (nous y reviendrons) et vous demandera le nombre d’axes que vous souhaitez conserver dans les résultats. Le plus souvent, cinq axes seront largement plus que suffisants. Vous pouvez également éviter cette étape en indiquant directement à `dudi.acm` de vous renvoyer les cinq premiers axes ainsi :

```
R> acm <- dudi.acm(d2, scannf = FALSE, nf = 5)
```

Le graphique des valeurs propres peut être reproduit avec `screeplot` :

```
R> screeplot(acm)
```

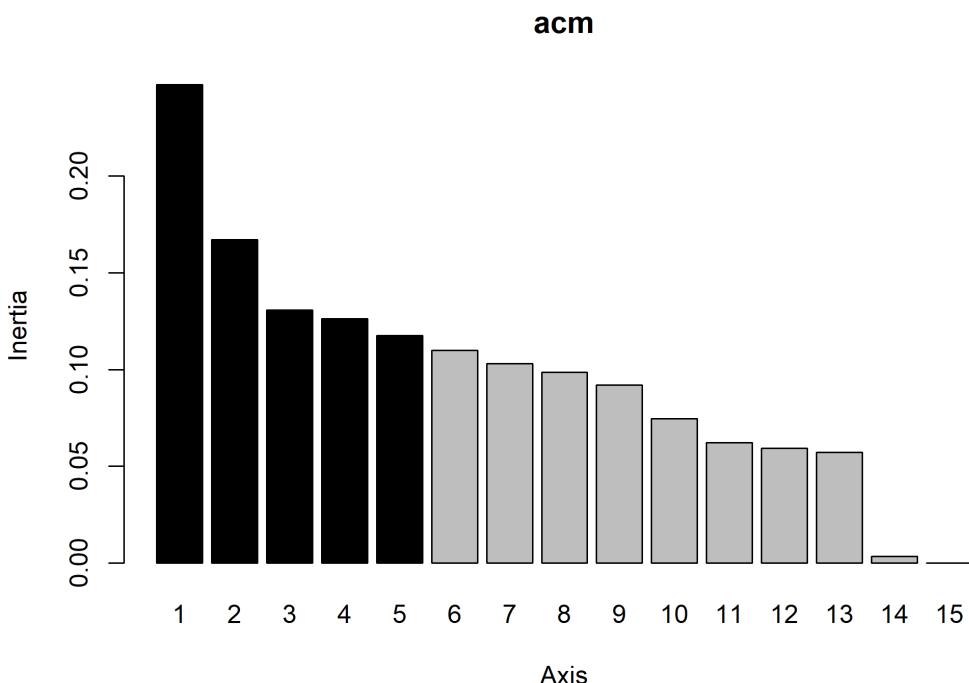


Figure 1. Valeurs propres ou inerties de chaque axe

Les mêmes valeurs pour les premiers axes s’obtiennent également avec `summary`<sup>2</sup>:

2. On pourra également avoir recours à la fonction `inertia.dudi` pour l’ensemble des axes.

```
R> summary(acm)
```

Class: acm dudi  
 Call: dudi.acm(df = d2, scannf = FALSE, nf = 5)

Total inertia: 1.451

Eigenvalues:

Ax1	Ax2	Ax3	Ax4	Ax5
0.2474	0.1672	0.1309	0.1263	0.1176

Projected inertia (%):

Ax1	Ax2	Ax3	Ax4	Ax5
17.055	11.525	9.022	8.705	8.109

Cumulative projected inertia (%):

Ax1	Ax1:2	Ax1:3	Ax1:4	Ax1:5
17.06	28.58	37.60	46.31	54.42

(Only 5 dimensions (out of 15) are shown)

L'inertie totale est de 1,451 et l'axe 1 en explique 0,1474 soit 17 %. L'inertie projetée cumulée nous indique que les deux premiers axes expliquent à eux seuls 29 % des variations observées dans notre échantillon.

Pour comprendre la signification des différents axes, il importe d'identifier quelles sont les variables/modalités qui contribuent le plus à chaque axe. Une première représentation graphique est le cercle de corrélation des modalités. Pour cela, on aura recours à `s.corcircle`. On indiquera d'abord `acm$co` si l'on souhaite représenter les modalités ou `acm$li` si l'on souhaite représenter les individus. Les deux chiffres suivant indiquent les deux axes que l'on souhaite afficher (dans le cas présent les deux premiers axes). Enfin, le paramètre `clabel` permet de modifier la taille des étiquettes.

```
R> s.corcircle(acm$co, 1, 2, clabel = 0.7)
```

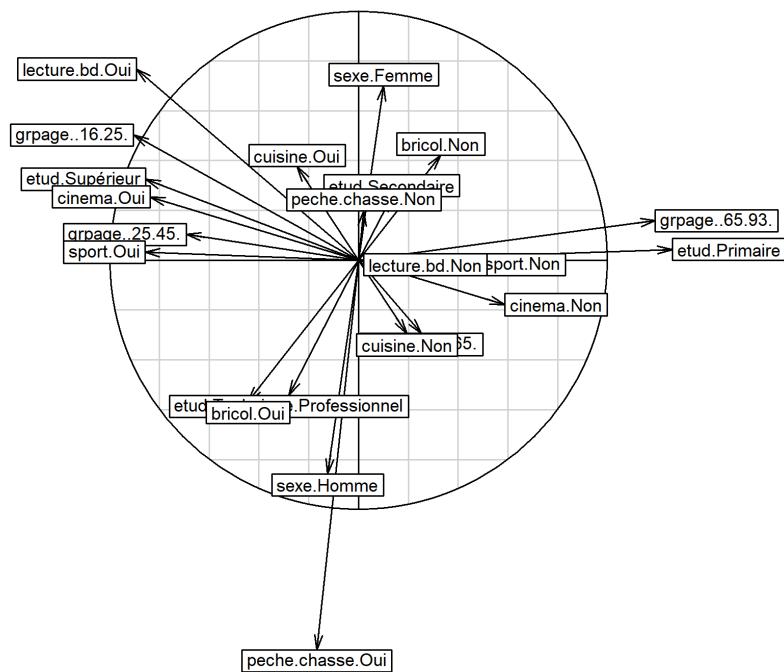


Figure 2. Cercle de corrélations des modalités sur les deux premiers axes

On pourra avoir également recours à `boxplot` pour visualiser comment se répartissent les modalités de chaque variable sur un axe donné<sup>3</sup>.

3. La fonction `score` constituera également une aide à l'interprétation des axes.

```
R> boxplot(acm)
```

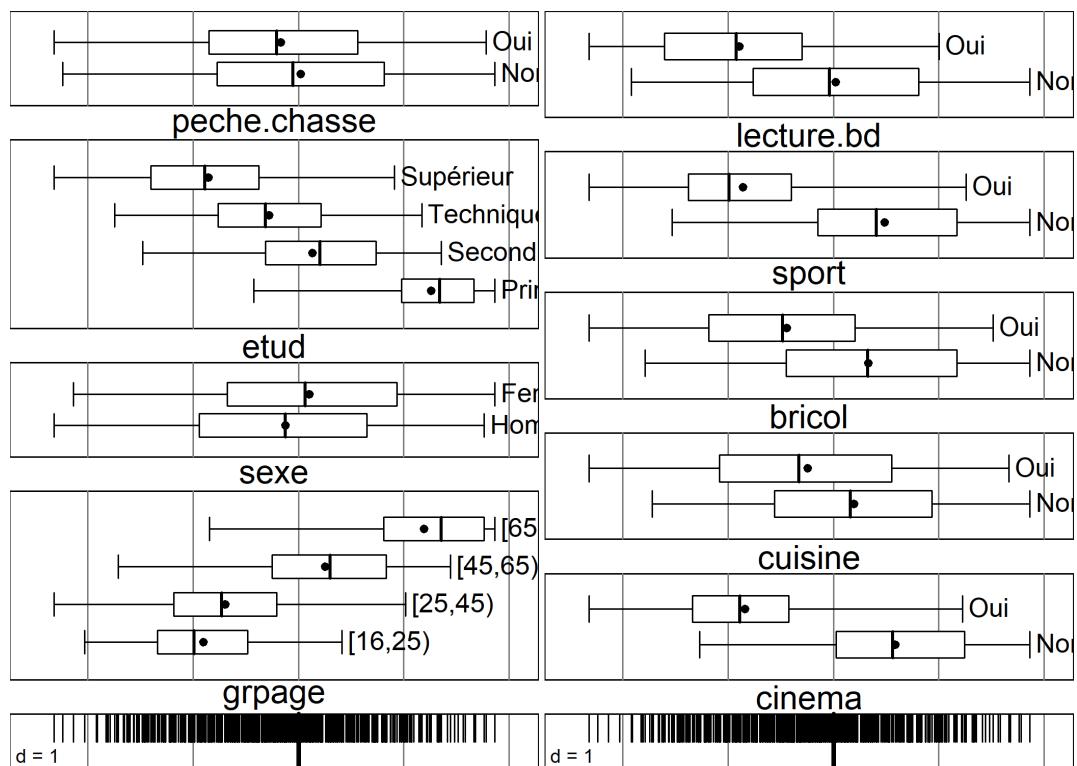


Figure 3. Répartition des modalités selon le premier axe

```
R> boxplot(acm, 2)
```

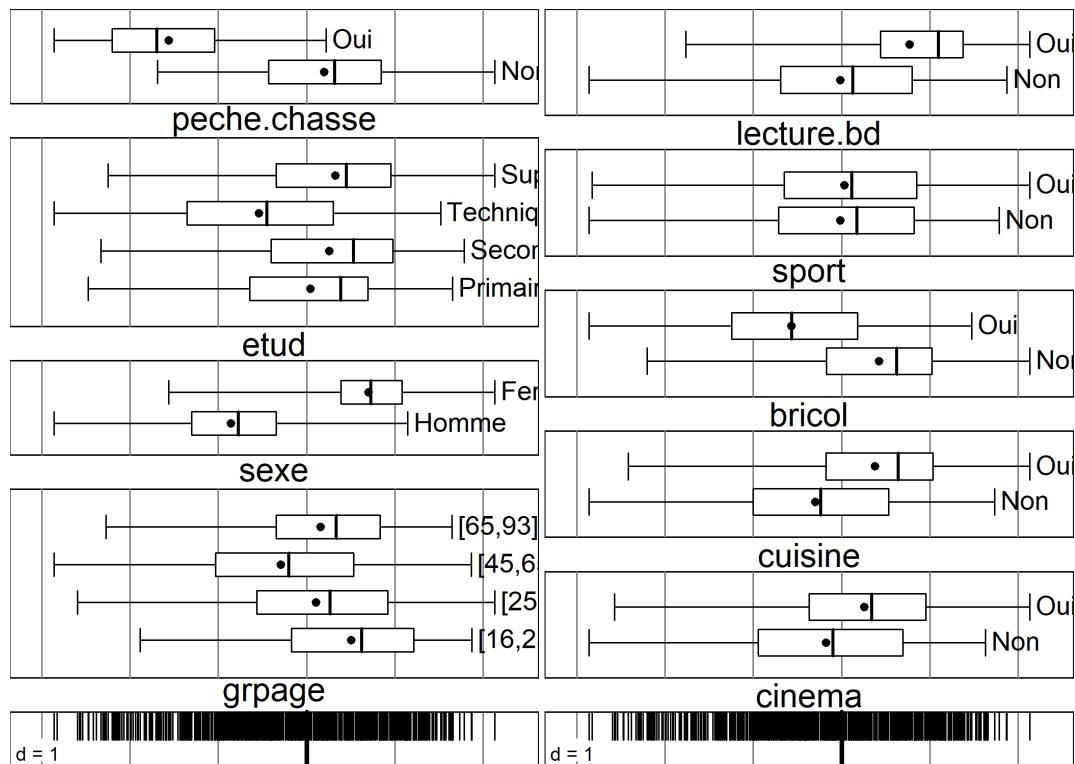
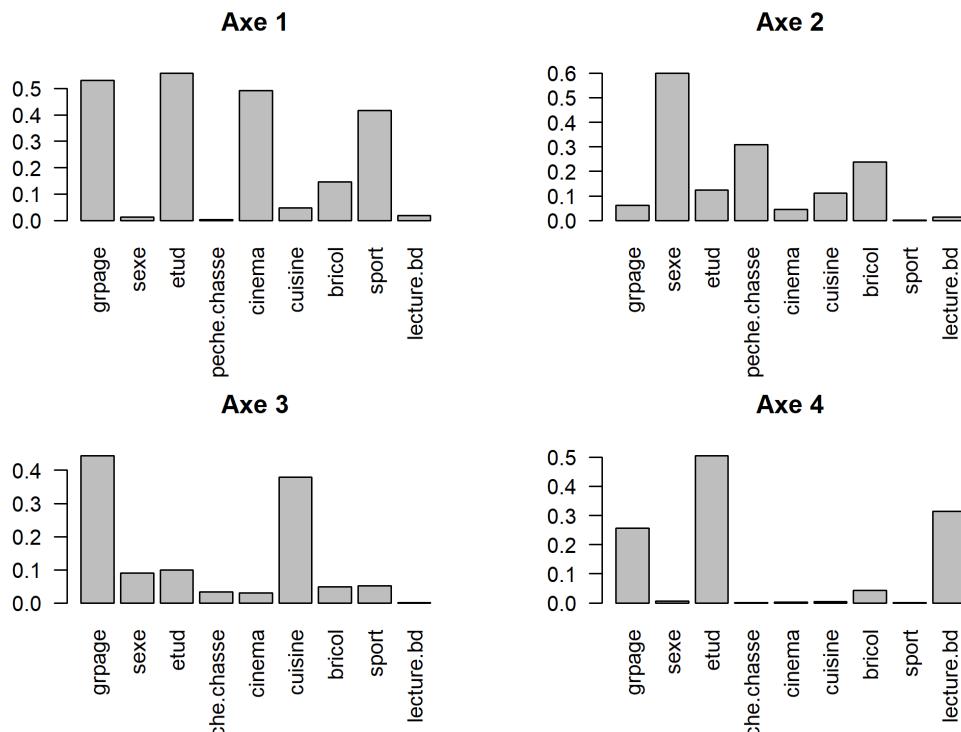


Figure 4. Répartition des modalités selon le second axe

Le tableau `acm$cr` contient les rapports de corrélation (variant de 0 à 1) entre les variables et les axes choisis au départ de l'ACM. Pour représenter graphiquement ces rapports, utiliser la fonction `barplot` ainsi : `barplot(acm$cr[, num], names.arg=rownames(acm$cr), las=2)` où `num` est le numéro de l'axe à représenter. Pour l'interprétation des axes, se concentrer sur les variables les plus structurantes, c'est-à-dire dont le rapport de corrélation est le plus proche de 1.

```
R> par(mfrow = c(2, 2))
for (i in 1:4) barplot(acm$cr[, i], names.arg = row.names(acm$cr), las =
2, main = paste("Axe", i))
```



```
R> par(mfrow = c(1, 1))
```

Figure 5. Rapports de corrélation des variables sur les 4 premiers axes

#### NOTE

Le paramètre `mfrow` de la fonction `par` permet d'indiquer à R que l'on souhaite afficher plusieurs graphiques sur une seule et même fenêtre, plus précisément que l'on souhaite diviser la fenêtre en deux lignes et deux colonnes.

Dans l'exemple précédent, après avoir produit notre graphique, nous avons réinitialisé cette valeur à `c(1, 1)` (un seul graphique par fenêtre) pour ne pas affecter les prochains graphiques que nous allons produire.

Pour représenter, les modalités dans le plan factoriel, on utilisera la fonction `s.label`. Par défaut, les deux premiers axes sont représentés.

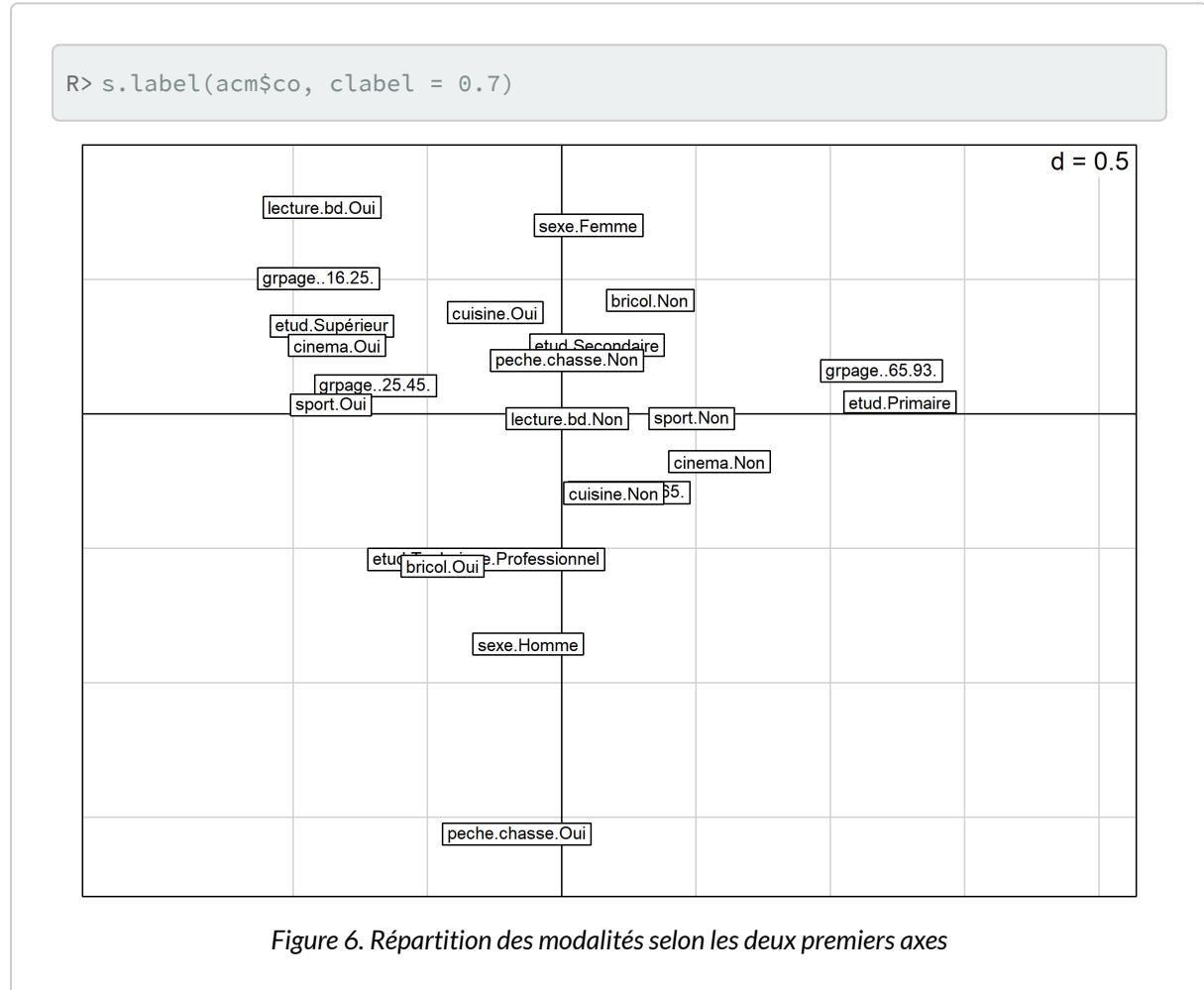


Figure 6. Répartition des modalités selon les deux premiers axes

Il est bien sûr possible de préciser les axes à représenter. L'argument `boxes` permet quant à lui d'indiquer si l'on souhaite tracer une boîte pour chaque modalité.

```
R> s.label(acm$co, 3, 4, clabel = 0.7, boxes = FALSE)
```



Figure 7. Répartition des modalités selon les axes 3 et 4

Bien entendu, on peut également représenter les individus. En indiquant `clabel=0` (une taille nulle pour les étiquettes), `s.label` remplace chaque observation par un symbole qui peut être spécifié avec `pch`.

```
R> s.label(acm$li, clabel = 0, pch = 17)
```

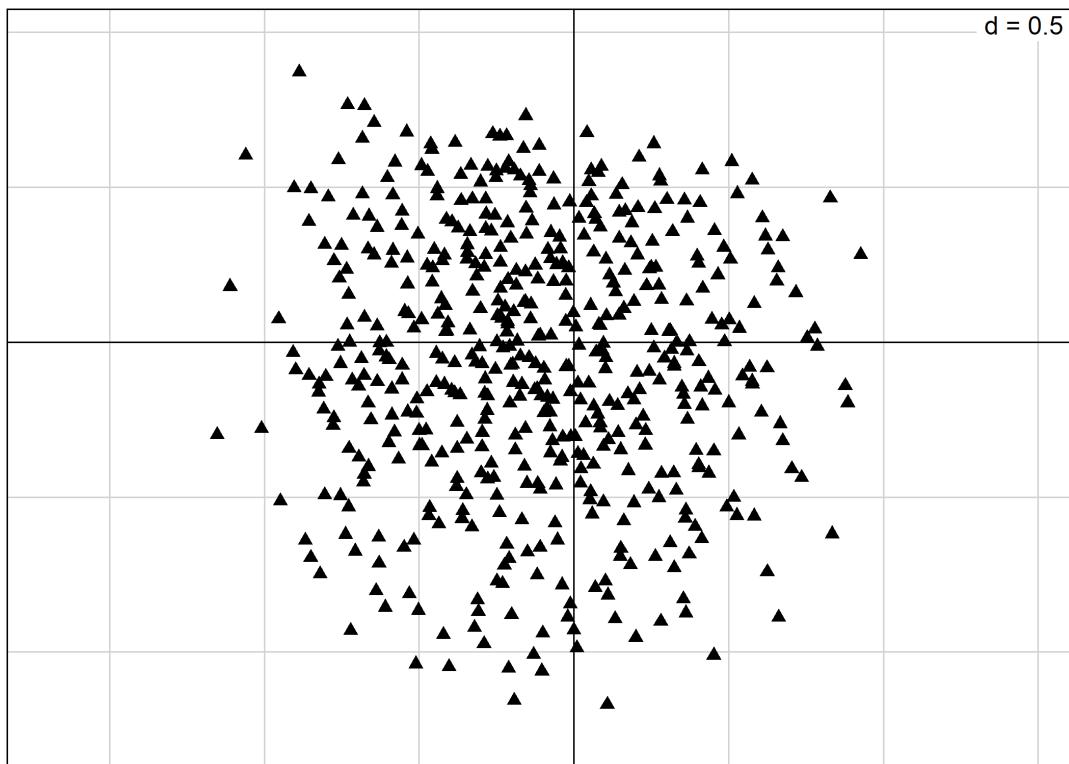
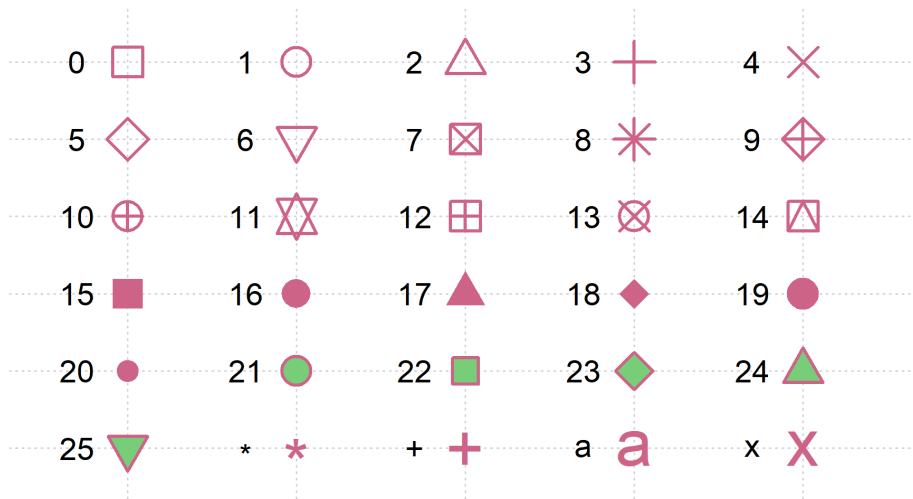


Figure 8. Répartition des individus selon les deux premiers axes

**NOTE**

L'argument `pch` permet de spécifier le symbole à utiliser. Il peut prendre soit un nombre entier compris entre 0 et 25, soit un caractère textuel.

**Différentes valeurs possibles pour l'argument pch**



#### NOTE

Lorsque l’on réalise une ACM, il n’est pas rare que plusieurs observations soient identiques, c'est-à-dire correspondent à la même combinaison de modalités. Dès lors, ces observations seront projetées sur le même point dans le plan factoriel. Une représentation classique des observations avec `s.label` ne permettra pas de rendre compte les effectifs de chaque point.

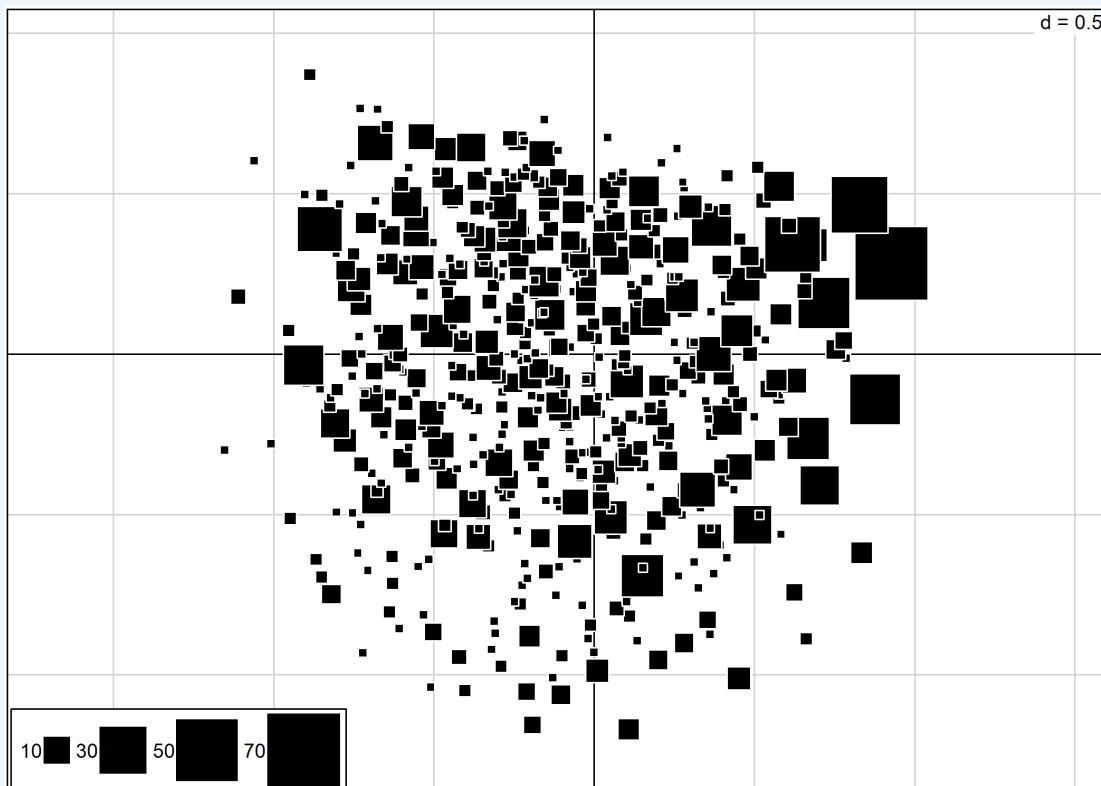
Le package **JLutils**, disponible seulement sur [GitHub](#), propose une fonction `s.freq` représentant chaque point par un carré proportionnel au nombre d’individus.

Pour installer **JLutils**, on aura recours au package **devtools** et à sa fonction `install_github` :

```
R> library(devtools)
install_github("lamarange/JLutils")
```

La fonction `s.freq` s’emploie de manière similaire aux autres fonctions graphiques de **ade4**. Le paramètre `cscale` permet d’ajuster la taille des carrés.

```
R> library(JLutils)
s.freq(acm$li)
```



L'interprétation est tout autre, non ?

#### NOTE

Gaston Sanchez propose un graphique amélioré des modalités dans le plan factoriel à cette adresse : <http://rpubs.com/gaston/MCA>.

La fonction `s.value` permet notamment de représenter un troisième axe factoriel. Dans l'exemple ci-après, nous projettons les individus selon les deux premiers axes factoriels. La taille et la couleur des carrés dépendent pour leur part de la coordonnée des individus sur le troisième axe factoriel. Le paramètre `csi` permet d'ajuster la taille des carrés.

```
R> s.value(acm$li, acm$li[, 3], 1, 2, csi = 0.5)
```

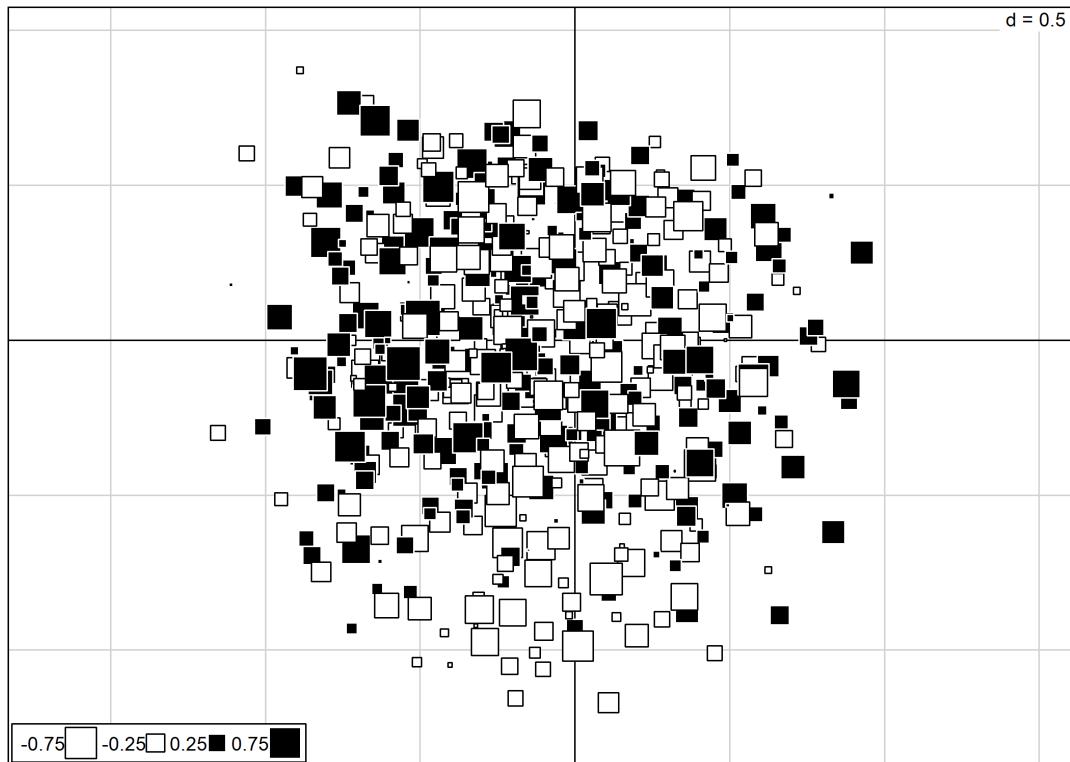


Figure 9. Répartition des individus selon les trois premiers axes

**s.arrow** permet de représenter les vecteurs variables ou les vecteurs individus sous la forme d'une flèche allant de l'origine du plan factoriel aux coordonnées des variables/individus :

```
R> s.arrow(acm$co, clabel = 0.7)
```

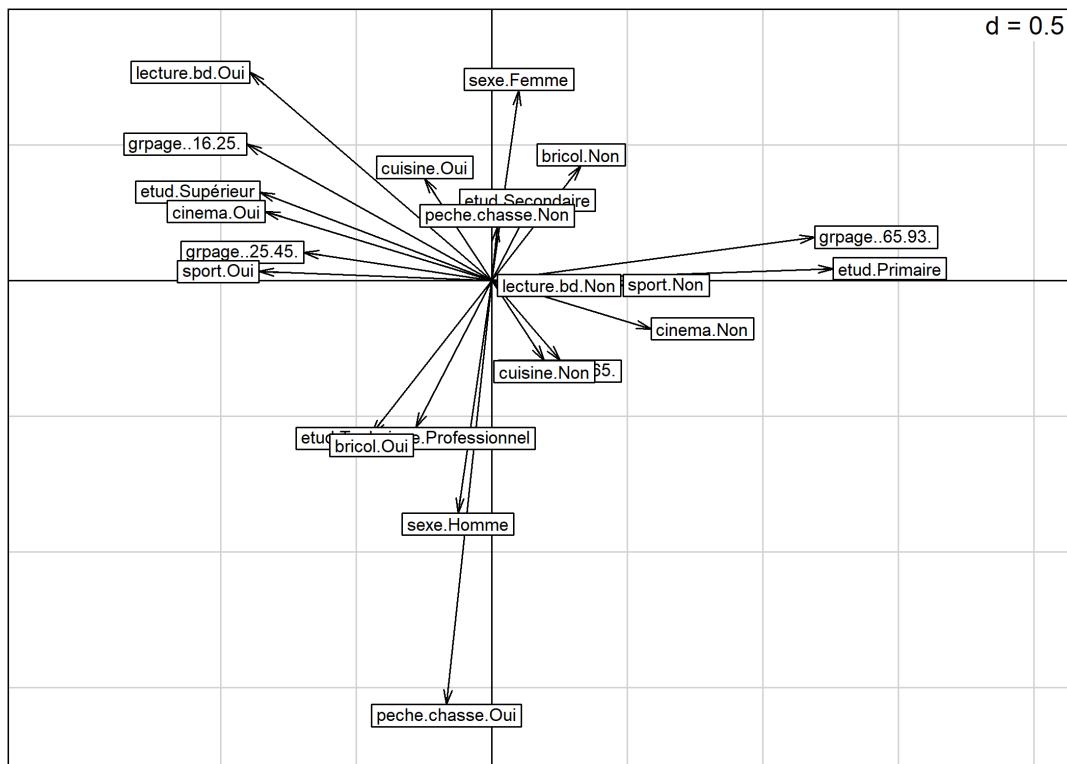


Figure 10. Vecteurs des modalités selon les deux premiers axes

`s.hist` permet de représenter des individus (ou des modalités) sur le plan factoriel et d'afficher leur distribution sur chaque axe :

```
R> s.hist(acm$li, clabel = 0, pch = 15)
```

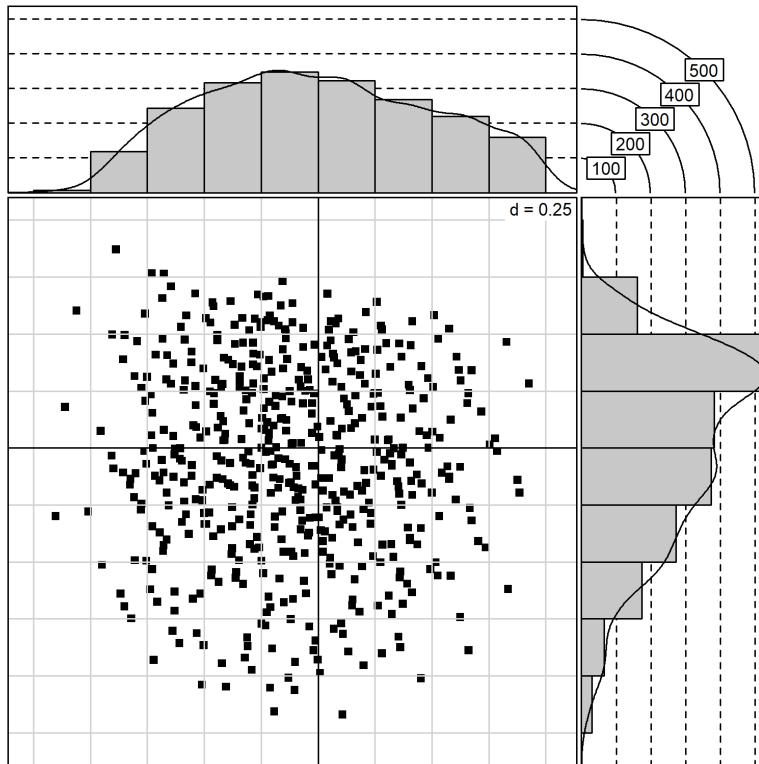


Figure 11. Distribution des individus dans le plan factoriel

`s.class` et `s.chull` permettent de représenter les différentes observations classées en plusieurs catégories. Cela permet notamment de projeter certaines variables.

`s.class` représente les observations par des points, lie chaque observation au barycentre de la modalité à laquelle elle appartient et dessine une ellipse représentant la forme générale du nuage de points :

```
R> library(RColorBrewer)
s.class(acm$li, d2$sex, col = brewer.pal(4, "Set1"))
```

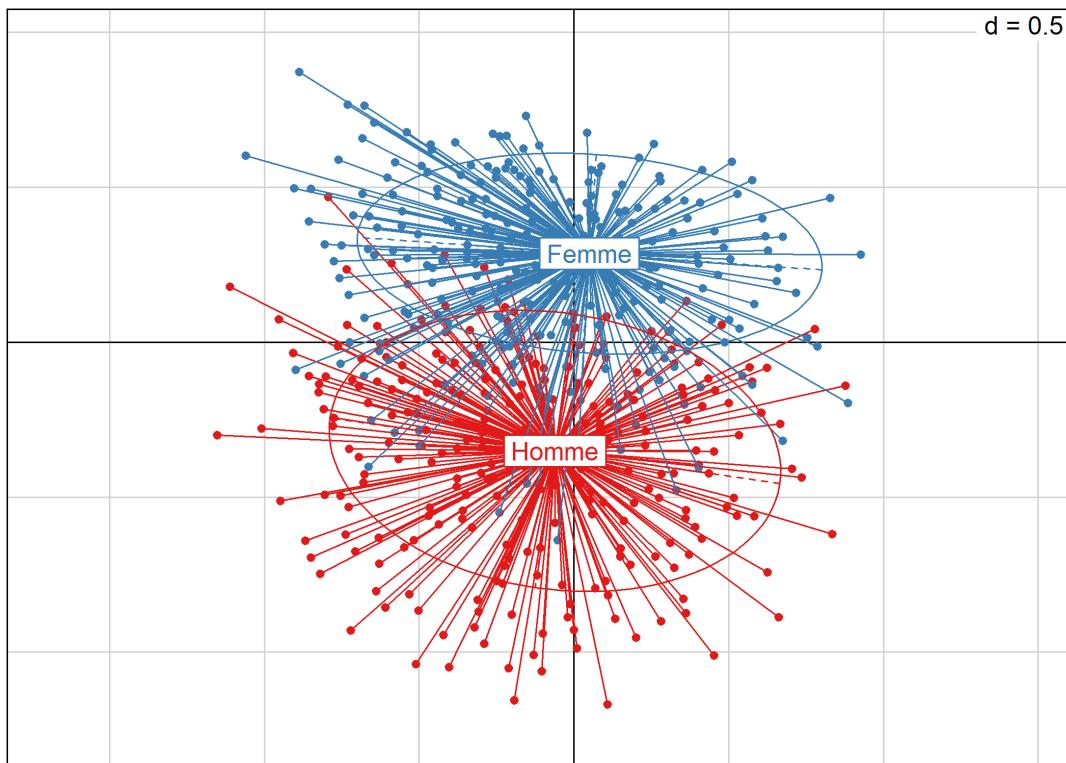


Figure 12. Individus dans le plan factoriel selon le sexe (s.class)

`s.chull` représente les barycentres de chaque catégorie et dessine des lignes de niveaux représentant la distribution des individus de cette catégorie. Les individus ne sont pas directement représentés :

```
R> s.chull(acm$li, d2$sex, col = brewer.pal(4, "Set1"))
```

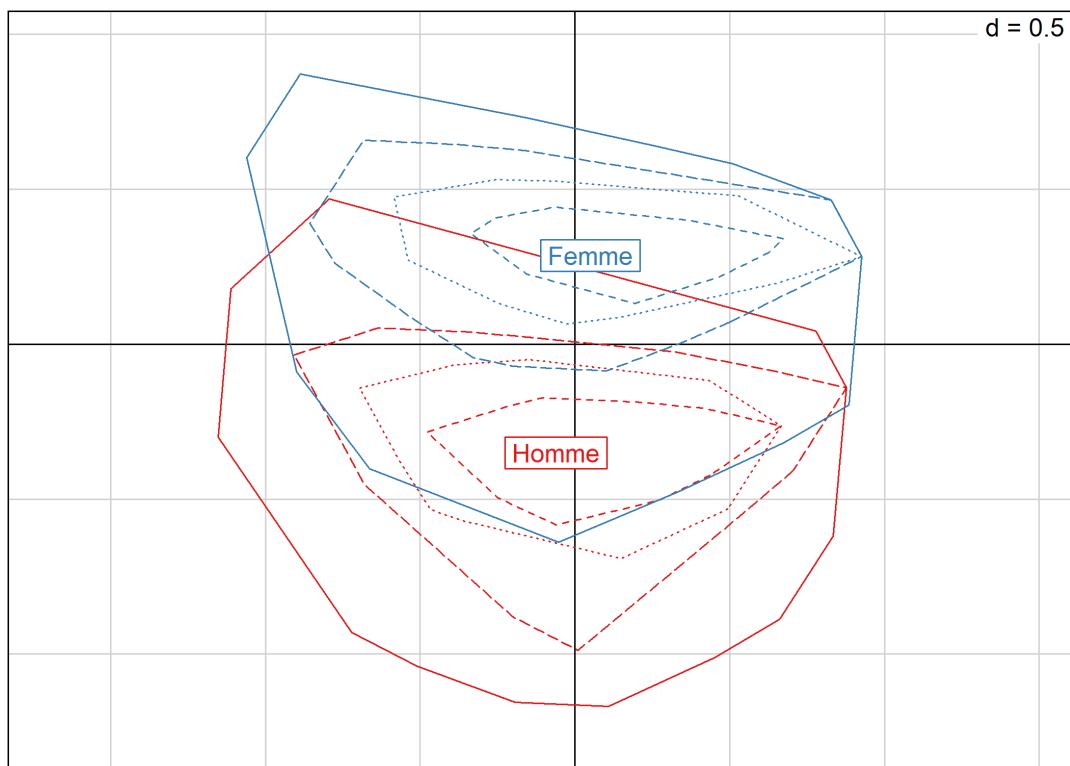
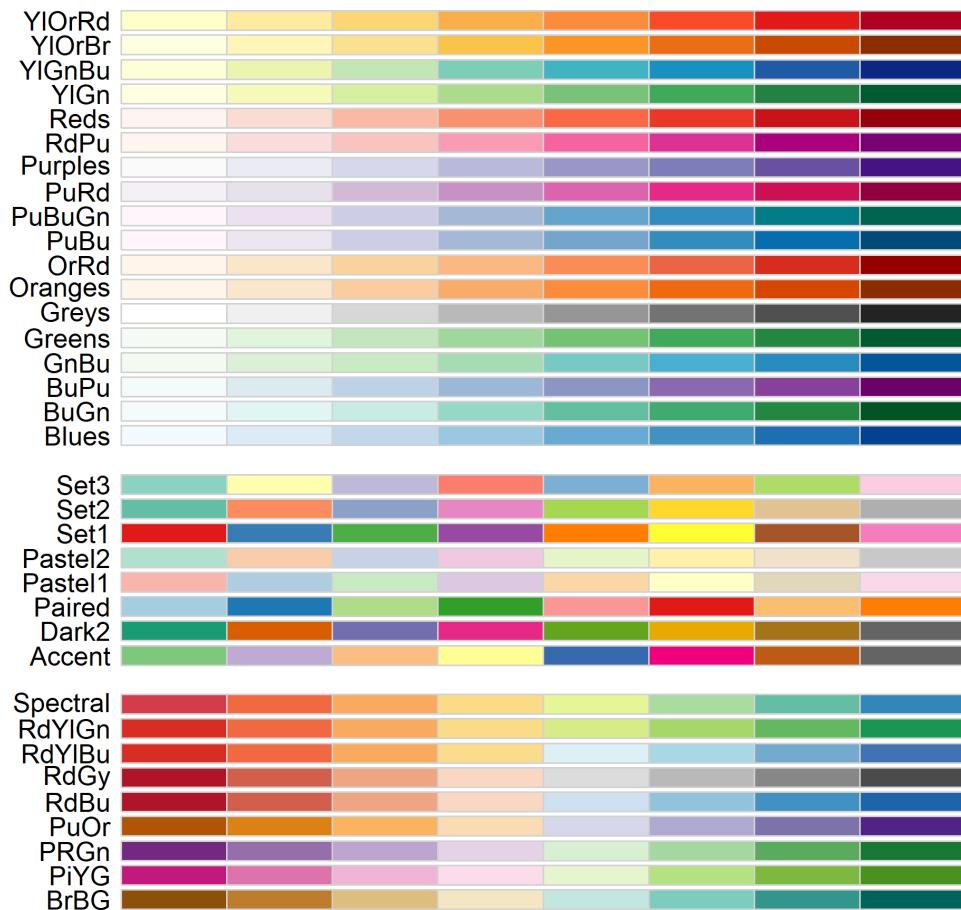


Figure 13. Individus dans le plan factoriel selon le sexe (s.chull)

**NOTE**

Il est préférable de fournir une liste de couleurs (via le paramètre `col`) pour rendre le graphique plus lisible. Si vous avez installé l'extension **RColorBrewer**, vous pouvez utiliser les différentes palettes de couleurs proposées. Pour afficher les palettes disponibles, utilisez `display.brewer.all`.

```
R> library(RColorBrewer)
display.brewer.all(8)
```



Pour obtenir une palette de couleurs, utilisez la fonction `brewer.pal` avec les arguments `n` (nombre de couleurs demandées) et `pal` (nom de la palette de couleurs désirée).

Pour plus d'informations sur les palettes *Color Brewer*, voir <http://colorbrewer2.org/>.

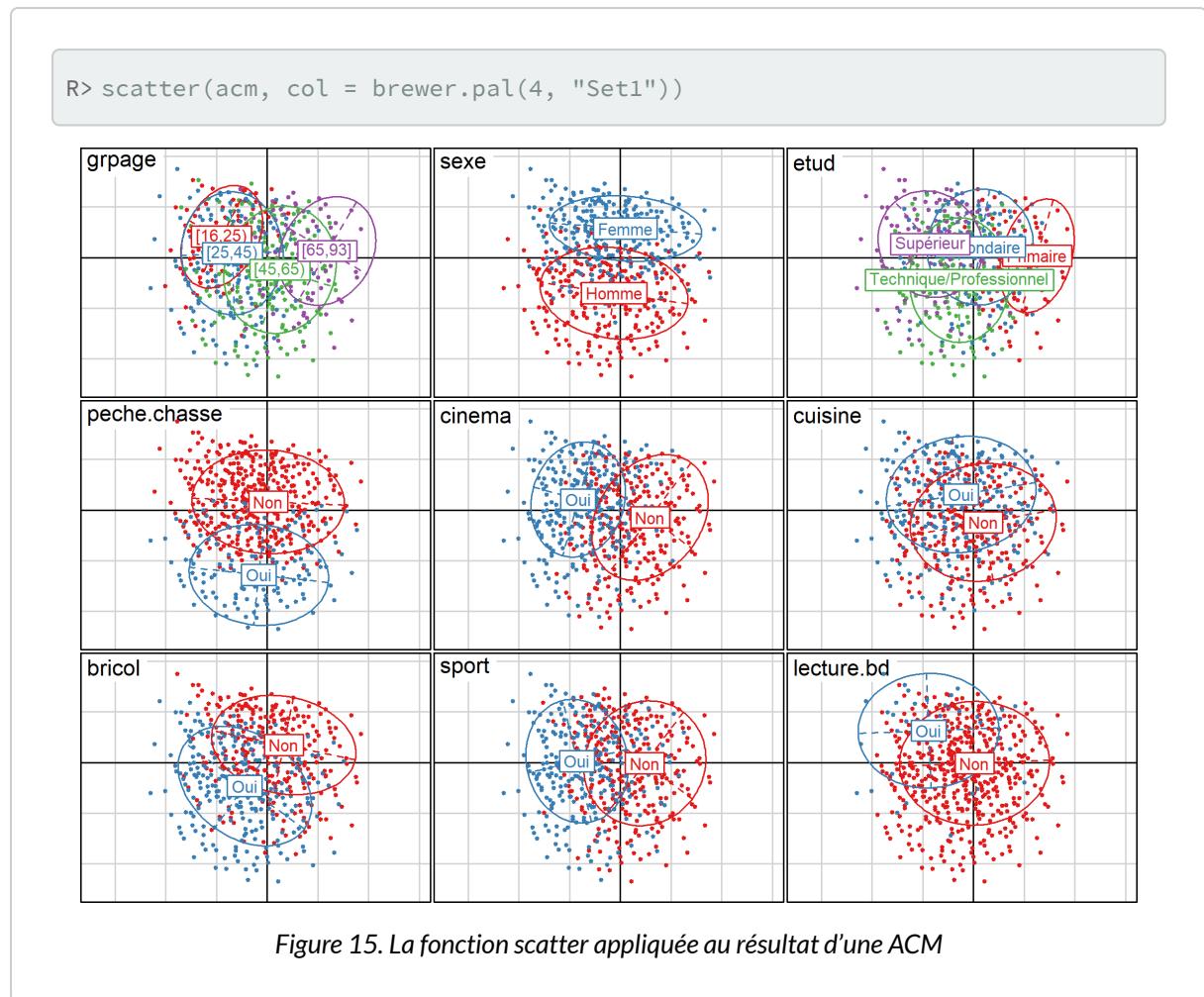
La variable catégorielle transmise à `s.class` ou `s.chull` n'est pas obligatoirement une des variables retenues pour l'ACM. Il est tout à fait possible d'utiliser une autre variable. Par exemple :

```
R> s.class(acm$li, d$trav.imp, col = brewer.pal(4, "Set1"))
```



Figure 14. Individus dans le plan factoriel selon l'importance donnée au travail

Les fonctions `scatter` et `biplot` sont équivalentes : elles appliquent `s.class` à chaque variable utilisée pour l'ACM.



#### NOTE

L'extension `explor` écrite par Julien Barnier offre une interface graphique interactive permettant d'explorer les résultats d'une analyse factorielle. Essayons donc la fonction `explor`. C'est magique !

```
R> library(explor)
explor(acm)
```

## ACM avec FactoMineR

Comme avec `ade4`, il est nécessaire de préparer les données au préalable (voir section précédente).

L'ACM se calcule avec la fonction `MCA`, l'argument `ncp` permettant de choisir le nombre d'axes à retenir :

```
R> library(FactoMineR)
```

```
R> acm2 <- MCA(d2, ncp = 5, graph = FALSE)
acm2
```

\*\*Results of the Multiple Correspondence Analysis (MCA)\*\*  
The analysis was performed on 2000 individuals, described by 9 variables  
\*The results are available in the following objects:

name	description
1 "\$eig"	"eigenvalues"
2 "\$var"	"results for the variables"
3 "\$var\$coord"	"coord. of the categories"
4 "\$var\$cos2"	"cos2 for the categories"
5 "\$var\$contrib"	"contributions of the categories"
6 "\$var\$v.test"	"v-test for the categories"
7 "\$ind"	"results for the individuals"
8 "\$ind\$coord"	"coord. for the individuals"
9 "\$ind\$cos2"	"cos2 for the individuals"
10 "\$ind\$contrib"	"contributions of the individuals"
11 "\$call"	"intermediate results"
12 "\$call\$marge.col"	"weights of columns"
13 "\$call\$marge.li"	"weights of rows"

```
R> acm2$eig
```

eigenvalue percentage of variance	
dim 1	0.25757489                    15.454493
dim 2	0.18363502                    11.018101
dim 3	0.16164626                    9.698776
dim 4	0.12871623                    7.722974
dim 5	0.12135737                    7.281442
dim 6	0.11213331                    6.727999
dim 7	0.10959377                    6.575626
dim 8	0.10340564                    6.204338
dim 9	0.09867478                    5.920487
dim 10	0.09192693                    5.515616
dim 11	0.07501208                    4.500725
dim 12	0.06679676                    4.007805
dim 13	0.06002063                    3.601238
dim 14	0.05832024                    3.499215
dim 15	0.03785276                    2.271166
cumulative percentage of variance	

```
dim 1          15.45449
dim 2          26.47259
dim 3          36.17137
dim 4          43.89434
dim 5          51.17579
dim 6          57.90378
dim 7          64.47941
dim 8          70.68375
dim 9          76.60424
dim 10         82.11985
dim 11         86.62058
dim 12         90.62838
dim 13         94.22962
dim 14         97.72883
dim 15        100.00000
```

```
R> sum(acm2$eig[, 1])
```

```
[1] 1.666667
```

En premier lieu, il apparait que l'inertie totale obtenue avec `MCA` est différente de celle observée avec `dudi.acm`. Cela est dû à un traitement différents des valeurs manquantes. Alors que `dudi.acm` exclut les valeurs manquantes, `MCA` les considèrent, par défaut, comme une modalité additionnelle. Pour calculer l'ACM uniquement sur les individus n'ayant pas de valeur manquante, on aura recours à `complete.cases` :

```
R> acm2 <- MCA(d2[complete.cases(d2), ], ncp = 5, graph = FALSE)
acm2$eig
```

	eigenvalue	percentage of variance
dim 1	0.24790700	17.162792
dim 2	0.16758465	11.602014
dim 3	0.13042357	9.029324
dim 4	0.12595105	8.719688
dim 5	0.11338629	7.849820
dim 6	0.10976674	7.599236
dim 7	0.10060204	6.964757
dim 8	0.09802387	6.786268
dim 9	0.09283131	6.426783
dim 10	0.07673502	5.312425
dim 11	0.06609694	4.575942
dim 12	0.05950655	4.119684
dim 13	0.05562942	3.851267

```
cumulative percentage of variance
dim 1                17.16279
dim 2                28.76481
dim 3                37.79413
dim 4                46.51382
dim 5                54.36364
dim 6                61.96287
dim 7                68.92763
dim 8                75.71390
dim 9                82.14068
dim 10               87.45311
dim 11               92.02905
dim 12               96.14873
dim 13              100.00000
```

```
R> sum(acm2$eig[, 1])
```

```
[1] 1.444444
```

Les possibilités graphiques de **FactoMineR** sont différentes de celles de **ade4**. Un recours à la fonction `plot` affichera par défaut les individus, les modalités et les variables. La commande `?plot.MCA` permet d'accéder au fichier d'aide de cette fonction (i.e. de la méthode générique `plot` appliquée aux objets de type `MCA`) et de voir toutes les options graphiques. L'argument `choix` permet de spécifier ce que l'on souhaite afficher (« `ind` » pour les individus et les catégories, « `var` » pour les variables). L'argument `invisible` quant à lui permet de spécifier ce que l'on souhaite masquer. Les axes à afficher se précisent avec `axes`. Voir les exemples ci-dessous.

```
R> plot(acm2)
```

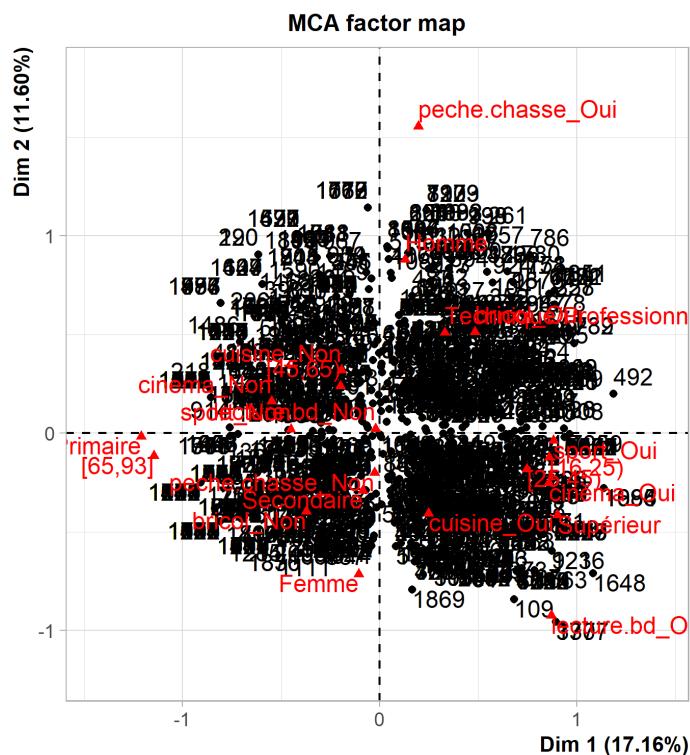


Figure 16. Plan factoriel (deux premiers axes)

```
R> plot(acm2, axes = c(3, 4))
```

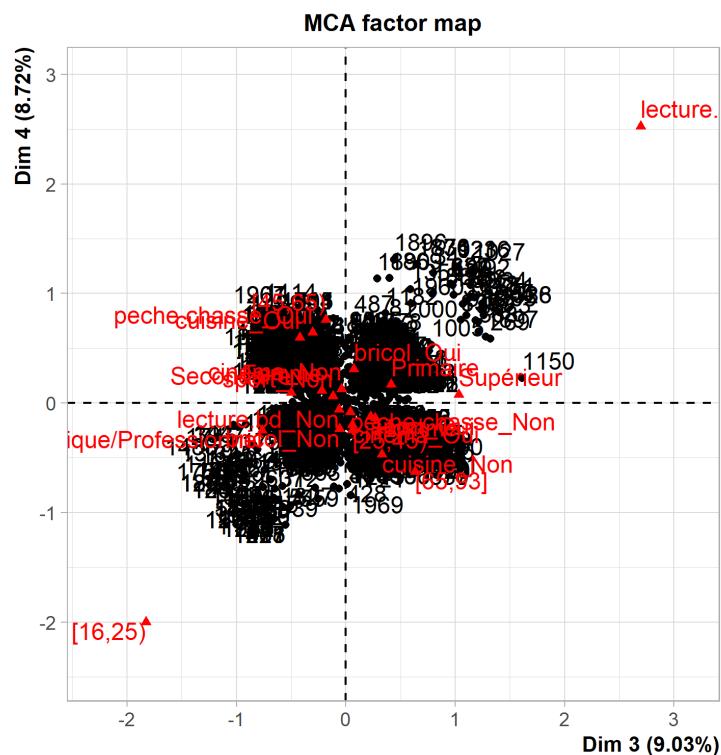


Figure 17. Plan factoriel (axes 3 et 4)

```
R> plot(acm2, choix = "ind")
```

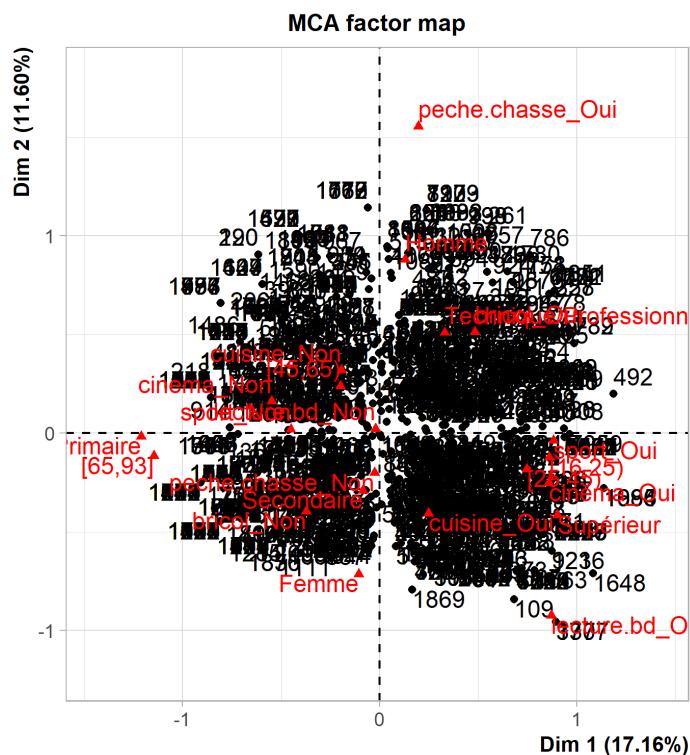


Figure 18. Plan factoriel (seulement les individus et les catégories)

```
R> plot(acm2, choix = "ind", invisible = "ind")
```

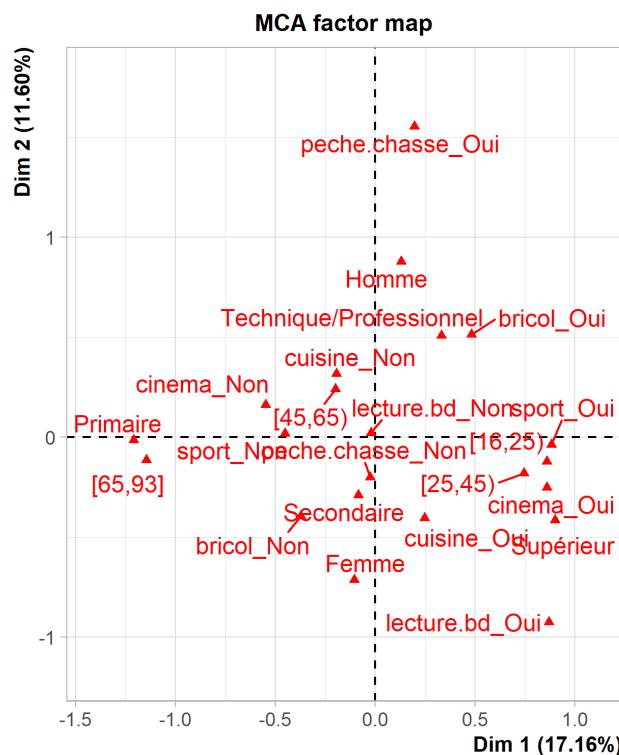


Figure 19. Plan factoriel (seulement les catégories)

```
R> plot(acm2, choix = "var")
```

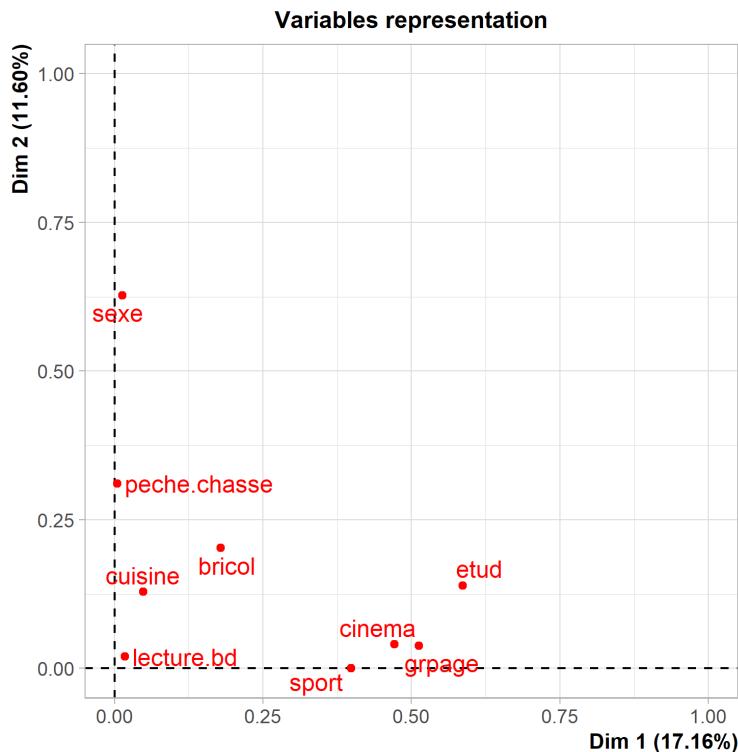


Figure 20. Plan factoriel (seulement les variables)

La fonction `plotellipses` trace des ellipses de confiance autour des modalités de variables qualitatives. L'objectif est de voir si les modalités d'une variable qualitative sont significativement différentes les unes des autres.

Par défaut (`means=TRUE`), les ellipses de confiance sont calculées pour les coordonnées moyennes de chaque catégorie.

```
R> plotellipses(acm2)
```

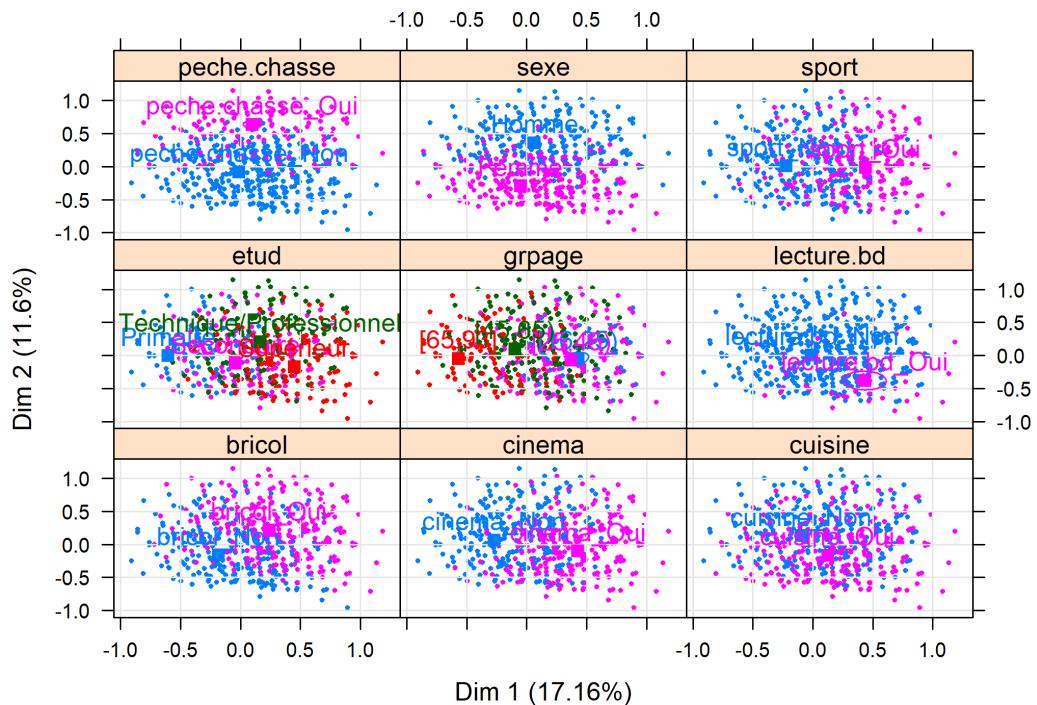


Figure 21. Ellipses de confiance (means=TRUE) dans le plan factoriel

L'option `means=FALSE` calculera les ellipses de confiance pour l'ensemble des coordonnées des observations relevant de chaque catégorie.

```
R> plotellipses(acm2, means = FALSE)
```

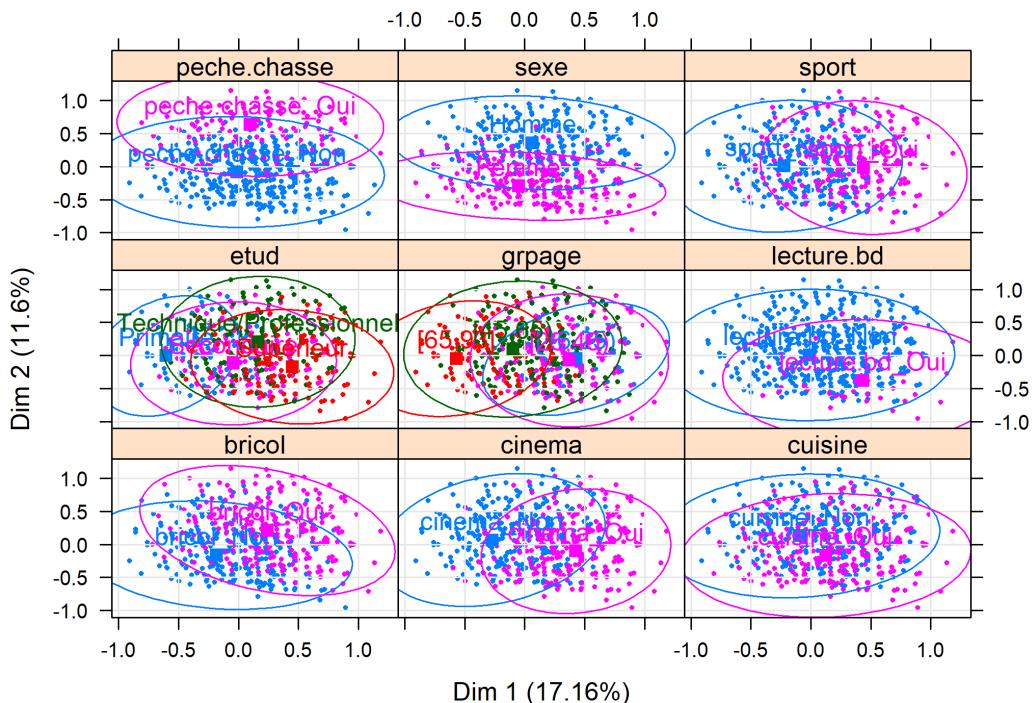


Figure 22. Ellipses de confiance (means=FALSE) dans le plan factoriel

La fonction `dimdesc` aide à décrire et interpréter les dimensions de l'ACM. Cette fonction est très utile quand le nombre de variables est élevé. Elle permet de voir à quelles variables les axes sont le plus liés : quelles variables et quelles modalités décrivent le mieux chaque axe ?

Pour les variables qualitatives, un modèle d'analyse de variance à un facteur est réalisé pour chaque dimension ; les variables à expliquer sont les coordonnées des individus et la variable explicative est une des variables qualitatives. Un test F permet de voir si la variable a un effet significatif sur la dimension et des tests T sont réalisés modalité par modalité (avec le contraste somme des  $\alpha_j=0$ ). Cela montre si les coordonnées des individus de la sous-population définie par une modalité sont significativement différentes de celles de l'ensemble de la population (i.e. différentes de 0). Les variables et modalités sont triées par probabilité critique et seules celles qui sont significatives sont gardées dans le résultat.

- Source : <http://factominer.free.fr/factosbest/description-des-dimensions.html>

```
R> dd <- dimdesc(acm2, axes = 1:2)
dd$`Dim 1`
```

## Extensions complémentaires

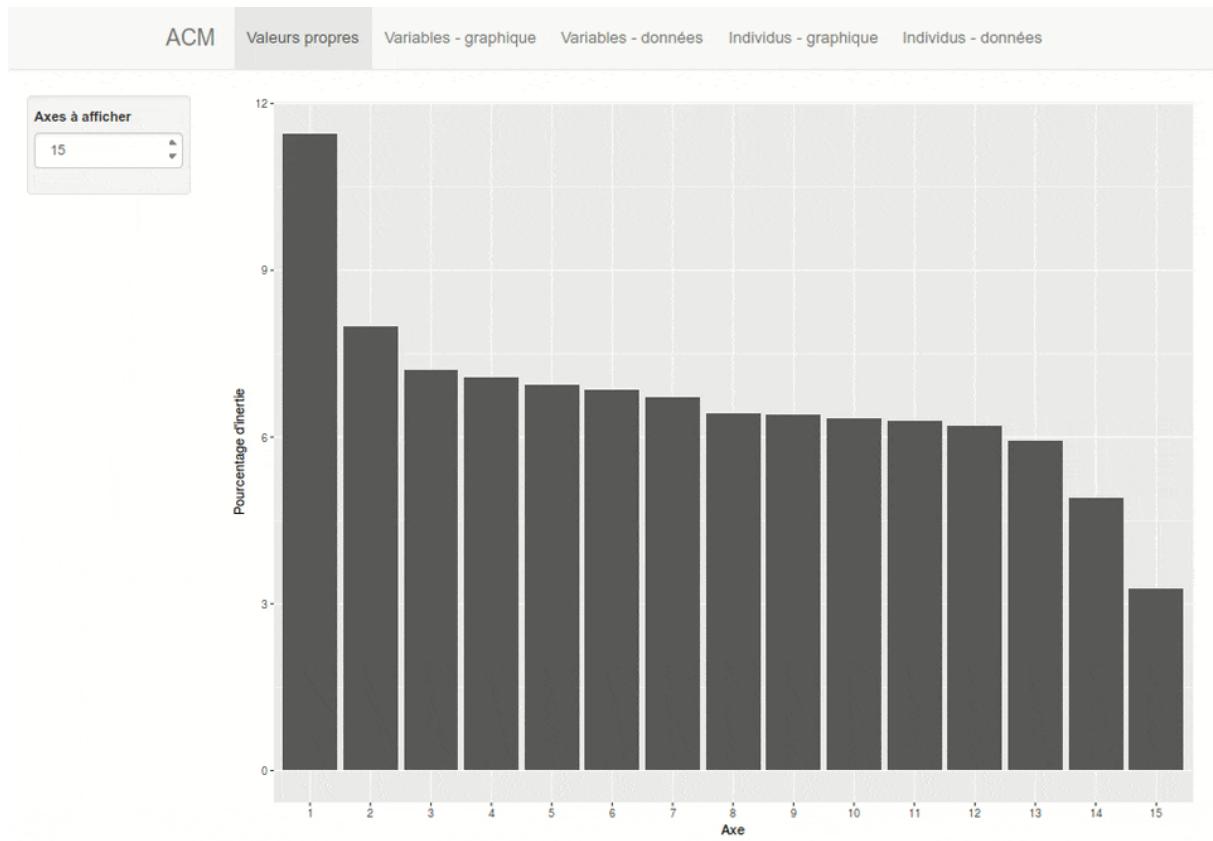
### factoextra

L'extension **factoextra** fournit des fonctions graphiques **ggplot2** pour visualiser les résultats d'une analyse factorielle, réalisée avec **ade4** ou **FactoMineR**.

Plus d'informations sur <https://rpkgs.datanovia.com/factoextra/>.

### explor

L'extension **explor** fournit une interface graphique pour explorer les résultats d'une analyse factorielle, réalisée avec **ade4** ou **FactoMineR**.



Interface graphique de explor

Plus d'informations sur <https://github.com/juba/explor/>.

## Factoshiny

L'extension **Factoshiny** permet d'améliorer facilement et de façon interactive les graphiques produits par **FactoMineR** pour les rendre beaucoup plus lisibles.

Plus d'informations sur <http://factominer.free.fr/graphs/factoshiny-fr.html>.

## FactoInvestigate

L'extension **FactoInvestigate** décrit et interprète automatiquement les résultats de votre analyse factorielle (ACP, AFC ou ACM) en choisissant les graphes les plus appropriés pour un rapport.

Vous avez juste à faire l'analyse comme habituellement avec **FactoMineR** ou **Factoshiny**, et ensuite utiliser **FactoInvestigate** pour obtenir un rapport automatisé.

Plus d'informations sur [http://factominer.free.fr/reporting/index\\_fr.html](http://factominer.free.fr/reporting/index_fr.html).

## Voir aussi

Un tutoriel détaillé en français, «Visualiser une analyse géométrique des données avec ggplot2 (R/RStudio)», est disponible sur le blog Quanti : <https://quanti.hypotheses.org/1871>.



# Classification ascendante hiérarchique (CAH)

Calculer une matrice des distances .....	596
Distance de Gower .....	596
Distance du $F^2$ .....	0
Exemple .....	598
Calcul du dendrogramme .....	599
Découper le dendrogramme .....	602
CAH avec l'extension FactoMineR .....	610

Il existe de nombreuses techniques statistiques visant à partitionner une population en différentes classes ou sous-groupes. La classification ascendante hiérarchique (CAH) est l'une d'entre elles. On cherche à ce que les individus regroupés au sein d'une même classe (homogénéité intra-classe) soient le plus semblables possibles tandis que les classes soient le plus dissemblables (hétérogénéité inter-classe).

Le principe de la CAH est de rassembler des individus selon un critère de ressemblance défini au préalable qui s'exprimera sous la forme d'une matrice de distances, exprimant la distance existant entre chaque individu pris deux à deux. Deux observations identiques auront une distance nulle. Plus les deux observations seront dissemblables, plus la distance sera importante. La CAH va ensuite rassembler les individus de manière itérative afin de produire un dendrogramme ou arbre de classification. La classification est *ascendante* car elle part des observations individuelles ; elle est *hiérarchique* car elle produit des classes ou groupes de plus en plus vastes, incluant des sous-groupes en leur sein. En découplant cet arbre à une certaine hauteur choisie, on produira la partition désirée.

## NOTE

On trouvera également de nombreux supports de cours en français sur la CAH sur le site de François Gilles Carpentier : <http://pagesperso.univ-brest.fr/~carpenti/>.

# Calculer une matrice des distances

La notion de ressemblance entre observations est évaluée par une distance entre individus. Plusieurs types de distances existent selon les données utilisées.

Il existe de nombreuses distances mathématiques pour les variables quantitatives (euclidiennes, Manhattan...) que nous n'aborderons pas ici<sup>1</sup>. La plupart peuvent être calculées avec la fonction `dist`.

Usuellement, pour un ensemble de variables qualitatives, on aura recours à la distance du F<sup>2</sup> qui est celle utilisée pour l'analyse des correspondances multiples (voir le [chapitre dédié](#)). Avec l'extension `ade4`, la distance du F<sup>2</sup> s'obtient avec la fonction `dist.dudi`<sup>2</sup>. Le cas particulier de la CAH avec l'extension `FactoMineR` sera abordée dans une section spécifique ci-après. Nous évoquerons également la distance de Gower qui peut s'appliquer à un ensemble de variables à la fois qualitatives et quantitatives et qui se calcule avec la fonction `daisy` de l'extension `cluster`. Enfin, dans le chapitre sur l'analyse de séquences, page 679, nous verrons également la fonction `seqdist` (extension `TraMineR`) permettant de calculer une distance entre séquences.

## Distance de Gower

En 1971, Gower a proposé un indice de similarité qui porte son nom<sup>3</sup>. L'objectif de cet indice consiste à mesurer dans quelle mesure deux individus sont semblables. L'indice de Gower varie entre 0 et 1. Si l'indice vaut 1, les deux individus sont identiques. À l'opposé, s'il vaut 0, les deux individus considérés n'ont pas de point commun. Si l'on note  $S_g$  l'indice de similarité de Gower, la distance de Gower  $D_g$  s'obtient simplement de la manière suivante :  $D_g = 1 - S_g$ . Ainsi, la distance sera nulle entre deux individus identiques et elle sera égale à 1 entre deux individus totalement différents. Cette distance s'obtient sous R avec la fonction `daisy` du package `cluster`.

L'indice de similarité de Gower entre deux individus  $x_1$  et  $x_2$  se calcule de la manière suivante :

$$S_g(x_1, x_2) = \frac{1}{p} \sum_{j=1}^p \frac{|x_{1j} - x_{2j}|}{\max(x_{1j}, x_{2j})}$$

$p$  représente le nombre total de caractères (ou de variables) descriptifs utilisés pour comparer les deux

- Pour une présentation de ces différentes distances, on pourra se référer à [http://old.biodiversite.wallonie.be/outils/methodo/similarity\\_distance.htm](http://old.biodiversite.wallonie.be/outils/methodo/similarity_distance.htm) ou encore à ce support de cours par D. Chessel, J. Thioulouse et A.B. Dufour disponible à <http://pbil.univ-lyon1.fr/R/pdf/stage7.pdf>.
- Cette même fonction peut aussi être utilisée pour calculer une distance après une analyse en composantes principales ou une analyse mixte de Hill et Smith.
- Voir Gower, J. (1971). A General Coefficient of Similarity and Some of Its Properties. *Biometrics*, 27(4), 857-871. doi:10.2307/2528823 (<http://www.jstor.org/stable/2528823>).

individus<sup>4</sup>.  $s_{12j}$  représente la similarité partielle entre les individus 1 et 2 concernant le descripteur  $j$ . Cette similarité partielle se calcule différemment s'il s'agit d'une variable qualitative ou quantitative :

- **variable qualitative** :  $s_{12j}$  vaut 1 si la variable  $j$  prend la même valeur pour les individus 1 et 2, et vaut 0 sinon. Par exemple, si 1 et 2 sont tous les deux « grand », alors  $s_{12j}$  vaudra 1. Si 1 est « grand » et 2 « petit »,  $s_{12j}$  vaudra 0.
- **variable quantitative** : la différence absolue entre les valeurs des deux variables est tout d'abord calculée, soit  $|y_{1j} - y_{2j}|$ . Puis l'écart maximum observé sur l'ensemble du fichier est déterminé et noté  $R_j$ . Dès lors, la similarité partielle vaut  $S_{12j} = 1 - |y_{1j} - y_{2j}| / R_j$ .

Dans le cas où l'on n'a que des variables qualitatives, la valeur de l'indice de Gower correspond à la proportion de caractères en commun. Supposons des individus 1 et 2 décrits ainsi :

1. homme / grand / blond / étudiant / urbain
2. femme / grande / brune / étudiante / rurale

Sur les 5 variables utilisées pour les décrire, 1 et 2 ont deux caractéristiques communes : ils sont grand(e)s et étudiant(e)s. Dès lors, l'indice de similarité de Gower entre 1 et 2 vaut  $2/5 = 0,4$  (soit une distance de  $1 - 0,4 = 0,6$ ).

Plusieurs approches peuvent être retenues pour traiter les valeurs manquantes :

- supprimer tout individu n'étant pas renseigné pour toutes les variables de l'analyse ;
- considérer les valeurs manquantes comme une modalité en tant que telle ;
- garder les valeurs manquantes en tant que valeurs manquantes.

Le choix retenu modifiera les distances de Gower calculées. Supposons que l'on ait :

1. homme / grand / blond / étudiant / urbain
2. femme / grande / brune / étudiante / manquant

Si l'on supprime les individus ayant des valeurs manquantes, 2 est retirée du fichier d'observations et aucune distance n'est calculée.

Si l'on traite les valeurs manquantes comme une modalité particulière, 1 et 2 partagent alors 2 caractères sur les 5 analysés, la distance de Gower entre eux est alors de  $1 - 2/5 = 1 - 0,4 = 0,6$ .

Si on garde les valeurs manquantes, l'indice de Gower est dès lors calculé sur les seuls descripteurs renseignés à la fois pour 1 et 2. La distance de Gower sera calculée dans le cas présent uniquement sur les 4 caractères renseignés et vaudra  $1 - 2/4 = 0,5$ .

## Distance du F<sup>2</sup>

Il s'agit de la distance utilisée dans les analyses de correspondance multiples (ACM). C'est une variante

4. Pour une description mathématique plus détaillée de cette fonction, notamment en cas de valeur manquante, se référer à l'article original de Gower précédemment cité.

de la distance du  $F^2$ . Nous considérons ici que nous avons  $Q$  questions (soit  $Q$  variables initiales de type facteur). À chaque individu est associé un patron c'est-à-dire une certaine combinaison de réponses aux  $Q$  questions. La distance entre deux individus correspond à la distance entre leurs deux patrons. Si les deux individus présentent le même patron, leur distance sera nulle. La distance du  $F^2$  peut s'exprimer ainsi :

$$d_{F^2}^2(L_i, L_j) = \frac{1}{Q} \sum_k \frac{(d_{ik} - d_{jk})^2}{f_k}$$

où  $L_i$  et  $L_j$  sont deux patrons,  $Q$  le nombre total de questions.  $d_{ik}$  vaut 1 si la modalité  $k$  est présente dans le patron  $L_i$ , 0 sinon.  $f_k$  est la fréquence de la modalité  $k$  dans l'ensemble de la population.

Exprimé plus simplement, on fait la somme de l'inverse des modalités non communes aux deux patrons, puis on divise par le nombre total de question. Si nous reprenons notre exemple précédent :

1. homme / grand / blond / étudiant / urbain
2. femme / grande / brune / étudiante / rurale

Pour calculer la distance entre 1 et 2, il nous faut connaître la proportion des différentes modalités dans l'ensemble de la population étudiée. En l'occurrence :

- hommes : 52 % / femmes : 48 %
- grand : 30 % / moyen : 45 % / petit : 25 %
- blond : 15 % / châtain : 45 % / brun : 30 % / blanc : 10 %
- étudiant : 20 % / salariés : 65 % / retraités : 15 %
- urbain : 80 % / rural : 20 %

Les modalités non communes entre les profils de 1 et 2 sont : homme, femme, blond, brun, urbain et rural. La distance du  $F^2$  entre 1 et 2 est donc la suivante :

$$d_{F^2}^2(L_1, L_2) = \frac{1}{5} \left( \frac{1}{0,52} + \frac{1}{0,48} + \frac{1}{0,15} + \frac{1}{0,30} + \frac{1}{0,80} + \frac{1}{0,20} \right) = 4,05$$

Cette distance, bien que moins intuitive que la distance de Gower évoquée précédemment, est la plus employée pour l'analyse d'enquêtes en sciences sociales. Il faut retenir que la distance entre deux profils est dépendante de la distribution globale de chaque modalité dans la population étudiée. Ainsi, si l'on recalcule les distances entre individus à partir d'un sous-échantillon, le résultat obtenu sera différent. De manière générale, les individus présentant des caractéristiques rares dans la population vont se retrouver éloignés des individus présentant des caractéristiques fortement représentées.

## Exemple

Nous allons reprendre l'ACM calculée avec `dudi.acm` (`ade4`) dans le [chapitre consacré à l'ACM](#) :

```
R> library(questionr)
data(hdv2003)
d <- hdv2003
d$grpage <- cut(d$age, c(16, 25, 45, 65, 93), right = FALSE, include.lowest =
TRUE)
d$etud <- d$nivetud
levels(d$etud) <- c(
  "Primaire", "Primaire", "Primaire", "Secondaire", "Secondaire",
  "Technique/Professionnel", "Technique/Professionnel", "Supérieur"
)
d2 <- d[, c("grpae", "sexe", "etud", "peche.chasse", "cinema", "cuisine", "bricol",
  "sport", "lecture.bd")]
library(ade4)
acm <- dudi.acm(d2, scannf = FALSE, nf = 5)
```

La matrice des distances s'obtient dès lors avec la fonction `dist.dudi` :

```
R> md <- dist.dudi(acm)
```

## Calcul du dendrogramme

Il faut ensuite choisir une méthode d'agrégation pour construire le dendrogramme. De nombreuses solutions existent (saut minimum, distance maximum, moyenne, Ward...). Chacune d'elle produira un dendrogramme différent. Nous ne détaillerons pas ici ces différentes techniques<sup>5</sup>. Cependant, à l'usage, on privilégierra le plus souvent la méthode de Ward<sup>6</sup>. De manière simplifiée, cette méthode cherche à minimiser l'inertie intra-classe et à maximiser l'inertie inter-classe afin d'obtenir des classes les plus homogènes possibles. Cette méthode est souvent incorrectement présentée comme une «méthode de minimisation de la variance» alors qu'au sens strict Ward vise «l'augmentation mininum de la somme des carrés» ("minimum increase of sum-of-squares (of errors)")<sup>7</sup>.

En raison de la variété des distances possibles et de la variété des techniques d'agrégation, on pourra être amené à réaliser plusieurs dendrogrammes différents sur un même jeu de données jusqu'à obtenir une classification qui fait « sens ».

La fonction de base pour le calcul d'un dendrogramme est `hclust` en précisant le critère d'agrégation avec `method`. Dans notre cas, nous allons opter pour la méthode de Ward appliquée au carré des

5. On pourra consulter le cours de FG Carpentier déjà cité ou bien des ouvrages d'analyse statistique.

6. Ward, J. (1963). Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301), 236-244. doi:10.2307/2282967. (<http://www.jstor.org/stable/2282967>)

7. Voir par exemple la discussion, en anglais, sur Wikipedia concernant la page présentant la méthode Ward : [https://en.wikipedia.org/wiki/Talk:Ward%27s\\_method](https://en.wikipedia.org/wiki/Talk:Ward%27s_method)

distances (ce qu’on indique avec `method = "ward.D2"`<sup>8</sup>):

```
R> arbre <- hclust(md, method = "ward.D2")
```

#### NOTE

Le temps de calcul d’un dendrogramme peut être particulièrement important sur un gros fichier de données. L’extension `fastcluster` permet de réduire significativement le temps de calcul. Il suffit d’installer puis d’appeler cette extension. La fonction `hclust` sera automatiquement remplacée par cette version optimisée. Elle prends les mêmes paramètres :

```
R> library(fastcluster)
arbre <- hclust(md, method = "ward.D2")
```

Le dendrogramme obtenu peut être affiché simplement avec `plot`. Lorsque le nombre d’individus est important, il peut être utile de ne pas afficher les étiquettes des individus avec `labels=FALSE`.

---

8. Depuis la version 3.1 de R. L’option `method = "ward.D"` correspondant à la version disponible dans les versions précédentes de R. Mais il est à noter que la méthode décrite par Ward dans son article de 1963 correspond en réalité à `method = "ward.D2"`.

```
R> plot(arbre, labels = FALSE, main = "Dendrogramme")
```

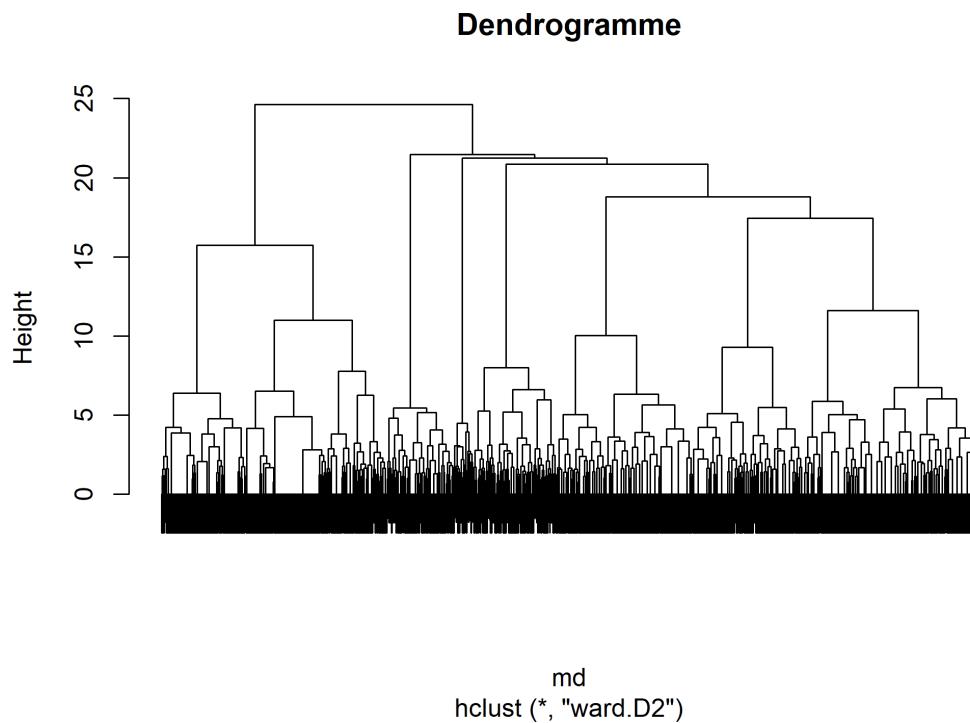


Figure 1. Dendrogramme obtenu avec `hclust`

La fonction `agnes` de l'extension `cluster` peut également être utilisée pour calculer le dendrogramme. Cependant, à l'usage, elle semble être un peu plus lente que `hclust`.

```
R> library(cluster)
arbre2 <- agnes(md, method = "ward")
```

ATTENTION: la méthode implémentée dans la fonction `agnes` correspond à l'option `method = "ward.D2"` de `hclust`.

Le résultat obtenu n'est pas au même format que celui de `hclust`. Il est possible de transformer un objet `agnes` au format `hclust` avec `as.hclust`.

```
R> as.hclust(arbre2)
```

## Découper le dendrogramme

Pour obtenir une partition de la population, il suffit de découper le dendrogramme obtenu à une certaine hauteur. En premier lieu, une analyse de la forme du dendrogramme pourra nous donner une indication sur le nombre de classes à retenir. Dans notre exemple, deux branches bien distinctes apparaissent sur l’arbre.

Pour nous aider, nous pouvons représenter les sauts d’inertie du dendrogramme selon le nombre de classes retenues.

```
R> inertie <- sort(arbre$height, decreasing = TRUE)
plot(inertie[1:20], type = "s", xlab = "Nombre de classes", ylab = "Inertie")
```

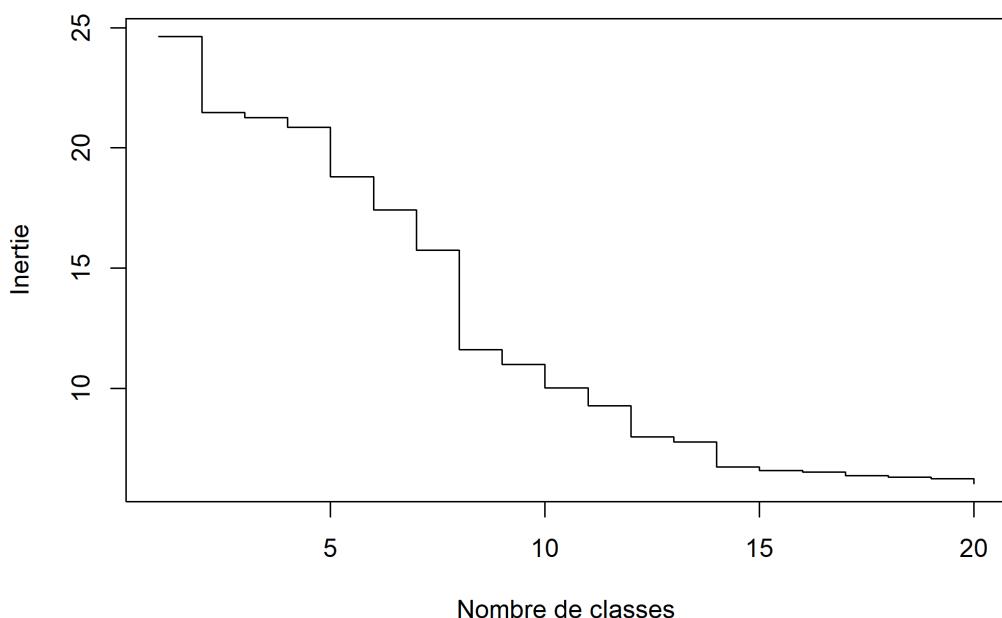


Figure 2. Inertie du dendrogramme

On voit trois sauts assez nets à 2, 5 et 8 classes, que nous avons représentés ci-dessous respectivement en vert, en rouge et en bleu.

```
R> plot(inertie[1:20], type = "s", xlab = "Nombre de classes", ylab = "Inertie")
  points(c(2, 5, 8), inertie[c(2, 5, 8)], col = c("green3", "red3", "blue3"), cex = 2, lwd = 3)
```

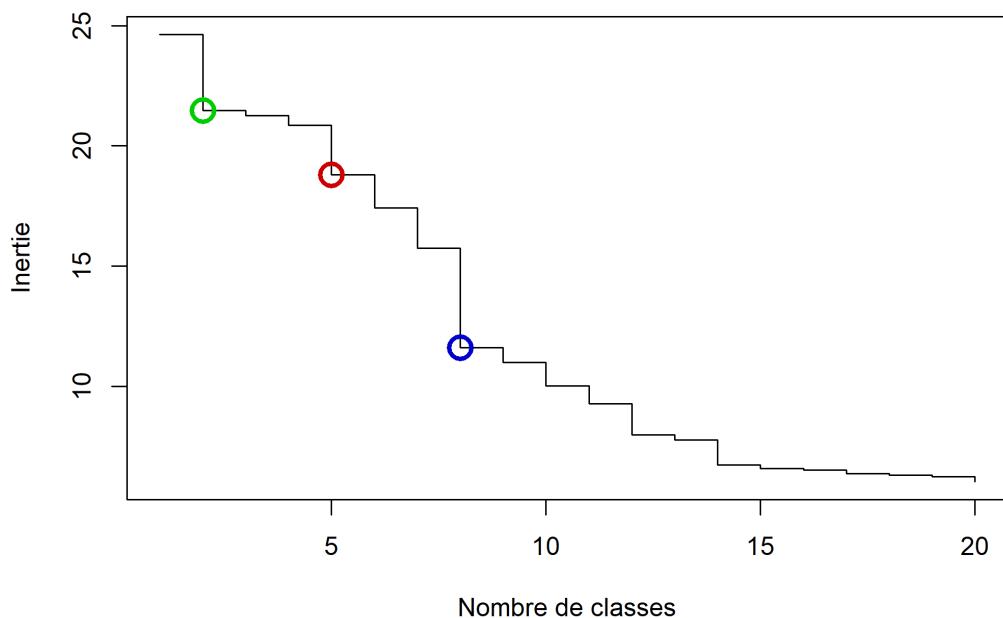


Figure 3. Sauts d'inertie du dendrogramme

La fonction `rect.hclust` permet de visualiser les différentes partitions directement sur le dendrogramme.

```
R> plot(arbre, labels = FALSE, main = "Partition en 2, 5 ou 8 classes", xla  
b = "", ylab = "", sub = "", axes = FALSE, hang = -1)  
rect.hclust(arbre, 2, border = "green3")  
rect.hclust(arbre, 5, border = "red3")  
rect.hclust(arbre, 8, border = "blue3")
```

### Partition en 2, 5 ou 8 classes

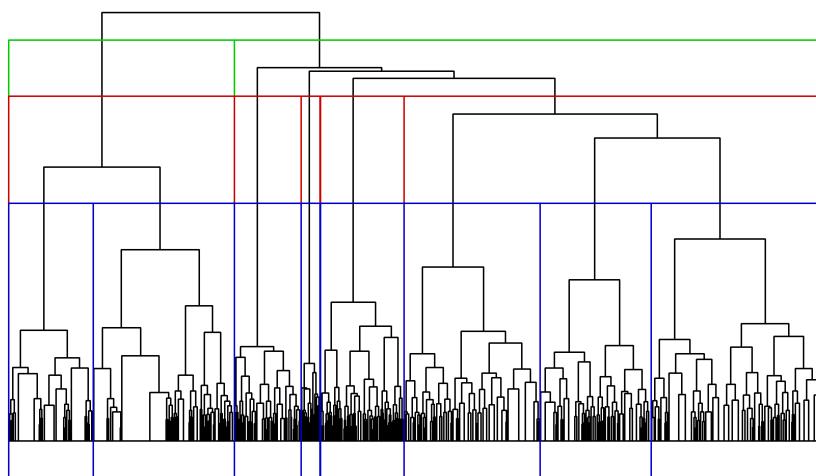


Figure 4. Différentes partitions du dendrogramme

L’extension **FactoMineR** (que nous aborderons dans une section dédiée ci-après, page 610) suggère d’utiliser la partition ayant la plus grande perte relative d’inertie.

L’extension **JLutils** (disponible sur [GitHub](#)) propose une fonction `best.cutree` qui permet de calculer cette indicateur à partir de n’importe quel dendrogramme calculé avec `hclust` ou `agnes`.

Pour installer **JLutils**, on aura recours au package `devtools` et à sa fonction `install_github` :

```
R> library(devtools)  
install_github("larmarange/JLutils")
```

Par défaut, `best.cutree` regarde quelle serait la meilleure partition entre 3 et 20 classes.

```
R> library(JLutils)
best.cutree(arbre)
```

```
[1] 5
```

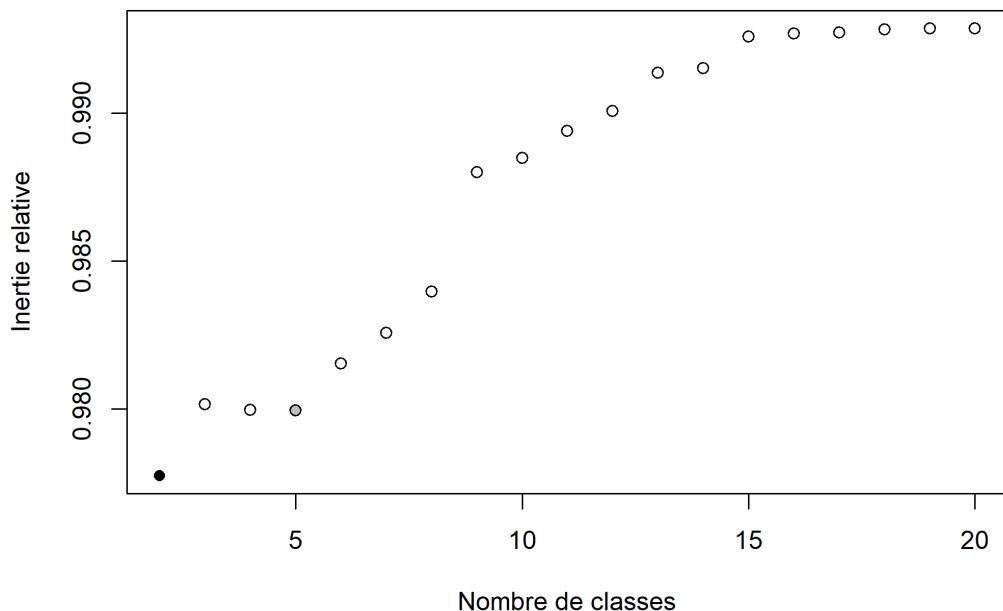
En l'occurrence il s'agirait d'une partition en 5 classes. Il est possible de modifier le minimum et le maximum des partitions recherchées avec `min` et `max`.

```
R> best.cutree(arbre, min = 2)
```

```
[1] 2
```

On peut également représenter le graphique des pertes relatives d'inertie avec `graph=TRUE`. La meilleure partition selon ce critère est représentée par un point noir et la seconde par un point gris.

```
R> best.cutree(arbre, min = 2, graph = TRUE, xlab = "Nombre de classes", ylab = "Inertie relative")
```



```
[1] 2
```

Figure 5. Perte relative d'inertie selon le nombre de classes

Un découpage en deux classes minimise ce critère. Cependant, si l'on souhaite réaliser une analyse un peu plus fine, un nombre de classes plus élevé serait pertinent. Nous allons donc retenir un découpage en cinq classes. Le découpage s'effectue avec la fonction `cutree`.

```
R> typo <- cutree(arbre, 5)
freq(typo)
```

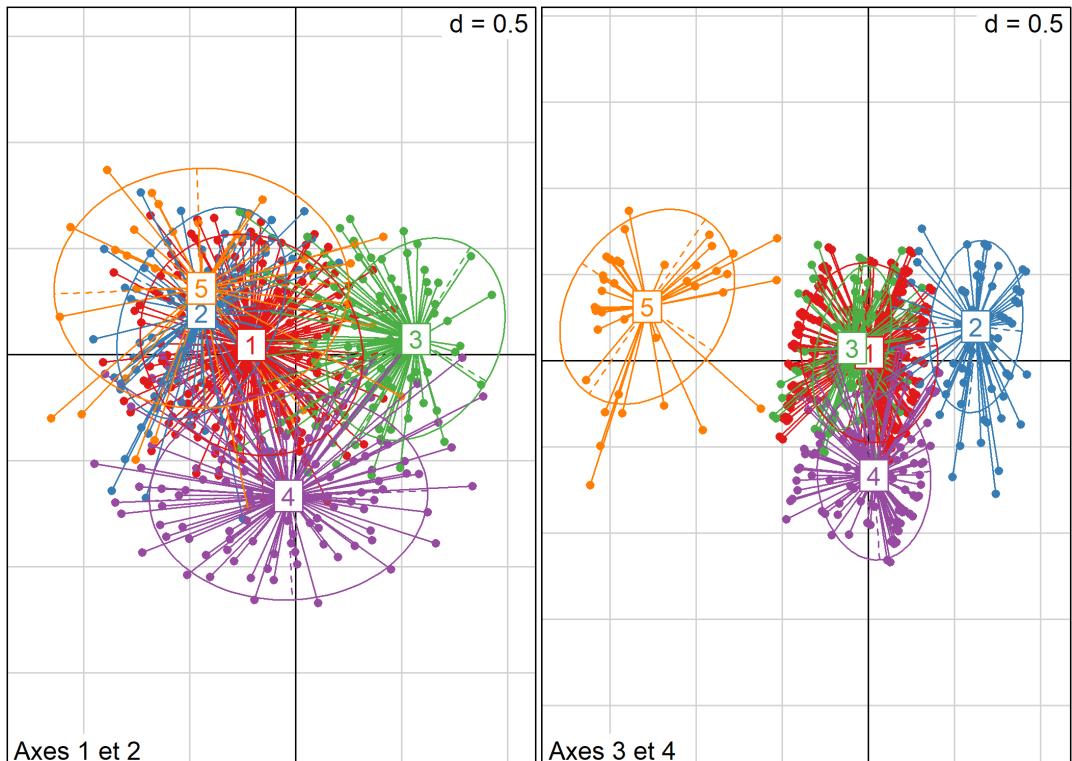
**NOTE**

Il existe de multiples autres indicateurs statistiques cherchant à mesurer la «qualité» de chaque partition. Pour cela, on pourra par exemple avoir recours à la fonction `as.clustrange` de l'extension **WeightedCluster**.

Pour plus d'informations, voir le *manuel de la librairie WeightedCluster*, chapitre 7.

La typologie obtenue peut être représentée dans le plan factoriel avec `s.class`.

```
R> par(mfrow = c(1, 2))
library(RColorBrewer)
s.class(acm$li, as.factor(typo), col = brewer.pal(5, "Set1"), sub = "Axes 1 et 2")
s.class(acm$li, as.factor(typo), 3, 4, col = brewer.pal(5, "Set1"), sub = "Axes 3 et 4")
```



```
R> par(mfrow = c(1, 1))
```

Figure 6. Projection de la typologie obtenue par CAH selon les 4 premiers axes

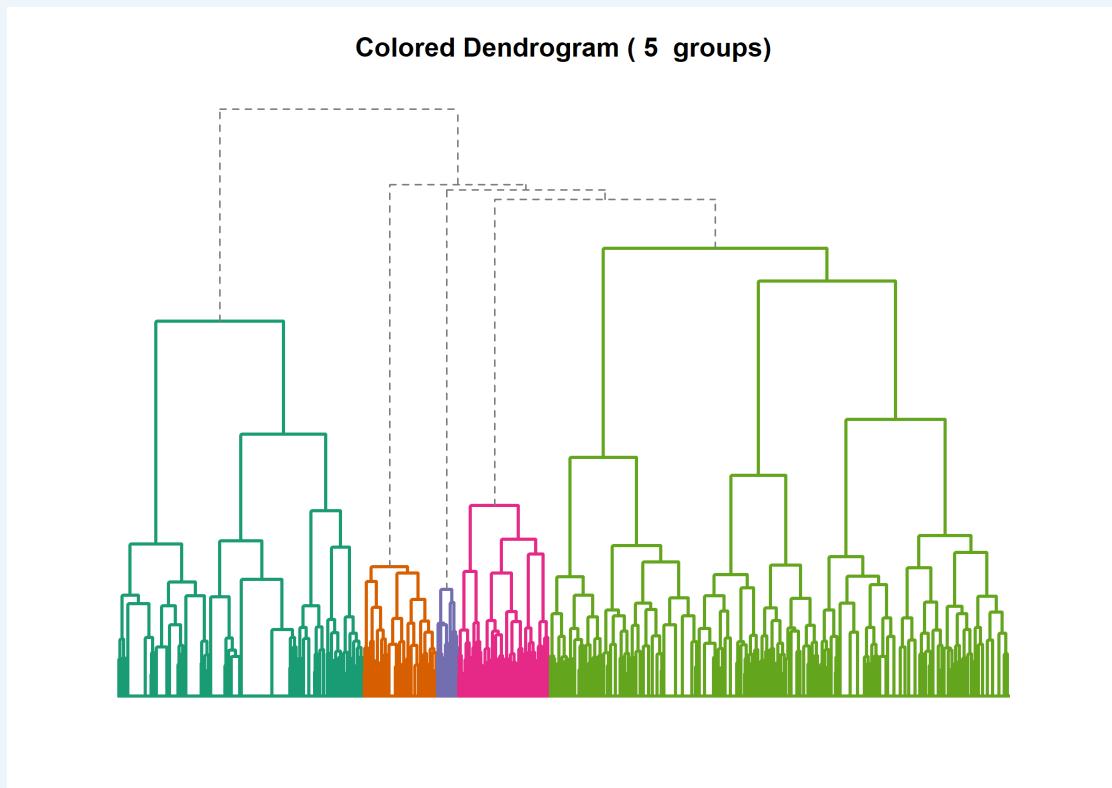
**NOTE**

De nombreuses possibilités graphiques sont possibles avec les dendrogrammes. Des exemples documentés sont disponibles à cette adresse : <http://rpubs.com/gaston/dendograms>.

Romain François a développé une fonction `A2Rplot` permettant de réaliser facilement un dendrogramme avec les branches colorées<sup>9</sup>. Par commodité, cette fonction est disponible directement au sein de l'extension `JLutils`.

Pour réaliser le graphique, on indiquera le nombre de classes et les couleurs à utiliser pour chaque branche de l'arbre :

```
R> A2Rplot(arbre, k = 5, boxes = FALSE, col.up = "gray50", col.down = brewer.pal(5, "Dark2"), show.labels = FALSE)
```



On pourra aussi noter l'extension `ggdendro` pour représenter des dendrogrammes avec `ggplot2` ou encore l'extension `dendextend` qui permet de manipuler, représenter et comparer des dendrogrammes<sup>10</sup>.

9. Voir <http://addicted2r.free.fr/packages/A2R/lastVersion/R/code.R>.

## CAH avec l’extension FactoMineR

L’extension **FactoMineR** fournit une fonction `HCPC` permettant de réaliser une classification hiérarchique à partir du résultats d’une analyse factorielle réalisée avec la même extension (voir la section dédiée du chapitre sur l’ACM, page 610).

`HCPC` réalise à la fois le calcul de la matrice des distances, du dendrogramme et le partitionnement de la population en classes. Par défaut, `HCPC` calcule le dendrogramme à partir du carré des distances du  $F^2$  et avec la méthode de Ward.

Par défaut, l’arbre est affiché à l’écran et l’arbre sera coupé selon la partition ayant la plus grande perte relative d’inertie (comme avec `best.cutree`). Utilisez `graph=FALSE` pour ne pas afficher le graphique et l’argument `nb.clust` pour indiquer le nombre de classes désirées.

```
R> library(FactoMineR)
acm2 <- MCA(d2[complete.cases(d2), ], ncp = 5, graph = FALSE)
cah <- HCPC(acm2, graph = FALSE)
```

On pourra représenter le dendrogramme avec `plot` et l’argument `choice="tree"`.

---

10. Pour une présentation succincte, voir l’introduction de l’extension sur <https://cran.r-project.org/web/packages/dendextend/vignettes/introduction.html> (en anglais).

```
R> plot(cah, choice = "tree")
```

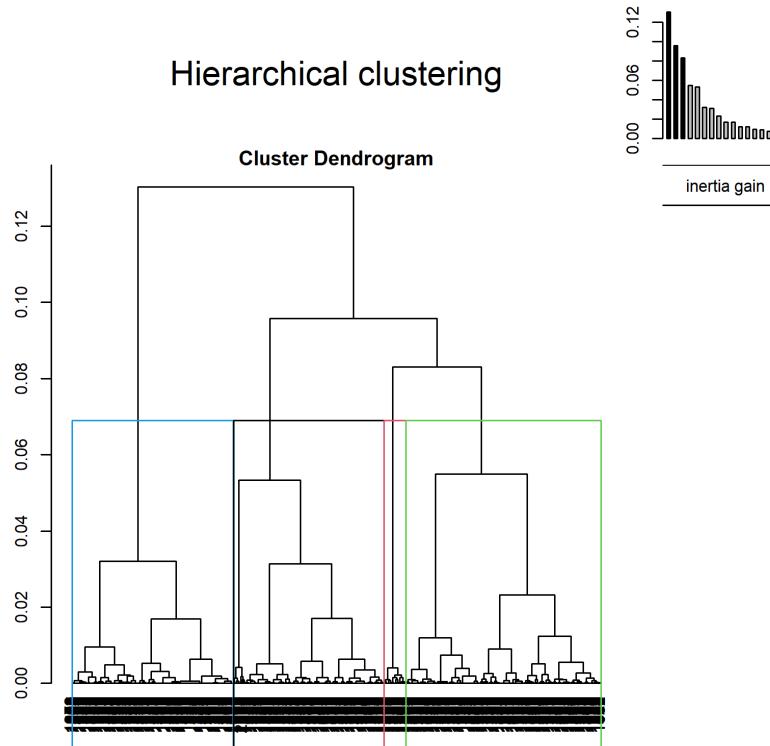


Figure 7. Dendrogramme obtenu avec HCPC (5 axes)

Il apparaît que le dendrogramme obtenu avec HCPC diffère de celui que nous avons calculé précédemment en utilisant la matrice des distances fournies par `dist.dudi`. Cela est dû au fait que HCPC procède différemment pour calculer la matrice des distances en ne prenant en compte que les axes retenus dans le cadre de l'ACM. Pour rappel, nous avions retenu que 5 axes dans le cadre de notre ACM :

```
R> acm2 <- MCA(d2[complete.cases(d2), ], ncp = 5, graph = FALSE)
```

HCPC n'a donc pris en compte que ces 5 premiers axes pour calculer les distances entre les individus, considérant que les autres axes n'apportent que du « bruit » rendant la classification instable. Cependant, comme le montre `summary(acm2)`, nos cinq premiers axes n'expliquent que 54 % de la variance. Il est généralement préférable de garder un plus grande nombre d'axes afin de couvrir au moins 80 à 90 % de la variance<sup>11</sup>. De son côté, `dist.dudi` prends en compte l'ensemble des axes pour calculer la matrice des distances. On peut reproduire cela avec FactoMineR en indiquant `ncp=Inf` lors du calcul de l'ACM.

11. Voir <http://factominer.free.fr/classical-methods/classification-hierarchique-sur-composantes-principales.html>

```
R> acm2 <- MCA(d2[complete.cases(d2), ], ncp = Inf, graph = FALSE)
      cah <- HCPC(acm2, nb.clust = -1, graph = FALSE)
```

On obtient bien cette fois-ci le même résultat.

```
R> plot(cah, choice = "tree")
```

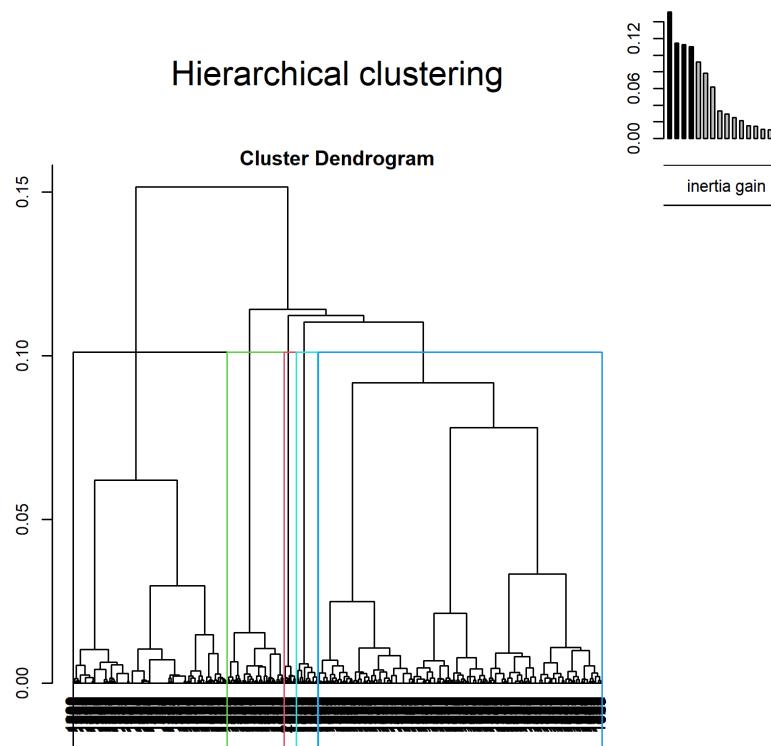


Figure 8. Dendrogramme obtenu avec HCPC (tous les axes)

D'autres graphiques sont disponibles, en faisant varier la valeur de l'argument `choice` :

```
R> plot(cah, choice = "3D.map")
```

### Hierarchical clustering on the factor map

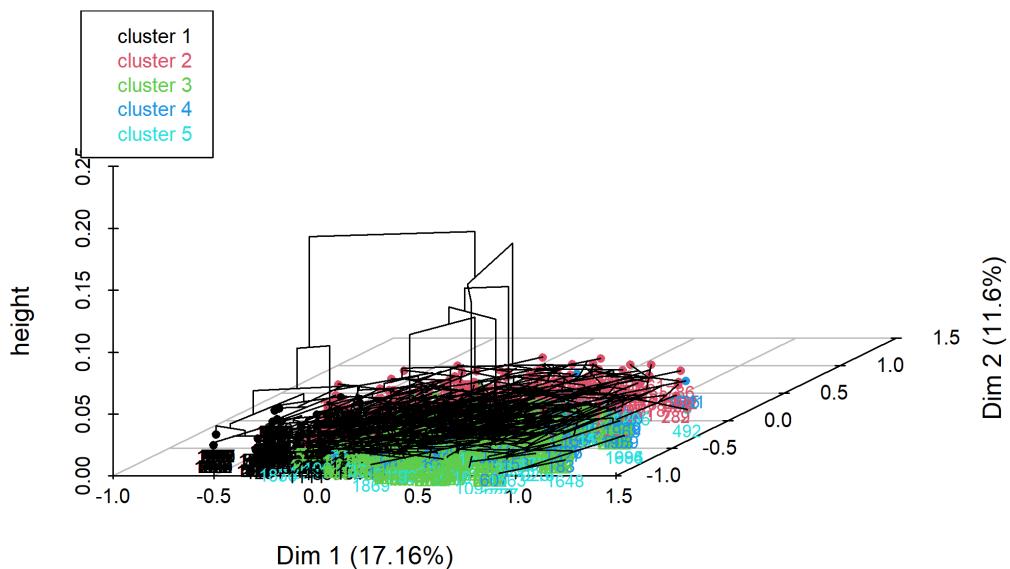


Figure 9. Représentation en 3 dimensions du dendrogramme

```
R> plot(cah, choice = "bar")
```

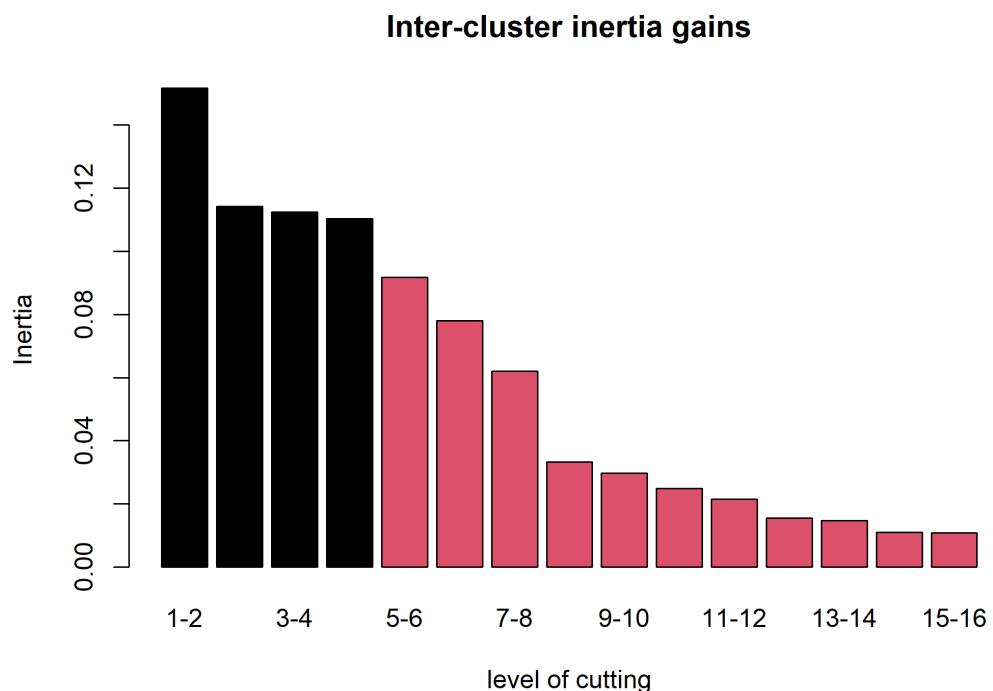
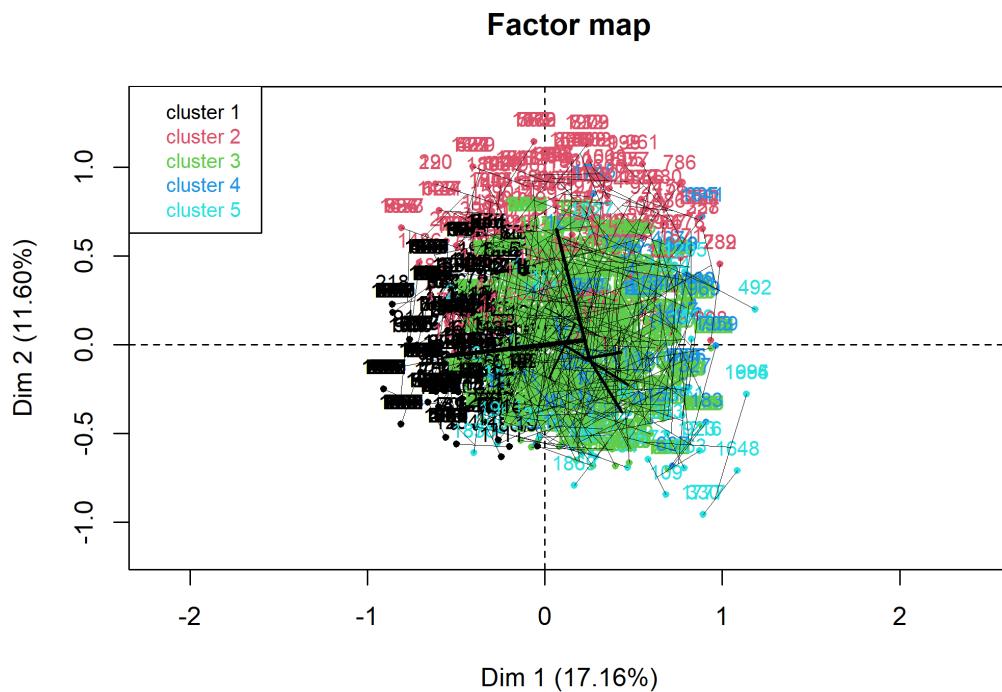


Figure 10. Gains d'inertie

```
R> plot(cah, choice = "map")
```



*Figure 11. Projection des catégories sur le plan factoriel*

L'objet renvoyé par `HCPC` contient de nombreuses informations. La partition peut notamment être récupérée avec `cah$data.clust$clust`. Il y a également diverses statistiques pour décrire les catégories.

```
R> cah
```

```
**Results for the Hierarchical Clustering on Principal Components**
  name
1  "$data.clust"
2  "$desc.var"
3  "$desc.var$test.chi2"
4  "$desc.axes$category"
5  "$desc.axes"
6  "$desc.axes$quanti.var"
7  "$desc.axes$quanti"
8  "$desc.ind"
```

```
9  "$desc.ind$para"
10 "$desc.ind$dist"
11 "$call"
12 "$call$t"
  description
1 "dataset with the cluster of the individuals"
2 "description of the clusters by the variables"
3 "description of the cluster var. by the categorical var."
4 "description of the clusters by the categories."
5 "description of the clusters by the dimensions"
6 "description of the cluster var. by the axes"
7 "description of the clusters by the axes"
8 "description of the clusters by the individuals"
9 "parangons of each clusters"
10 "specific individuals"
11 "summary statistics"
12 "description of the tree"
```

```
R> freq(cah$data.clust$clust)
```

# Effets d'interaction dans un modèle

Exemple d'interaction .....	617
Un second exemple d'interaction .....	635
Explorer les différentes interactions possibles .....	641
Pour aller plus loin .....	644

Dans un modèle statistique classique, on fait l'hypothèse implicite que chaque variable explicative est indépendante des autres. Cependant, cela ne se vérifie pas toujours. Par exemple, l'effet de l'âge peut varier en fonction du sexe. Il est dès lors nécessaire de prendre en compte dans son modèle les effets d'interaction<sup>1</sup>.

## Exemple d'interaction

Reprendons le modèle que nous avons utilisé dans le chapitre sur la régression logistique, page 507.

---

1. Pour une présentation plus statistique et mathématique des effets d'interaction, on pourra se référer au [cours de Jean-François Bickel disponible en ligne](#).

```
R> library(questionr)
data(hdv2003)
d <- hdv2003
d$sexe <- relevel(d$sexe, "Femme")
d$grpage <- cut(d$age, c(16, 25, 45, 65, 99), right = FALSE, include.lowest =
TRUE)
d$etud <- d$nivetud
levels(d$etud) <- c(
  "Primaire", "Primaire", "Primaire",
  "Secondaire", "Secondaire", "Technique/Professionnel",
  "Technique/Professionnel", "Supérieur"
)
d$etud <- addNAstr(d$etud, "Manquant")
library(labelled)
var_label(d$sport) <- "Pratique du sport ?"
var_label(d$sexe) <- "Sexe"
var_label(d$grpage) <- "Groupe d'âges"
var_label(d$etud) <- "Niveau d'étude"
var_label(d$relig) <- "Pratique religieuse"
var_label(d$heures.tv) <- "Nombre d'heures passées devant la télévision par jo
ur"
```

Nous avions alors exploré les facteurs associés au fait de pratiquer du sport.

```
R> mod <- glm(sport ~ sexe + grpage + etud + heures.tv + relig, data = d, family = binomial())
library(gtsummary)
tbl_regression(mod, exponentiate = TRUE)
```

Caractéristique	OR <sup>1</sup>	95% CI <sup>1</sup>	p-value
Sexe			
Femme	—	—	
Homme	1,55	1,26 – 1,91	<0,001
Groupe d'âges			
[16,25)	—	—	
[25,45)	0,66	0,42 – 1,03	0,065
[45,65)	0,34	0,21 – 0,54	<0,001
[65,99]	0,25	0,15 – 0,43	<0,001
Niveau d'étude			
Primaire	—	—	
Secondaire	2,59	1,77 – 3,83	<0,001
Technique/Professionnel	2,86	1,98 – 4,17	<0,001
Supérieur	6,63	4,55 – 9,80	<0,001
Manquant	8,59	4,53 – 16,6	<0,001
Nombre d'heures passées devant la télévision par jour	0,89	0,83 – 0,95	<0,001
Pratique religieuse			
Pratiquant régulier	—	—	
Pratiquant occasionnel	0,98	0,68 – 1,42	>0,9
Appartenance sans pratique	0,99	0,71 – 1,40	>0,9
Ni croyance ni appartenance	0,81	0,55 – 1,18	0,3
Rejet	0,68	0,39 – 1,19	0,2
NSP ou NVPR	0,92	0,40 – 2,02	0,8

<sup>1</sup> OR = rapport de cotes, CI = intervalle de confiance

Selon les résultats de notre modèle, les hommes pratiquent plus un sport que les femmes et la pratique du sport diminue avec l'âge. Pour représenter les effets différentes variables, on peut avoir recours à la

fonction `allEffects` de l'extension `effects`.

```
R> library(effects)
```

```
Loading required package: carData
```

```
Registered S3 methods overwritten by 'lme4':
method from
cooks.distance.influence.merMod car
influence.merMod car
dfbeta.influence.merMod car
dfbetas.influence.merMod car
```

```
lattice theme set by effectsTheme()
See ?effectsTheme for details.
```

```
R> plot(allEffects(mod))
```

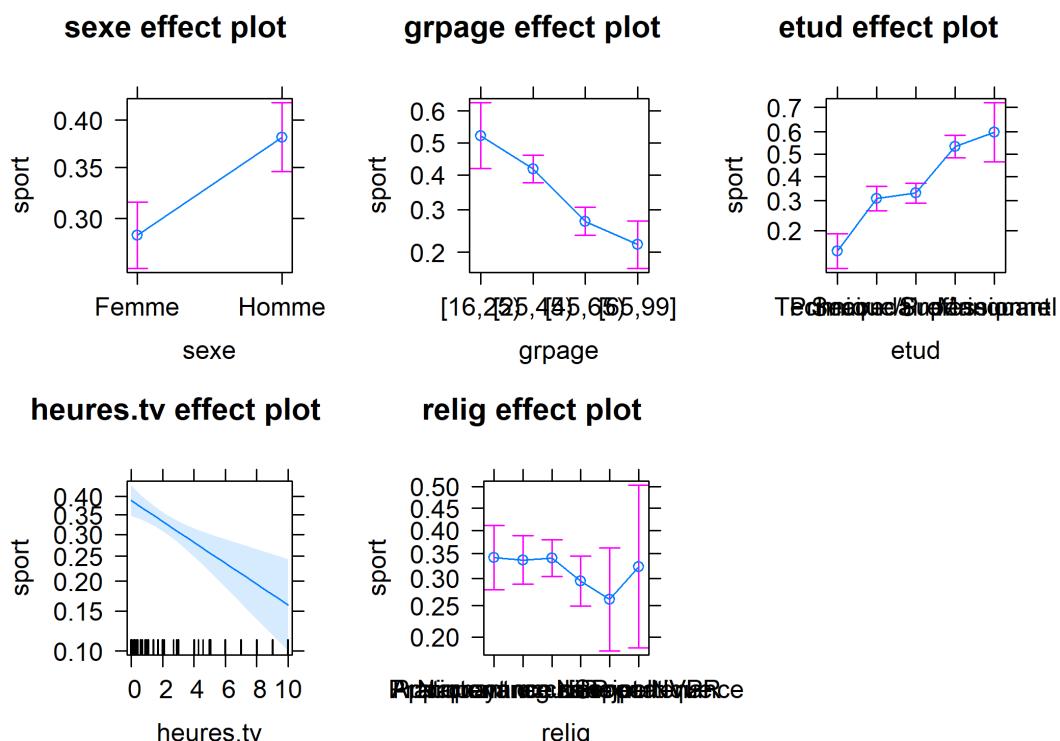


Figure 1. Représentation graphique des effets du modèle

Cependant, l'effet de l'âge est-il le même selon le sexe ? Nous allons donc introduire une interaction entre l'âge et le sexe dans notre modèle, ce qui sera représenté par `sexe * grpage` dans l'équation du modèle.

```
R> mod2 <- glm(sport ~ sexe * grpage + etud + heures.tv + relig, data = d, family  
y = binomial())  
tbl_regression(mod2, exponentiate = TRUE)
```

Caractéristique	OR <sup>1</sup>	95% CI <sup>1</sup>	p-value
Sexe			
Femme	—	—	
Homme	5,20	2,46 - 11,7	<0,001
Groupe d'âges			
[16,25)	—	—	
[25,45)	1,02	0,59 - 1,80	>0,9
[45,65)	0,61	0,34 - 1,10	0,10
[65,99]	0,46	0,23 - 0,92	0,027
Niveau d'étude			
Primaire	—	—	
Secondaire	2,60	1,77 - 3,85	<0,001
Technique/Professionnel	2,88	1,99 - 4,21	<0,001
Supérieur	6,77	4,64 - 10,0	<0,001
Manquant	9,02	4,68 - 17,8	<0,001
Nombre d'heures passées devant la télévision par jour	0,89	0,83 - 0,95	<0,001
Pratique religieuse			
Pratiquant régulier	—	—	
Pratiquant occasionnel	0,99	0,68 - 1,43	>0,9
Appartenance sans pratique	1,02	0,73 - 1,44	0,9
Ni croyance ni appartenance	0,83	0,57 - 1,22	0,3
Rejet	0,68	0,38 - 1,19	0,2
NSP ou NVPR	0,93	0,40 - 2,04	0,9
Sexe * Groupe d'âges			

<sup>1</sup> OR = rapport de cotes, CI = intervalle de confiance

Caractéristique	OR <sup>1</sup>	95% CI <sup>1</sup>	p-value
Homme * [25,45)	0,31	0,13 - 0,70	0,006
Homme * [45,65)	0,24	0,10 - 0,54	<0,001
Homme * [65,99]	0,23	0,09 - 0,59	0,003

<sup>1</sup> OR = rapport de cotes, CI = intervalle de confiance

Commençons par regarder les effets du modèle.

```
R> plot(allEffects(mod2))
```

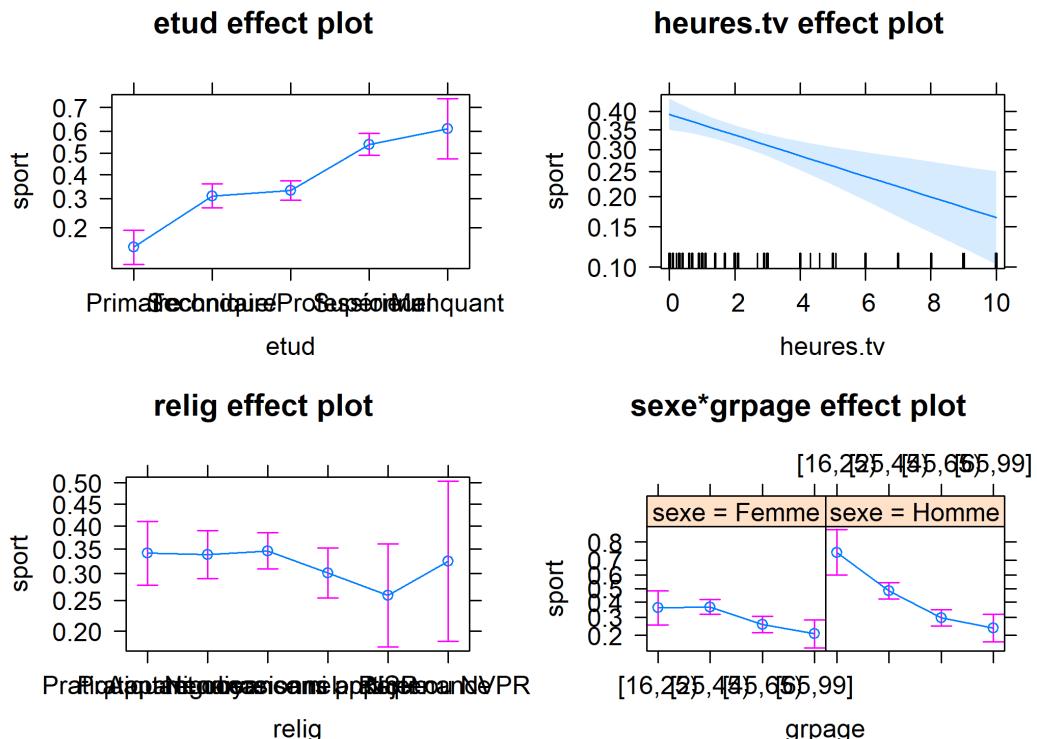


Figure 2. Représentation graphique des effets du modèle avec interaction entre le sexe et le groupe d'âge

Sur ce graphique, on voit que l'effet de l'âge sur la pratique d'un sport est surtout marqué chez les hommes. Chez les femmes, le même effet est observé, mais dans une moindre mesure et seulement à partir de 45 ans.

On peut tester si l'ajout de l'interaction améliore significativement le modèle avec [anova](#).

```
R> anova(mod2, test = "Chisq")
```

Jetons maintenant un oeil aux coefficients du modèle. Pour rendre les choses plus visuelles, nous aurons recours à `ggcoef_model` de l'extension **GGally**.

```
R> library(GGally)
ggcoef_model(mod2, exponentiate = TRUE)
```

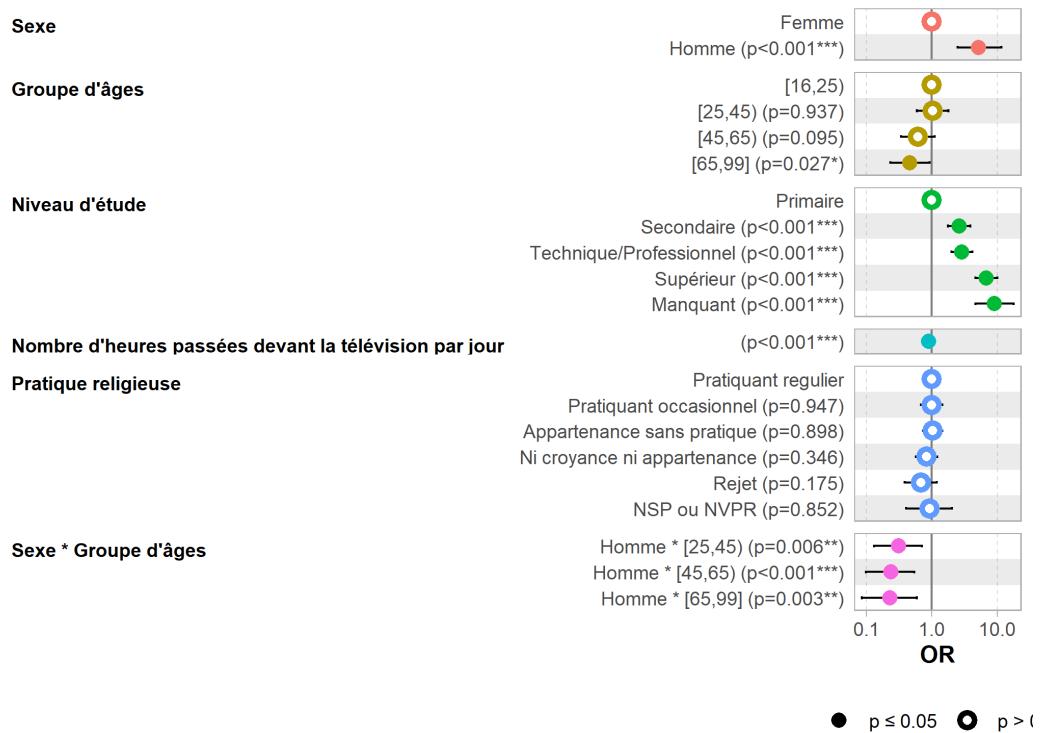


Figure 3. Représentation graphique des coefficients du modèle avec interaction entre le sexe et le groupe d'âge

Concernant l'âge et le sexe, nous avons trois séries de coefficients : trois coefficients (`grpage[25,45]`, `grpage[45,65]` et `grpage[65,99]`) qui correspondent à l'effet global de la variable `âge`, un coefficient (`sexHomme`) pour l'effet global du sexe et trois coefficients qui sont des modificateurs de l'effet d'âge pour les hommes (`grpage[25,45]`, `grpage[45,65]` et `grpage[65,99]`).

Pour bien interpréter ces coefficients, il faut toujours avoir en tête les modalités choisies comme référence pour chaque variable. Supposons une femme de 60 ans, dont toutes autres variables correspondent aux modalités de référence (c'est donc une pratiquante régulière, de niveau primaire, qui ne regarde pas la télévision). Regardons ce que prédit le modèle quant à sa probabilité de faire du sport au travers d'une représentation graphique

```
R> library(breakDown)
library(ggplot2)
logit <- function(x) exp(x) / (1 + exp(x))
nouvelle_observation <- d[1, ]
nouvelle_observation$sexe[1] <- "Femme"
nouvelle_observation$grpage[1] <- "[45,65]"
nouvelle_observation$etud[1] <- "Primaire"
nouvelle_observation$relig[1] <- "Pratiquant regulier"
nouvelle_observation$heures.tv[1] <- 0
plot(
  broken(mod2, nouvelle_observation, predict.function = betas),
  trans = logit
) + ylim(0, 1) + ylab("Probabilité de faire du sport")
```

Scale for 'y' is already present. Adding another scale for 'y', which will replace the existing scale.

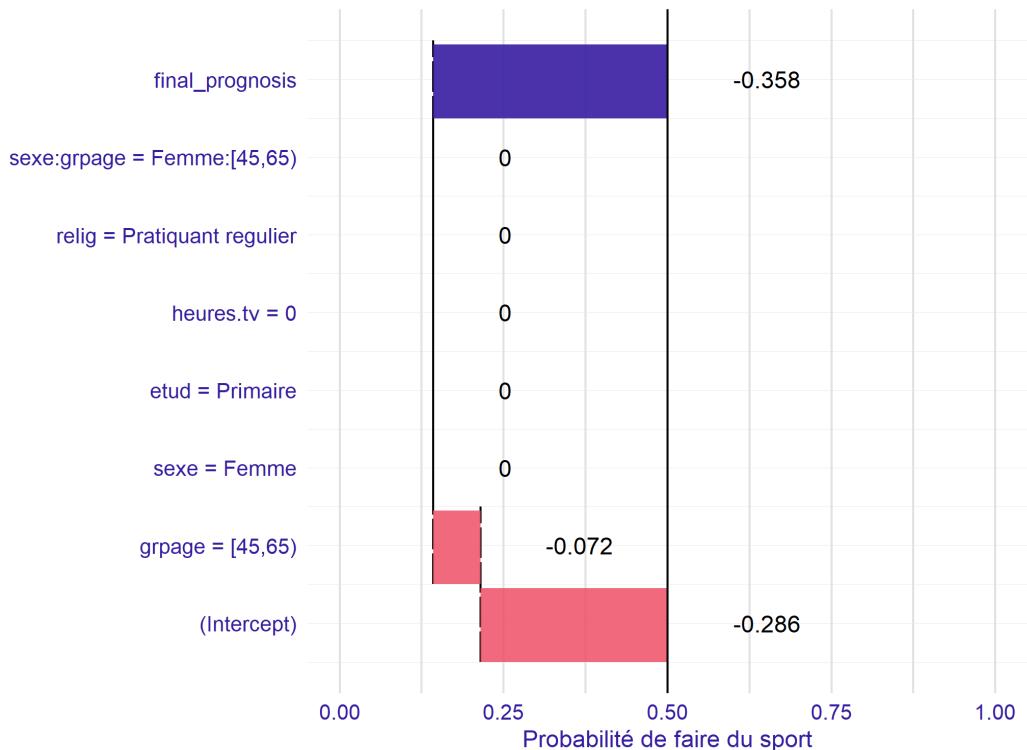


Figure 4. Représentation graphique de l'estimation de la probabilité de faire du sport pour une femme de 60 ans

En premier lieu, l'intercept s'applique et permet de déterminer la probabilité de base de faire du sport

(si toutes les variables sont à leur valeur de référence). «Femme» étant la modalité de référence pour la variable sexe, cela ne modifie pas le calcul de la probabilité de faire du sport. Par contre, il y a une modification induite par la modalité «45-65» de la variable grpae.

Regardons maintenant la situation d'un homme de 20 ans.

```
R> nouvelle_observation$sex[1] <- "Homme"
nouvelle_observation$grpae[1] <- "[16,25]"
plot(
  broken(mod2, nouvelle_observation, predict.function = betas),
  trans = logit
) + ylim(0, 1) + ylab("Probabilité de faire du sport")
```

Scale for 'y' is already present. Adding another scale  
for 'y', which will replace the existing scale.

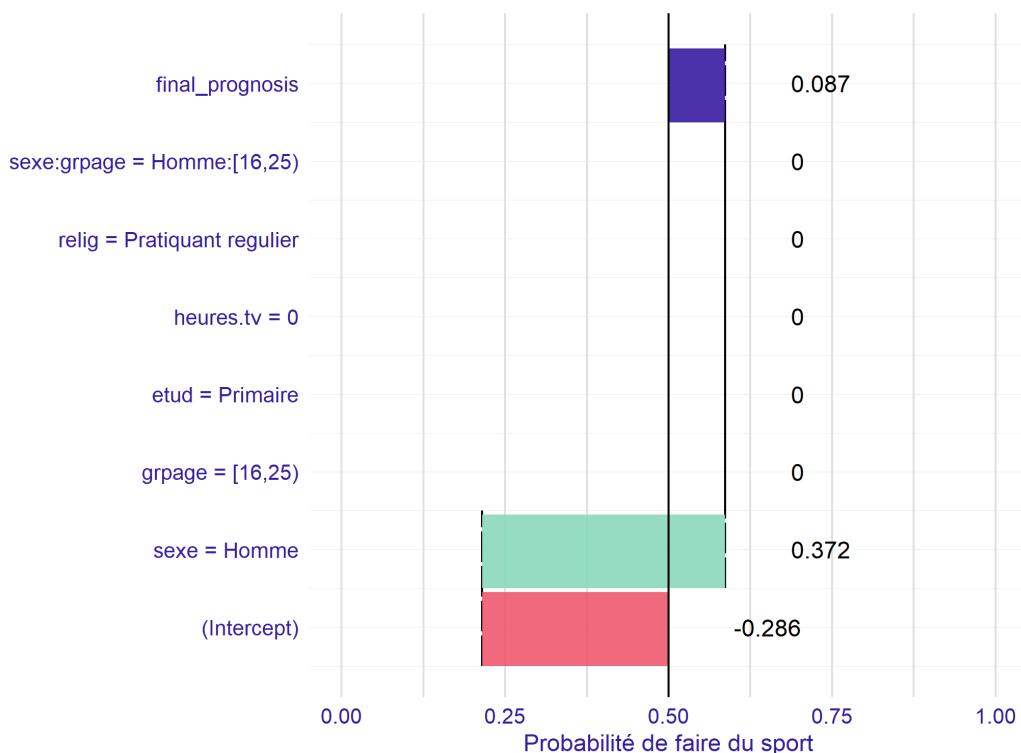


Figure 5. Représentation graphique de l'estimation de la probabilité de faire du sport pour un homme de 20 ans

Nous sommes à la modalité de référence pour l'âge par contre il y a un effet important du sexe. Le coefficient associé globalement à la variable sexe correspond donc à l'effet du sexe à la modalité de

référence du groupe d'âges.

La situation est différente pour un homme de 60 ans.

```
R> nouvelle_observation$grpage[1] <- "[45,65]"
plot(
  broken(mod2, nouvelle_observation, predict.function = betas),
  trans = logit
) + ylim(0, 1) + ylab("Probabilité de faire du sport")
```

Scale for 'y' is already present. Adding another scale  
for 'y', which will replace the existing scale.

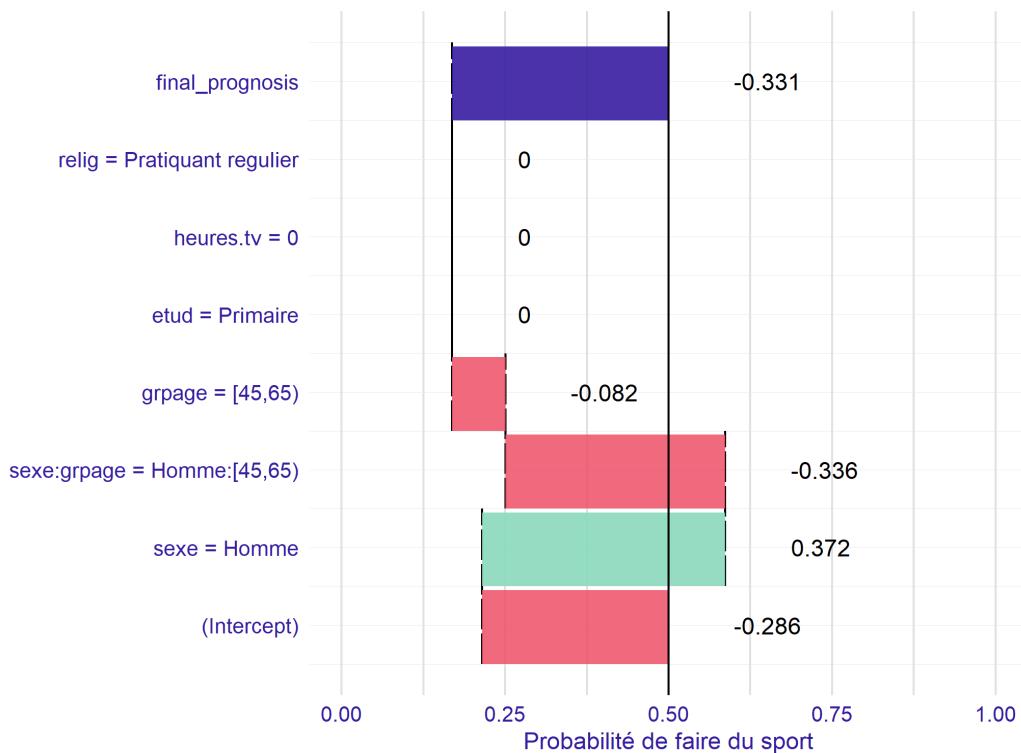


Figure 6. Représentation graphique de l'estimation de la probabilité de faire du sport pour un homme de 60 ans

Cette fois-ci, il y a plusieurs modifications d'effet. On applique en effet à la fois le coefficient «sexe = Homme» (effet du sexe pour les 15-24 ans), le coefficient «grpae = [45-65]» qui est l'effet de l'âge pour les femmes de 45-64 ans et le coefficient «sexe:grpae = Homme:[45-65]» qui indique l'effet spécifique qui s'applique aux hommes de 45-64, d'une part par rapport aux femmes du même et d'autre part par rapport aux hommes de 16-24 ans. L'effet des coefficients d'interaction doivent donc être interprétés par rapport

aux autres coefficients du modèle qui s'appliquent, en tenant compte des modalités de référence.

Il est cependant possible d'écrire le même modèle différemment. En effet, `sex * grpage` dans la formule du modèle est équivalent à l'écriture `sex + grpae + sex:grpae`, c'est-à-dire à modéliser un coefficient global pour chaque variable plus un des coefficients d'interaction. On aurait pu demander juste des coefficients d'interaction, en ne mettant que `sex:grpae`.

```
R> mod3 <- glm(sport ~ sexe:grpage + etud + heures.tv + relig, data = d, family = binomial())
tbl_regression(mod3, exponentiate = TRUE)
```

Caractéristique	OR <sup>1</sup>	95% CI <sup>1</sup>	p-value
Niveau d'étude			
Primaire	—	—	
Secondaire	2,60	1,77 – 3,85	<0,001
Technique/Professionnel	2,88	1,99 – 4,21	<0,001
Supérieur	6,77	4,64 – 10,0	<0,001
Manquant	9,02	4,68 – 17,8	<0,001
Nombre d'heures passées devant la télévision par jour	0,89	0,83 – 0,95	<0,001
Pratique religieuse			
Pratiquant régulier	—	—	
Pratiquant occasionnel	0,99	0,68 – 1,43	>0,9
Appartenance sans pratique	1,02	0,73 – 1,44	0,9
Ni croyance ni appartenance	0,83	0,57 – 1,22	0,3
Rejet	0,68	0,38 – 1,19	0,2
NSP ou NVPR	0,93	0,40 – 2,04	0,9
Sexe * Groupe d'âges			
Femme * [16,25)	1,82	0,92 – 3,58	0,083
Homme * [16,25)	9,46	4,34 – 21,8	<0,001
Femme * [25,45)	1,86	1,17 – 3,00	0,009
Homme * [25,45)	3,02	1,87 – 4,96	<0,001
Femme * [45,65)	1,11	0,69 – 1,80	0,7
Homme * [45,65)	1,35	0,84 – 2,21	0,2
Femme * [65,99]	0,84	0,47 – 1,50	0,6
Homme * [65,99]			

<sup>1</sup> OR = rapport de cotes, CI = intervalle de confiance

Au sens strict, ce modèle explique tout autant le phénomène étudié que le modèle précédent. On peut le

vérifier facilement avec `anova`.

```
R> anova(mod2, mod3, test = "Chisq")
```

De même, les effets modélisés sont les mêmes.

```
R> plot(allEffects(mod3))
```

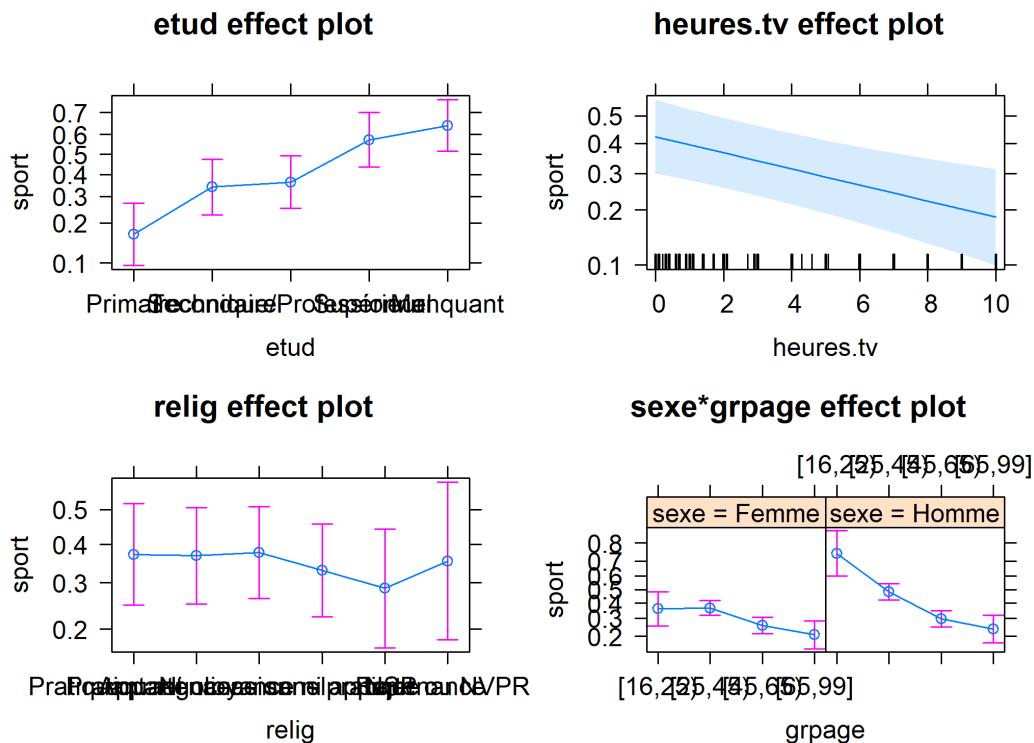


Figure 7. Représentation graphique des effets du modèle avec interaction simple entre le sexe et le groupe d'âge

Par contre, regardons d'un peu plus près les coefficients de ce nouveau modèle. Nous allons voir que leur interprétation est légèrement différente.

```
R> ggcoef_model(mod3, exponentiate = TRUE)
```

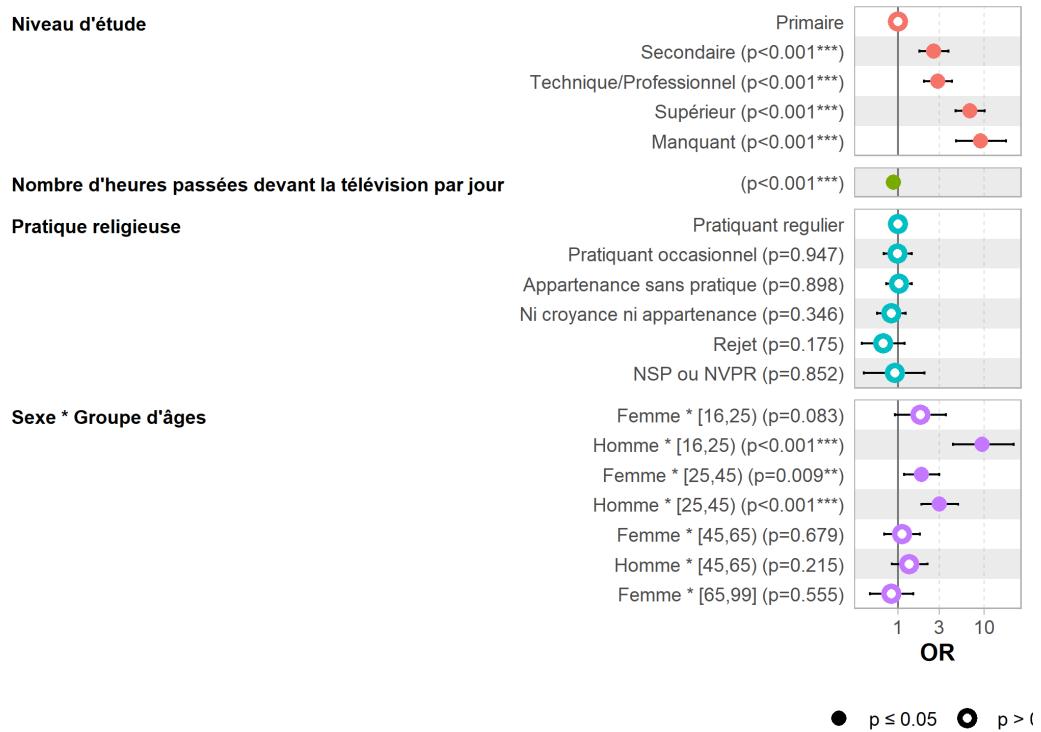


Figure 8. Représentation graphique des coefficients du modèle avec interaction simple entre le sexe et le groupe d'âge

Cette fois-ci, il n'y a plus de coefficients globaux pour la variable sexe ni pour grpage mais des coefficients pour chaque combinaison de ces deux variables.

```
R> plot(
  broken(mod3, nouvelle_observation, predict.function = betas),
  trans = logit
) + ylim(0, 1) + ylab("Probabilité de faire du sport")
```

Scale for 'y' is already present. Adding another scale  
for 'y', which will replace the existing scale.

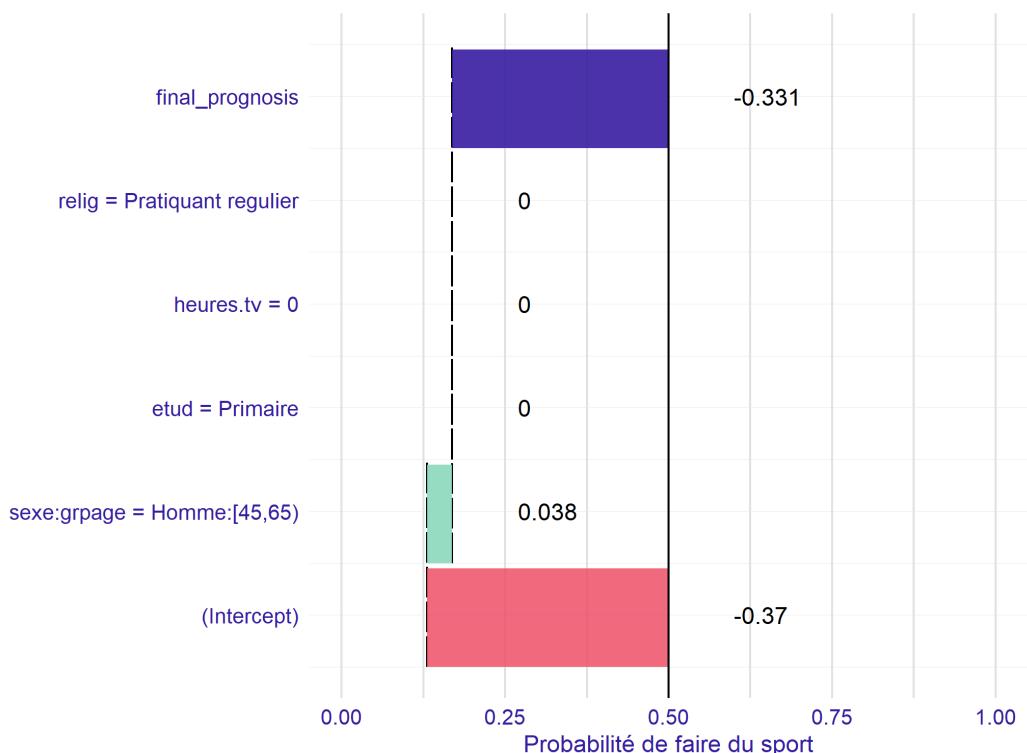


Figure 9. Représentation graphique de l'estimation de la probabilité de faire du sport pour un homme de 40 ans

Cette fois-ci, le coefficient d'interaction fournit l'effet global du sexe et de l'âge, et non plus la modification de cette combinaison par rapport aux coefficients globaux. Leur sens est donc différent et il faudra les interpréter en conséquence.

## Un second exemple d'interaction

Intéressons-nous maintenant à l'interaction entre le sexe et le niveau d'étude. L'effet du niveau d'étude diffère-t-il selon le sexe ?

```
R> mod4 <- glm(sport ~ sexe * etud + grpage + heures.tv + relig, data = d, family = binomial())
```

Regardons d’abord les effets.

```
R> plot(allEffects(mod4))
```

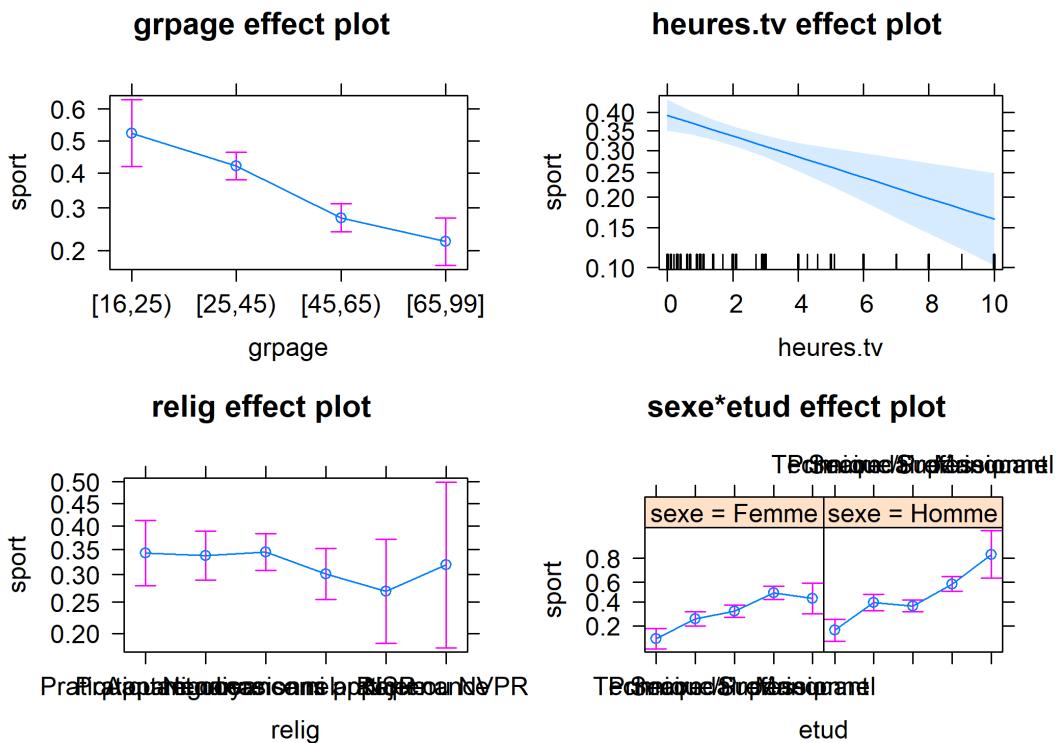


Figure 10. Représentation graphique des effets du modèle avec interaction entre le sexe et le niveau d’étude

À première vue, l’effet du niveau d’étude semble être le même chez les hommes et chez les femmes. Ceci dit, cela serait peut être plus lisible si l’on superposait les deux sexe sur un même graphique. Nous allons utiliser la fonction `ggeffect` de l’extension `ggeffects` qui permet de récupérer les effets calculés avec `effect` dans un format utilisable avec `ggplot2`.

```
R> library(ggeffects)
```

Attaching package: 'ggeffects'

The following object is masked from 'package:cowplot':

get\_title

```
R> plot(ggeffect(mod4, c("etud", "sexe")))
```

Predicted probabilities of Pratique du sport ?

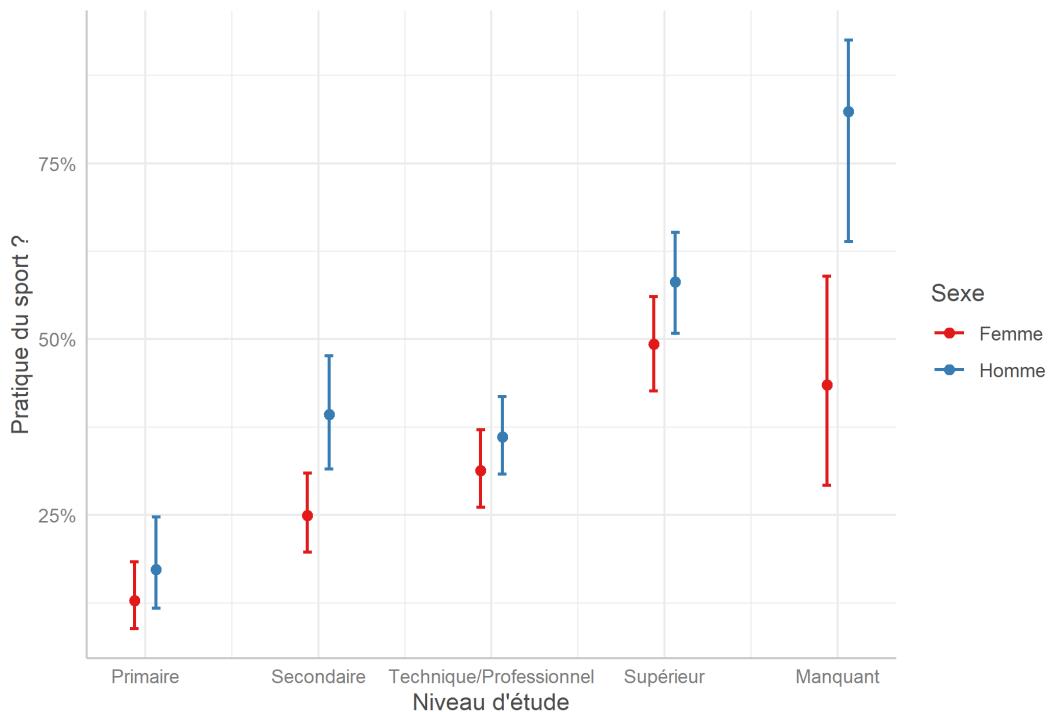


Figure 11. Effets du niveau d'étude selon le sexe

Cela confirme ce que l'on suppose. Regardons les coefficients du modèle.

```
R> ggcoef_model(mod4, exponentiate = TRUE)
```

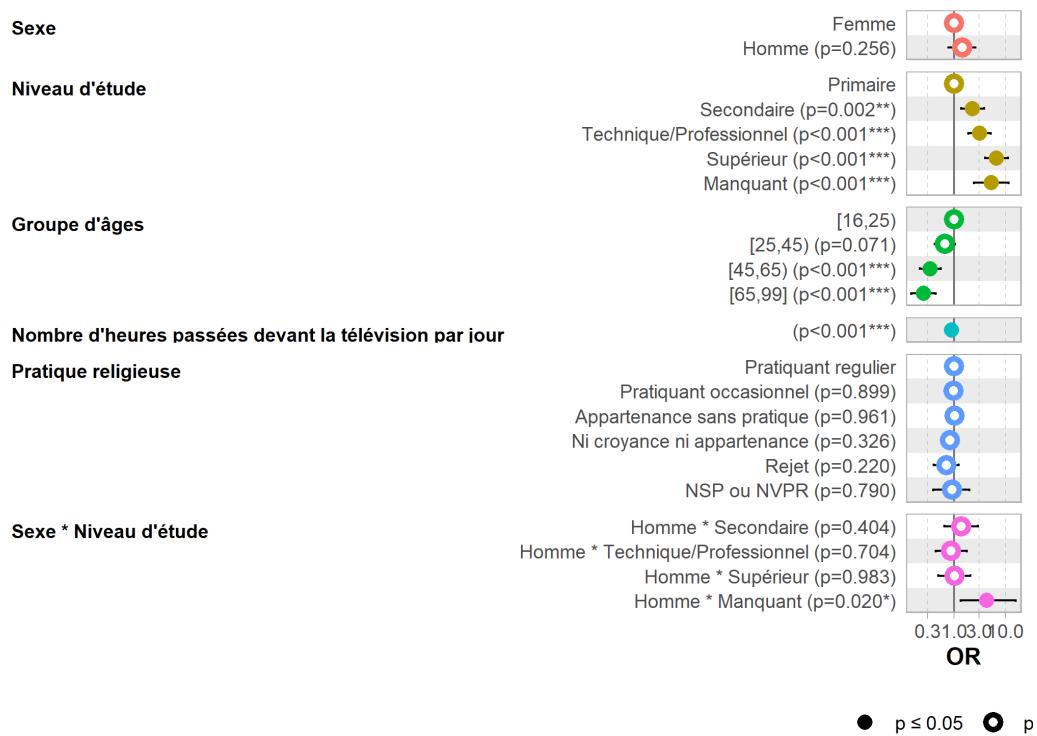


Figure 12. Représentation graphique des coefficients du modèle avec interaction simple entre le sexe et le niveau d'étude

```
R> tbl_regression(mod4, exponentiate = TRUE)
```

Caractéristique	OR <sup>1</sup>	95% CI <sup>1</sup>	p-value
Sexe			
Femme	—	—	
Homme	1,42	0,77 – 2,58	0,3
Niveau d'étude			
Primaire	—	—	
Secondaire	2,25	1,35 – 3,82	0,002
Technique/Professionnel	3,09	1,90 – 5,17	<0,001
Supérieur	6,59	4,02 – 11,1	<0,001
Manquant	5,21	2,42 – 11,4	<0,001
Groupe d'âges			
[16,25)	—	—	
[25,45)	0,66	0,42 – 1,04	0,071
[45,65)	0,34	0,21 – 0,55	<0,001
[65,99]	0,25	0,15 – 0,43	<0,001
Nombre d'heures passées devant la télévision par jour	0,89	0,83 – 0,95	<0,001
Pratique religieuse			
Pratiquant régulier	—	—	
Pratiquant occasionnel	0,98	0,67 – 1,42	0,9
Appartenance sans pratique	1,01	0,72 – 1,43	>0,9
Ni croyance ni appartenance	0,83	0,56 – 1,21	0,3
Rejet	0,70	0,40 – 1,23	0,2
NSP ou NVPR	0,90	0,39 – 1,98	0,8
Sexe * Niveau d'étude			

<sup>1</sup> OR = rapport de cotes, CI = intervalle de confiance

Caractéristique	OR <sup>1</sup>	95% CI <sup>1</sup>	p-value
Homme * Secondaire	1,38	0,65 – 2,93	0,4
Homme * Technique/Professionnel	0,87	0,44 – 1,75	0,7
Homme * Supérieur	1,01	0,49 – 2,07	>0,9
Homme * Manquant	4,28	1,32 – 15,8	0,020

<sup>1</sup> OR = rapport de cotes, CI = intervalle de confiance

Si les coefficients associés au niveau d'étude sont significatifs, ceux de l'interaction ne le sont pas (sauf «sexeHomme:etudManquant») et celui associé au sexe, précédemment significatif ne l'est plus. Testons avec `anova` si l'interaction est belle et bien significative.

```
R> anova(mod4, test = "Chisq")
```

L'interaction est bien significative mais faiblement. Vu que l'effet du niveau d'étude reste néanmoins très similaire selon le sexe, on peut se demander s'il est pertinent de la conserver.

## Explorer les différentes interactions possibles

Il peut y avoir de multiples interactions dans un modèle, d'ordre 2 (entre deux variables) ou plus (entre trois variables ou plus). Il est dès lors tentant de tester les multiples interactions possibles de manière itératives afin d'identifier celles à retenir. C'est justement le but de la fonction `glmulti` de l'extension du même nom. `glmulti` permet de tester toutes les combinaisons d'interactions d'ordre 2 dans un modèle, en retenant le meilleur modèle à partir d'un critère spécifié (par défaut l'AIC). ATTENTION : le temps de calcul de `glmulti` peut-être long.

```
R> library(glmulti)
glmulti(sport ~ sexe + grpage + etud + heures.tv + relig, data = d, family = binomial())
```

```
Initialization...
TASK: Exhaustive screening of candidate set.
Fitting...

After 50 models:
Best model: sport~1+grpage+heures.tv+sexe:heures.tv+grpage:heures.tv+etud:heures.tv
Crit= 2284.87861987263
Mean crit= 2406.80086471225
```

```
After 100 models:  
Best model: sport~1+etud+heures.tv+grpage:heures.tv  
Crit= 2267.79462883348  
Mean crit= 2360.46497457747
```

```
After 150 models:  
Best model: sport~1+grpage+etud+heures.tv+sexe:heures.tv  
Crit= 2228.88574082404  
Mean crit= 2286.60589884071
```

```
After 200 models:  
Best model: sport~1+grpage+etud+heures.tv+sexe:heures.tv  
Crit= 2228.88574082404  
Mean crit= 2254.99359340075
```

```
After 250 models:  
Best model: sport~1+sexe+grpage+etud+heures.tv+etud:sexe+sexe:heures.tv  
Crit= 2226.00088609349  
Mean crit= 2241.76611580481
```

```
After 300 models:  
Best model: sport~1+sexe+grpage+etud+heures.tv+grpage:sexe+sexe:heures.tv  
Crit= 2222.67161519005  
Mean crit= 2234.95020358944
```

On voit qu’au bout d’un moment, l’algorithme se stabilise autour d’un modèle comportant une interaction entre le sexe et l’âge d’une part et entre le sexe et le nombre d’heures passées quotidiennement devant la télé. On voit également que la variable religion a été retirée du modèle final.

```
R> best <- glm(sport ~ 1 + sexe + grpage + etud + heures.tv + grpage:sexe + sex  
e:heures.tv, data = d, family = binomial())  
odds.ratio(best)
```

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

```
R> ggcoef_model(best, exponentiate = TRUE)
```



Figure 13. Représentation graphique des coefficients du modèle avec interaction entre le sexe, le niveau d'étude et le nombre d'heures passées devant la télévision

```
R> plot(allEffects(best))
```

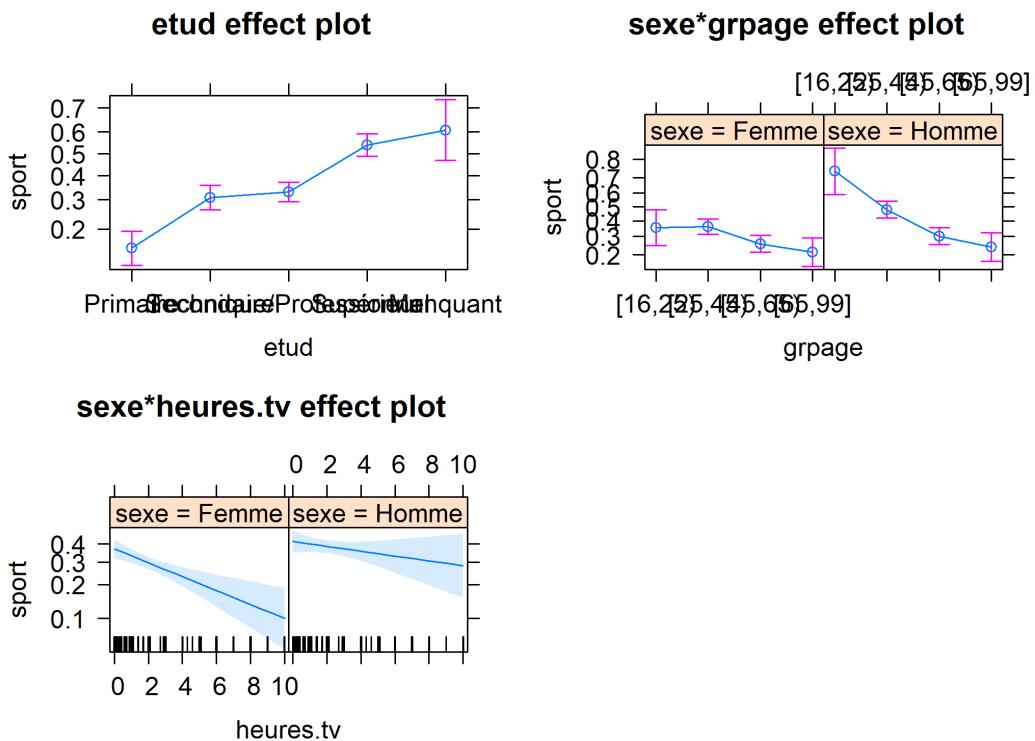


Figure 14. Représentation graphique des effets du modèle avec interaction entre le sexe, le niveau d'étude et le nombre d'heures passées devant la télévision

## Pour aller plus loin

Il y a d'autres extensions dédiées à l'analyse des interactions d'un modèle, de même que de nombreux supports de cours en ligne dédiés à cette question.

On pourra en particulier se référer à la vignette inclue avec l'extension **phia** : <https://cran.r-project.org/web/packages/phia/vignettes/phia.pdf>.

# Multicolinéarité dans la régression

Définition .....	645
Mesure de la colinéarité .....	646
La multicolinéarité est-elle toujours un problème ? .....	648

Dans une régression, la multicolinéarité est un problème qui survient lorsque certaines variables de prévision du modèle mesurent le même phénomène. Une multicolinéarité prononcée s'avère problématique, car elle peut augmenter la variance des coefficients de régression et les rendre instables et difficiles à interpréter. Les conséquences de coefficients instables peuvent être les suivantes :

- les coefficients peuvent sembler non significatifs, même lorsqu'une relation significative existe entre le prédicteur et la réponse ;
- les coefficients de prédicteurs fortement corrélés varieront considérablement d'un échantillon à un autre ;
- lorsque des termes d'un modèle sont fortement corrélés, la suppression de l'un de ces termes aura une incidence considérable sur les coefficients estimés des autres. Les coefficients des termes fortement corrélés peuvent même présenter le mauvais signe.

La multicolinéarité n'a aucune incidence sur l'adéquation de l'ajustement, ni sur la qualité de la prévision. Cependant, les coefficients individuels associés à chaque variable explicative ne peuvent pas être interprétés de façon fiable.

## Définition

Au sens strict, on parle de multicolinéarité parfaite lorsqu'une des variables explicatives d'un modèle est une combinaison linéaire d'une ou plusieurs autres variables explicatives introduites dans le même modèle. L'absence de multicolinéarité parfaite est une des conditions requises pour pouvoir estimer un modèle linéaire et, par extension, un modèle linéaire généralisé (dont les modèles de régression logistique).

Dans les faits, une multicolinéarité parfaite n'est quasiment jamais observée. Mais une forte multicolinéarité entre plusieurs variables peut poser problème dans l'estimation et l'interprétation d'un modèle.

Une erreur fréquente est de confondre multicolinéarité et corrélation. Si des variables colinéaires sont *de facto* fortement corrélées entre elles, deux variables corrélées ne sont pas forcément colinéaires. En termes non statistiques, il y a colinéarité lorsque deux ou plusieurs variables mesurent la «même chose».

Prenons un exemple. Nous étudions les complications après l'accouchement dans différentes maternités d'un pays en développement. On souhaite mettre dans le modèle, à la fois le milieu de résidence (urbain ou rural) et le fait qu'il y ait ou non un médecin dans la clinique. Or, dans la zone d'enquête, les maternités rurales sont dirigées seulement par des sage-femmes tandis que l'on trouve un médecin dans toutes les maternités urbaines sauf une. Dès lors, dans ce contexte précis, le milieu de résidence prédit presque totalement la présence d'un médecin et on se retrouve face à une multicolinéarité (qui serait même parfaite s'il n'y avait pas une clinique urbaine sans médecin). On ne peut donc distinguer l'effet de la présence d'un médecin de celui du milieu de résidence et il ne faut mettre qu'une seule de ces deux variables dans le modèle, sachant que du point de vue de l'interprétation elle capturera à la fois l'effet de la présence d'un médecin et celui du milieu de résidence.

Par contre, si dans notre région d'étude, seule la moitié des maternités urbaines disposait d'un médecin, alors le milieu de résidence n'aurait pas été suffisant pour prédire la présence d'un médecin. Certes, les deux variables seraient corrélées mais pas colinéaires. Un autre exemple de corrélation sans colinéarité, c'est la relation entre milieu de résidence et niveau d'instruction. Il y a une corrélation entre ces deux variables, les personnes résidant en ville étant généralement plus instruites. Cependant, il existe également des personnes non instruites en ville et des personnes instruites en milieu rural. Le milieu de résidence n'est donc pas suffisant pour prédire le niveau d'instruction.

## Mesure de la colinéarité

Il existe différentes mesures de la multicolinéarité. L'extension `mctest` en fournit plusieurs, mais elle n'est utilisable que si l'ensemble des variables explicatives sont de type numérique.

L'approche la plus classique consiste à examiner les *facteurs d'inflation de la variance* (FIV) ou *variance inflation factor* (VIF) en anglais. Les FIV estimentent de combien la variance d'un coefficient est «augmentée» en raison d'une relation linéaire avec d'autres prédicteurs. Ainsi, un FIV de 1,8 nous dit que la variance de ce coefficient particulier est supérieure de 80 % à la variance que l'on aurait dû observer si ce facteur n'est absolument pas corrélé aux autres prédicteurs.

Si tous les FIV sont égaux à 1, il n'existe pas de multicolinéarité, mais si certains FIV sont supérieurs à 1, les prédicteurs sont corrélés. Il n'y a pas de consensus sur la valeur au-delà de laquelle on doit considérer qu'il y a multicolinéarité. Certains auteurs, comme Paul Allison<sup>1</sup>, disent regarder plus en détail les variables avec un FIV supérieur à 2,5. D'autres ne s'inquiètent qu'à partir de 5. Il n'existe pas de test statistique qui

---

1. <https://statisticalhorizons.com/multicollinearity>

permettrait de dire s'il y a colinéarité ou non<sup>2</sup>.

L'extension `car` fournit une fonction `vif` permettant de calculer les FIV à partir d'un modèle. Elle implémente même une version «généralisée» permettant de considérer des facteurs catégoriels et des modèles linéaires généralisés comme la régression logistique.

Reprendons, pour exemple, un modèle logistique que nous avons déjà abordé dans d'autres chapitres.

```
R> library(questionr)
data(hdv2003)
d <- hdv2003
d$sexe <- relevel(d$sexe, "Femme")
d$grpage <- cut(d$age, c(16, 25, 45, 65, 99), right = FALSE, include.lowest =
TRUE)
d$etud <- d$nivetud
levels(d$etud) <- c(
  "Primaire", "Primaire", "Primaire", "Secondaire", "Secondaire",
  "Technique/Professionnel", "Technique/Professionnel", "Supérieur"
)
d$etud <- addNAstr(d$etud, "Manquant")
mod <- glm(sport ~ sexe + grpae + etud + heures.tv + relig, data = d, family =
  binomial())
```

Le calcul des FIV se fait simplement en passant le modèle à la fonction `vif`.

```
R> library(car)
```

```
Attaching package: 'car'
```

```
The following object is masked _by_ '.GlobalEnv':
```

```
logit
```

```
The following object is masked from 'package:purrr':
```

```
some
```

```
The following object is masked from 'package:dplyr':
```

```
recode
```

---

2. Pour plus de détails, voir ce [post de David Giles](#) qui explique pourquoi ce n'est pas possible.

```
R> vif(mod)
```

	GVIF	Df	GVIF^(1/(2*Df))
sexe	1.048	1	1.024
grpage	1.842	3	1.107
etud	1.848	4	1.080
heures.tv	1.062	1	1.031
relig	1.121	5	1.011

Dans notre exemple, tous les FIV sont proches de 1. Il n'y a donc pas de problème potentiel de colinéarité à explorer.

## La multicolinéarité est-elle toujours un problème ?

Là encore, il n'y a pas de consensus sur cette question. Certains analystes considèrent que tout modèle où certains prédicteurs seraient colinéaires n'est pas valable. Dans un [billet sur internet](#), Paul Allison évoque quant à lui des situations où la multicolinéarité peut être ignorée en toute sécurité. Le texte ci-dessous est une traduction du billet de Paul Allison.

### 1. Les variables avec des FIV élevés sont des variables de contrôle, et les variables d'intérêt n'ont pas de FIV élevés.

Voici le problème de la multicolinéarité : ce n'est un problème que pour les variables qui sont colinéaires. Il augmente les erreurs-types de leurs coefficients et peut rendre ces coefficients instables de plusieurs façons. Mais tant que les variables colinéaires ne sont utilisées que comme variables de contrôle, et qu'elles ne sont pas colinéaires avec vos variables d'intérêt, il n'y a pas de problème. Les coefficients des variables d'intérêt ne sont pas affectés et la performance des variables de contrôle n'est pas altérée.

Voici un exemple tiré de ces propres travaux : l'échantillon est constitué de collèges américains, la variable dépendante est le taux d'obtention de diplôme et la variable d'intérêt est un indicateur (factice) pour les secteurs public et privé. Deux variables de contrôle sont les scores moyens au SAT et les scores moyens à l'ACT pour l'entrée en première année. Ces deux variables ont une corrélation supérieure à ,9, ce qui correspond à des FIV d'au moins 5,26 pour chacune d'entre elles. Mais le FIV pour l'indicateur public/privé n'est que de 1,04. Il n'y a donc pas de problème à se préoccuper et il n'est pas nécessaire de supprimer l'un ou l'autre des deux contrôles, à condition que l'on ne cherche pas à interpréter ou comparer l'un par rapport à l'autre les coefficients de ces deux variables de contrôle.

### 2. Les FIV élevés sont causés par l'inclusion de puissances ou de produits d'autres variables.

Si vous spécifiez un modèle de régression avec  $x$  et  $x^2$ , il y a de bonnes chances que ces deux variables soient fortement corrélées. De même, si votre modèle a  $x$ ,  $z$  et  $xz$ ,  $x$  et  $z$  sont susceptibles d'être fortement corrélés avec leur produit. Il n'y a pas de quoi s'inquiéter, car la valeur  $p$  de  $xz$  n'est pas affectée par la multicolinéarité. Ceci est facile à démontrer : vous pouvez réduire considérablement les corrélations en

«centrant» les variables (c'est-à-dire en soustrayant leurs moyennes) avant de créer les puissances ou les produits. Mais la valeur  $p$  pour  $x^2$  ou pour  $xz$  sera exactement la même, que l'on centre ou non. Et tous les résultats pour les autres variables (y compris le  $R^2$  mais sans les termes d'ordre inférieur) seront les mêmes dans les deux cas. La multicollinearité n'a donc pas de conséquences négatives.

### **3. Les variables avec des FIV élevés sont des variables indicatrices (factices) qui représentent une variable catégorielle avec trois catégories ou plus.**

Si la proportion de cas dans la catégorie de référence est faible, les variables indicatrices auront nécessairement des FIV élevés, même si la variable catégorielle n'est pas associée à d'autres variables dans le modèle de régression.

Supposons, par exemple, qu'une variable de l'état matrimonial comporte trois catégories : actuellement marié, jamais marié et anciennement marié. Vous choisissez «anciennement marié» comme catégorie de référence, avec des variables d'indicateur pour les deux autres. Ce qui se passe, c'est que la corrélation entre ces deux indicateurs devient plus négative à mesure que la fraction de personnes dans la catégorie de référence diminue. Par exemple, si 45 % des personnes ne sont jamais mariées, 45 % sont mariées et 10 % sont anciennement mariées, les valeurs du FIV pour les personnes mariées et les personnes jamais mariées seront d'au moins 3,0.

Est-ce un problème ? Eh bien, cela signifie que les valeurs  $p$  des variables indicatrices peuvent être élevées. Mais le test global selon lequel tous les indicateurs ont des coefficients de zéro n'est pas affecté par des FIV élevés. Et rien d'autre dans la régression n'est affecté. Si vous voulez vraiment éviter des FIV élevés, il suffit de choisir une catégorie de référence avec une plus grande fraction des cas. Cela peut être souhaitable pour éviter les situations où aucun des indicateurs individuels n'est statistiquement significatif, même si l'ensemble des indicateurs est significatif.



# Analyse de survie

Ressources en ligne . . . . .	651
Un exemple concret : mortalité infanto-juvénile . . . . .	652
Préparation des données avec data.table . . . . .	652
Préparation des données avec dplyr . . . . .	658
Kaplan-Meier . . . . .	664
Modèle de Cox . . . . .	668
Vérification de la validité du modèle . . . . .	674
Données pondérées . . . . .	677

## Ressources en ligne

L'extension centrale pour l'analyse de survie est **survival**.

Un très bon tutoriel (en anglais et en 3 étapes), introduisant les concepts de l'analyse de survie, des courbes de Kaplan-Meier et des modèles de Cox et leur mise en oeuvre pratique sous R est disponible en ligne :

- <http://www.sthda.com/english/wiki/survival-analysis-basics>
- <http://www.sthda.com/english/wiki/cox-proportional-hazards-model>
- <http://www.sthda.com/english/wiki/cox-model-assumptions>

Pour un autre exemple (toujours en anglais) d'analyse de survie avec **survival**, on pourra se référer à <https://rpubs.com/vinubalan/hrsurvival>.

Pour représenter vos résultats avec **ggplot2**, on pourra avoir recours à l'extension **survminer** présentée en détails sur son site officiel (en anglais) : <http://www.sthda.com/english/rpkgs/survminer/>. On pourra également avoir recours à la fonction **ggsurv** de l'extension **GGally** présentée à l'adresse <http://ggobi.github.io/ggally/#ggallyggsurv>.

A noter, il est possible d'utiliser la fonction **step** sur un modèle de Cox, pour une sélection pas à pas d'un meilleur modèle basé sur une minimisation de l'AIC (voir le chapitre sur la régression logistique, page 507).

L'excellente extension **broom** peut également être utilisée sur des modèles de survie (Kaplan-Meier ou Cox) pour en convertir les résultats sous la forme d'un tableau de données.

Pour approfondir les possibilités offertes par l’extension **survival**, on pourra également consulter les différentes vignettes fournies avec l’extension (voir <https://cran.r-project.org/package=survival>).

## Un exemple concret : mortalité infanto-juvénile

Dans cet exemple, nous allons utiliser le jeu de données `fecondite` fourni par l’extension **questionr**. Ce jeu de données comporte trois tableaux de données : `menages`, `femmes` et `enfants`.

Nous souhaitons étudier ici la survie des enfants entre la naissance et l’âge de 5 ans. Dans un premier temps, nous comparerons la survie des jeunes filles et des jeunes garçons. Dans un second temps, nous procéderons à une analyse multivariée en prenant en compte les variables suivantes :

- sexe de l’enfant
- milieu de résidence
- niveau de vie du ménage
- structure du ménage
- niveau d’éducation de la mère
- âge de la mère à la naissance de l’enfant
- enfin, une variable un peu plus compliquée, à savoir si le rang de naissance de l’enfant (second, troisième, quatrième, etc.) est supérieur au nombre idéal d’enfants selon la mère.

Nous allons préparer les données selon deux approches : soit en utilisant l’extension **data.table** (voir le chapitre dédié à `data.table`, page 251), soit en utilisant l’extension **dplyr** (voir le chapitre sur `dplyr`, page 233).

Chargeons les données en mémoire et listons les variables disponibles.

```
R> library(questionr, quietly = TRUE)
  data(fecondite)
  lookfor(menages)
```

```
R> lookfor(femmes)
```

```
R> lookfor(enfants)
```

## Préparation des données avec `data.table`

Tout d’abord, regardons sous quel format elles sont stockées.

```
R> class(menages)
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

```
R> describe(menages)
```

```
[1814 obs. x 5 variables] tbl_df tbl data.frame
```

```
$id_menage: Identifiant du ménage
numeric: 1 2 3 4 5 6 7 8 9 10 ...
min: 1 - max: 1814 - NAs: 0 (0%) - 1814 unique values
```

```
$taille: Taille du ménage (nombre de membres)
numeric: 7 3 6 5 7 6 15 6 5 19 ...
min: 1 - max: 31 - NAs: 0 (0%) - 30 unique values
```

```
$sexe_chef: Sexe du chef de ménage
labelled double: 2 1 1 1 1 2 2 2 1 1 ...
min: 1 - max: 2 - NAs: 0 (0%) - 2 unique values
2 value labels: [1] homme [2] femme
```

```
$structure: Structure démographique du ménage
labelled double: 4 2 5 4 4 4 5 2 5 5 ...
min: 1 - max: 5 - NAs: 0 (0%) - 5 unique values
6 value labels: [0] pas d'adulte [1] un adulte [2] deux adultes de sexe opposé
[3] deux adultes de même sexe [4] trois adultes ou plus avec lien de parenté
[5] adultes sans lien de parenté
```

```
$richesse: Niveau de vie (quintiles)
labelled double: 1 2 2 1 1 3 2 5 4 3 ...
min: 1 - max: 5 - NAs: 0 (0%) - 5 unique values
5 value labels: [1] très pauvre [2] pauvre [3] moyen [4] riche [5] très riche
```

Les tableaux de données sont au format `tibble` (c'est-à-dire sont de la classe `tbl_df`) et les variables catégorielles sont du type `haven_labelled` (voir le chapitre sur les vecteurs labellisés, page 123). Ce format correspond au format de données si on les avait importées depuis SPSS avec l'extension `haven` (voir le chapitre sur l'import de données, page 150).

En premier lieu, il nous faut convertir les tableaux de données au format `data.table`, ce qui peut se faire avec la fonction `setDT`. Par ailleurs, nous allons également charger en mémoire l'extension `labelled` pour la gestion des vecteurs labellisés.

```
R> library(labelled)
library(data.table)
setDT(menages)
setDT(femmes)
setDT(enfants)
```

En premier lieu, il nous faut calculer la durée d'observation des enfants, à savoir le temps passé entre la date de naissance (variable du fichier `enfants`) et la date de passation de l'entretien (fournie par le tableau de données `femmes`). Pour récupérer des variables du fichier `femmes` dans le fichier `enfants`, nous allons procéder à une fusion de table (voir le chapitre dédié, page 269). Pour le calcul de la durée d'observation, nous allons utiliser le package `lubridate` (voir le chapitre calculer un âge, page 1103 et celui sur la gestion des dates, page 285). Nous effectuerons l'analyse en mois (puisque l'âge au décès est connu en mois). Dès lors, la durée d'observation sera calculée en mois.

```
R> enfants <- merge(
  enfants,
  femmes[, .(id_femme, date_entretien)],
  by = "id_femme",
  all.x = TRUE
)

# duree observation en mois
library(lubridate, quietly = TRUE)
enfants[, duree_observation := time_length(interval(date_naissance, date_entretien), unit = "months")]
```

ATTENTION : il y a 11 enfants soi-disant nés après la date d'enquête ! Quelle que soit l'enquête, il est rare de ne pas observer d'incohérences. Dans le cas présent, il est fort possible que la date d'entretien puisse parfois être erronée (par exemple si l'enquêteur a inscrit une date sur le questionnaire papier le jour du recensement du ménage mais n'a pu effectué le questionnaire individuel que plus tard). Nous décidons ici de procéder à une correction en ajoutant un mois aux dates d'entretien problématiques. D'autres approches auraient pu être envisagées, comme par exemple exclure ces observations problématiques. Cependant, cela aurait impacté le calcul du range de naissance pour les autres enfants issus de la même mère. Quoiqu'il en soit, il n'y a pas de réponse unique. À vous de vous adapter au contexte particulier de votre analyse.

```
R> enfants[duree_observation < 0, date_entretien := date_entretien %m+% month
s(1)]
enfants[, duree_observation := time_length(interval(date_naissance, date_entretien), unit = "months")]
```

Regardons maintenant comment les âges au décès ont été collectés.

```
R> freq(enfants$age_deces)
```

Les âges au décès sont ici exprimés en mois révolus. Les décès à un mois révolu correspondent à des décès entre 1 et 2 mois exacts. Par ailleurs, les durées d'observation que nous avons calculées avec `time_length` sont des durées exactes, c'est-à-dire avec la partie décimale. Pour une analyse de survie, on ne peut mélanger des durées exactes et des durées révolues. Trois approches peuvent être envisagées :

1. faire l'analyse en mois révolus, auquel cas on ne gardera que la partie entière des durées d'observations avec la fonction `trunc` ;
2. considérer qu'un âge au décès de 3 mois révolus correspond en moyenne à 3,5 mois exacts et donc ajouter 0,5 à tous les âges révolus ;
3. imputer un âge au décès exact en distribuant aléatoirement les décès à 3 mois révolus entre 3 et 4 mois exacts, autrement dit en ajoutant aléatoirement une partie décimale aux âges révolus.

Nous allons ici adopter la troisième approche en considérant que les décès se répartissent de manière uniforme au sein d'un même mois. Nous aurons donc recours à la fonction `runif` qui permet de générer des valeurs aléatoires entre 0 et 1 selon une distribution uniforme.

```
R> enfants[, age_deces_impute := age_deces + runif(.N)]
```

Pour définir notre objet de survie, il nous faudra deux variables. Une première, temporelle, indiquant la durée à laquelle survient l'événement étudié (ici le décès) pour ceux ayant vécu l'événement et la durée d'observation pour ceux n'ayant pas vécu l'événement (censure à droite). Par ailleurs, une seconde variable indiquant si les individus ont vécu l'événement (0 pour non, 1 pour oui). Or, ici, la variable `survie` est codée 0 pour les décès et 1 pour ceux ayant survécu. Pour plus de détails, voir l'aide de la fonction `Surv`.

```
R> enfants[, deces := 0]
enfants[surveie == 0, deces := 1]
var_label(enfants$deces) <- "Est décédé ?"
val_labels(enfants$deces) <- c(non = 0, oui = 1)

enfants[, time := duree_observation]
enfants[deces == 1, time := age_deces_impute]
```

Occupons-nous maintenant des variables explicatives que nous allons inclure dans l'analyse. Tout d'abord, ajoutons à la table `enfants` les variables nécessaires des tables `femmes` et `menages`. Notons qu'il nous faudra importer `id_menage` de la table `femmes` pour pouvoir fusionner ensuite la table `enfants` avec la table `menages`. Par ailleurs, pour éviter une confusion sur la variable `date_naissance`, nous renommons à la volée cette variable de la table `femmes` en `date_naissance_mere`.

```
R> enfants <- merge(
  enfants,
  femmes[, .(
    id_femme, id_menage, milieu, educ,
    date_naissance_mere = date_naissance, nb_enf_ideal
  )],
  by = "id_femme",
  all.x = TRUE
)
enfants <- merge(
  enfants,
  menages[, .(id_menage, structure, richesse)],
  by = "id_menage",
  all.x = TRUE
)
```

Les variables catégorielles sont pour l'heure sous formes de vecteurs labellisés. Or, dans un modèle, il est impératif de les convertir en facteurs pour qu'elles soient bien traitées comme des variables catégorielles (autrement elles seraient traitées comme des variables continues). On aura donc recours à la fonction `to_factor` de l'extension `labelled`.

```
R> enfants[, sexe := to_factor(sexe)]
enfants[, richesse := to_factor(richesse)]
```

Regardons plus attentivement, la variable *structure*.

```
R> freq(enfants$structure)
```

Tout d'abord, la modalité «pas d'adulte» n'est pas représentée dans l'échantillon. On aura donc recours à l'argument `drop_unused_labels` pour ne pas conserver cette modalité. Par ailleurs, nous considérons que la situation familiale à partir de laquelle nous voudrons comparer les autres dans notre modèle, donc celle qui doit être considérée comme la modalité de référence, est celle du ménage nucléaire. Cette modalité («deux adultes de sexe opposé») n'étant pas la première, nous aurons recours à la fonction `relevel`.

```
R> enfants[, structure := to_factor(structure, drop_unused_labels = TRUE)]
enfants[, structure := relevel(structure, "deux adultes de sexe opposé")]
```

Regardons la variable *educ*.

```
R> freq(enfants$educ)
```

La modalité «supérieur» est peu représentée dans notre échantillon. Nous allons la fusionner avec la modalité «secondaire» (voir la section Regrouper les modalités d'une variable, page 213 du chapitre

Recodage, page 197).

```
R> enfants[, educ2 := educ]
  enfants[educ == 3, educ2 := 2]
  val_label(enfants$educ2, 2) <- "secondaire ou plus"
  val_label(enfants$educ2, 3) <- NULL
  enfants[, educ2 := to_factor(educ2)]
  freq(enfants$educ2)
```

Calculons maintenant l'âge de la mère à la naissance de l'enfant (voir le chapitre Calculer un âge, page 1103) et découpons le en groupes d'âges (voir la section Découper une variable numérique en classes, page 209 du chapitre Recodage, page 197).

```
R> enfants[, age_mere_naissance := time_length(
  interval(date_naissance_mere, date_naissance),
  unit = "years"
  )]

enfants$gpage_mere_naissance <- cut(
  enfants$age_mere_naissance,
  include.lowest = TRUE, right = FALSE,
  breaks=c(13, 20, 30, 50)
)
levels(enfants$gpage_mere_naissance) <- c(
  "19 ou moins", "20-29", "30 et plus"
)
enfants$gpage_mere_naissance <- relevel(enfants$gpage_mere_naissance, "20-29")
freq(enfants$gpage_mere_naissance)
```

Reste à calculer si le rang de naissance de l'enfant est supérieur au nombre idéal d'enfants tel que défini par la mère. On aura recours à la fonction `rank` appliquée par groupe (ici calculé séparément pour chaque mère). L'argument `ties.method` permet d'indiquer comment gérer les égalités (ici les naissances multiples, e.g. les jumeaux). Comme nous voulons comparer le rang de l'enfant au nombre idéal d'enfants, nous allons retenir la méthode `"max"` pour obtenir, dans le cas présent, le nombre total d'enfants déjà nés<sup>1</sup>. Avant de calculer un rang, il est impératif de trier préalablement le tableau (voir le chapitre Tris, page 257).

---

1. Ici, pour plus de simplicité, nous n'avons pas pris en compte les décès éventuels des enfants de rang inférieur avant la naissance considérée.

```
R> setorder(enfants, id_femme, date_naissance)
enfants[, rang := rank(date_naissance, ties.method = "max"), by = id_femme]
enfants[, rang_apres_ideal := "non"]
# note: unclass() requis en raison d'un bug non corrigé dans haven empêchant de comparer haven_labelled_spss et integer
enfants[rang > unclass(nb_enf_ideal), rang_apres_ideal := "oui"]
enfants[, rang_apres_ideal := factor(rang_apres_ideal)]
enfants[, rang_apres_ideal := relevel(rang_apres_ideal, "non")]
```

## Préparation des données avec dplyr

Tout d'abord, regardons sous quel format elles sont stockées.

```
R> data(fecondite)
class(menages)
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

```
R> describe(menages)
```

```
[1814 obs. x 5 variables] tbl_df tbl data.frame

$id_menage: Identifiant du ménage
numeric: 1 2 3 4 5 6 7 8 9 10 ...
min: 1 - max: 1814 - NAs: 0 (0%) - 1814 unique values

$taille: Taille du ménage (nombre de membres)
numeric: 7 3 6 5 7 6 15 6 5 19 ...
min: 1 - max: 31 - NAs: 0 (0%) - 30 unique values

$sexe_chef: Sexe du chef de ménage
labelled double: 2 1 1 1 1 2 2 2 1 1 ...
min: 1 - max: 2 - NAs: 0 (0%) - 2 unique values
2 value labels: [1] homme [2] femme

$structure: Structure démographique du ménage
labelled double: 4 2 5 4 4 4 5 2 5 5 ...
min: 1 - max: 5 - NAs: 0 (0%) - 5 unique values
6 value labels: [0] pas d'adulte [1] un adulte [2] deux adultes de sexe opposé
[3] deux adultes de même sexe [4] trois adultes ou plus avec lien de parenté
[5] adultes sans lien de parenté
```

```
$richesse: Niveau de vie (quintiles)
labelled double: 1 2 2 1 1 3 2 5 4 3 ...
min: 1 - max: 5 - NAs: 0 (0%) - 5 unique values
5 value labels: [1] très pauvre [2] pauvre [3] moyen [4] riche [5] très riche
```

Les tableaux de données sont déjà au format *tibble* (c'est-à-dire sont de la classe `tbl_df`)<sup>2</sup> et les variables catégorielles sont du type `labelled` (voir le chapitre sur les vecteurs labellisés, page 123). Ce format correspond au format de données si on les avait importées depuis SPSS avec l'extension `haven` (voir le chapitre sur l'import de données, page 150).

Nous allons charger en mémoire l'extension `labelled` pour la gestion des vecteurs labellisés en plus de `dplyr`.

```
R> library(dplyr)
library(labelled)
```

En premier lieu, il nous faut calculer la durée d'observation des enfants, à savoir le temps passé entre la date de naissance (variable du fichier `enfants`) et la date de passation de l'entretien (fournie par le tableau de données `femmes`). Pour récupérer des variables du fichier `femmes` dans le fichier `enfants`, nous allons procéder à une fusion de table (voir le chapitre dédié, page 269). Pour le calcul de la durée d'observation, nous allons utiliser le package `lubridate` (voir le chapitre calculer un âge, page 1103 et celui sur la gestion des dates, page 285). Nous effectuerons l'analyse en mois (puisque l'âge au décès est connu en mois). Dès lors, la durée d'observation sera calculée en mois.

**ATTENTION :** les étiquettes de valeurs sont le plus souvent perdus lors des fusions de tables avec `dplyr`. Pour les récupérer après fusion, nous allons conserver une version originale du tableau de données `enfants` et utiliser la fonction `copy_labels_from` de l'extension `labelled`.

---

2. Si cela n'avait pas été le cas, nous aurions eu recours à la fonction `tbl_df`.

```
R> enfants_original <- enfants

library(lubridate)
enfants <- enfants %>%
  left_join(
    femmes %>% select(id_femme, date_entretien),
    by = "id_femme"
  ) %>%
  copy_labels_from(enfants_original) %>%
  mutate(duree_observation = time_length(
    interval(date_naissance, date_entretien),
    unit = "months"
  ))
```

ATTENTION : il y a 11 enfants soi-disant nés après la date d'enquête ! Quelle que soit l'enquête, il est rare de ne pas observer d'incohérences. Dans le cas présent, il est fort possible que la date d'entretien puisse parfois être erronée (par exemple si l'enquêteur a inscrit une date sur le questionnaire papier le jour du recensement du ménage mais n'a pu effectué le questionnaire individuel que plus tard). Nous décidons ici de procéder à une correction en ajoutant un mois aux dates d'entretien problématiques. D'autres approches auraient pu être envisagées, comme par exemple exclure ces observations problématiques. Cependant, cela aurait impacté le calcul du range de naissance pour les autres enfants issus de la même mère. Quoiqu'il en soit, il n'y a pas de réponse unique. À vous de vous adapter au contexte particulier de votre analyse.

```
R> enfants$date_entretien[enfants$duree_observation < 0] <-
  enfants$date_entretien[enfants$duree_observation < 0] %m+% months(1)
enfants <- enfants %>%
  mutate(duree_observation = time_length(
    interval(date_naissance, date_entretien),
    unit = "months"
  ))
```

Regardons maintenant comment les âges au décès ont été collectés.

```
R> freq(enfants$age_deces)
```

Les âges au décès sont ici exprimés en mois révolus. Les décès à un mois révolu correspondent à des décès entre 1 et 2 mois exacts. Par ailleurs, les durées d'observation que nous avons calculées avec `time_length` sont des durées exactes, c'est-à-dire avec la partie décimale. Pour une analyse de survie, on ne peut mélanger des durées exactes et des durées révolues. Trois approches peuvent être envisagées :

1. faire l'analyse en mois révolus, auquel cas on ne gardera que la partie entière des durées d'observations avec la fonction `trunc` ;
2. considérer qu'un âge au décès de 3 mois révolus correspond en moyenne à 3,5 mois exacts et donc ajouter 0,5 à tous les âges révolus ;

3. imputer un âge au décès exact en distribuant aléatoirement les décès à 3 mois révolus entre 3 et 4 mois exacts, autrement dit en ajoutant aléatoirement une partie décimale aux âges révolus.

Nous allons ici adopter la troisième approche en considérant que les décès se répartissent de manière uniforme au sein d'un même mois. Nous aurons donc recours à la fonction `runif` qui permet de générer des valeurs aléatoires entre 0 et 1 selon une distribution uniforme.

```
R> enfants <- enfants %>%
  dplyr::mutate(age_deces_impute = age_deces + runif(n()))
```

Pour définir notre objet de survie, il nous faudra deux variables. Une première, temporelle, indiquant la durée à laquelle survient l'événement étudié (ici le décès) pour ceux ayant vécu l'événement et la durée d'observation pour ceux n'ayant pas vécu l'événement (censure à droite). Par ailleurs, une seconde variable indiquant si les individus ont vécu l'événement (0 pour non, 1 pour oui). Or, ici, la variable `survie` est codée 0 pour les décès et 1 pour ceux ayant survécu. Pour plus de détails, voir l'aide de la fonction `Surv`.

```
R> enfants <- enfants %>%
  mutate(deces = if_else(survie == 0, 1, 0)) %>%
  set_variable_labels(deces = "Est décédé ?") %>%
  set_value_labels(deces = c(non = 0, oui = 1)) %>%
  mutate(time = if_else(deces == 1, age_deces_impute, duree_observation))
```

Occupons-nous maintenant des variables explicatives que nous allons inclure dans l'analyse. Tout d'abord, ajoutons à la table `enfants` les variables nécessaires des tables `femmes` et `menages`. Notons qu'il nous faudra importer `id_menage` de la table `femmes` pour pouvoir fusionner ensuite la table `enfants` avec la table `menages`. Par ailleurs, pour éviter une confusion sur la variable `date_naissance`, nous renommons à la volée cette variable de la table `femmes` en `date_naissance_mere`.

```
R> enfants <- enfants %>%
  left_join(
    select(femmes,
           id_femme, id_menage, milieu, educ,
           date_naissance_mere = date_naissance, nb_enf_ideal
    ),
    by = "id_femme"
  ) %>%
  left_join(
    select(menages, id_menage, structure, richesse),
    by = "id_menage"
  ) %>%
  copy_labels_from(enfants_original) %>%
  copy_labels_from(femmes) %>%
  copy_labels_from(menages)
```

Les variables catégorielles sont pour l'heure sous formes de vecteurs labellisés. Or, dans un modèle, il est impératif de les convertir en facteurs pour qu'elles soient bien traitées comme des variables catégorielles (autrement elles seraient traitées comme des variables continues). On aura donc recours à la fonction `to_factor` de l'extension `labelled`.

```
R> enfants <- enfants %>%
  mutate(sexe = to_factor(sexe), richesse = to_factor(richesse))
```

Regardons plus attentivement, la variable `structure`.

```
R> freq(enfants$structure)
```

Tout d'abord, la modalité «pas d'adulte» n'est pas représentée dans l'échantillon. On aura donc recours à l'argument `drop_unused_labels` pour ne pas conserver cette modalité. Par ailleurs, nous considérons que la situation familiale à partir de laquelle nous voudrons comparer les autres dans notre modèle, donc celle qui doit être considérée comme la modalité de référence, est celle du ménage nucléaire. Cette modalité («deux adultes de sexe opposé») n'étant pas la première, nous aurons recours à la fonction `relevel` {data-pkg = “stats”}.

```
R> enfants <- enfants %>%
  mutate(structure = relevel(
    to_factor(structure, drop_unused_labels = TRUE),
    "deux adultes de sexe opposé"
  ))
```

Regardons la variable `educ`.

```
R> freq(enfants$educ)
```

La modalité «supérieur» est peu représentée dans notre échantillon. Nous allons la fusionner avec la modalité «secondaire» (voir la section Regrouper les modalités d'une variable, page 213 du chapitre Recodage, page 197).

```
R> enfants <- enfants %>%
  mutate(educ2 = ifelse(educ == 3, 2, educ)) %>%
  set_value_labels(educ2 = c(
    aucun = 0,
    primaire = 1,
    "secondaire ou plus" = 2
  )) %>%
  mutate(educ2 = to_factor(educ2))
freq(enfants$educ2)
```

Calculons maintenant l'âge de la mère à la naissance de l'enfant (voir le chapitre Caluler un âge, page 1103) et découpons le en groupes d'âges (voir la section Découper une variable numérique en classes, page 209 du chapitre Recodage, page 197).

```
R> enfants <- enfants %>%
  mutate(
    age_mere_naissance = time_length(
      interval(date_naissance_mere, date_naissance),
      unit = "years"
    ),
    gpage_mere_naissance = cut(
      age_mere_naissance,
      include.lowest = TRUE, right = FALSE,
      breaks=c(13, 20, 30, 50)
    )
  )

  levels(enfants$gpage_mere_naissance) <- c(
    "19 ou moins", "20-29", "30 et plus"
  )
  enfants$gpage_mere_naissance <- relevel(enfants$gpage_mere_naissance, "20-29")
freq(enfants$gpage_mere_naissance)
```

Reste à calculer si le rang de naissance de l'enfant est supérieur au nombre idéal d'enfants tel que défini par la mère. On aura recours à la fonction `rank` appliquée par groupe (ici calculé séparément pour chaque mère). L'argument `ties.method` permet d'indiquer comment gérer les égalités (ici les naissances multiples, e.g. les jumeaux). Comme nous voulons comparer le rang de l'enfant au nombre idéal d'enfants, nous allons retenir la méthode "`max`" pour obtenir, dans le cas présent, le nombre total d'enfants déjà nés<sup>3</sup>. Avant de calculer un rang, il est impératif de trier préalablement le tableau (voir le chapitre Tris,

page 257).

```
R> enfants <- enfants %>%
  arrange(id_femme, date_naissance) %>%
  group_by(id_femme) %>%
  mutate(
    rang = rank(date_naissance, ties.method = "max"),
    rang_apres_ideal = ifelse(rang > as.integer(nb_enf_ideal), "oui", "non"),
    rang_apres_ideal = factor(rang_apres_ideal, levels = c("non", "oui"))
  )
```

## Kaplan-Meier

La courbe de survie de Kaplan-Meier s’obtient avec la fonction `survfit` de l’extension `survival`.

```
R> library(survival)
km_global <- survfit(Surv(time, deces) ~ 1, data = enfants)
km_global
```

```
Call: survfit(formula = Surv(time, deces) ~ 1, data = enfants)

n  events  median 0.95LCL 0.95UCL
1584      142       NA       NA       NA
```

Pour la représenter, on pourra avoir recours à la fonction `ggsurvplot` de l’extension `survminer`.

---

3. Ici, pour plus de simplicité, nous n’avons pas pris en compte les décès éventuels des enfants de rang inférieur avant la naissance considérée.

```
R> library(survminer, quietly = TRUE)
ggsurvplot(km_global)
```

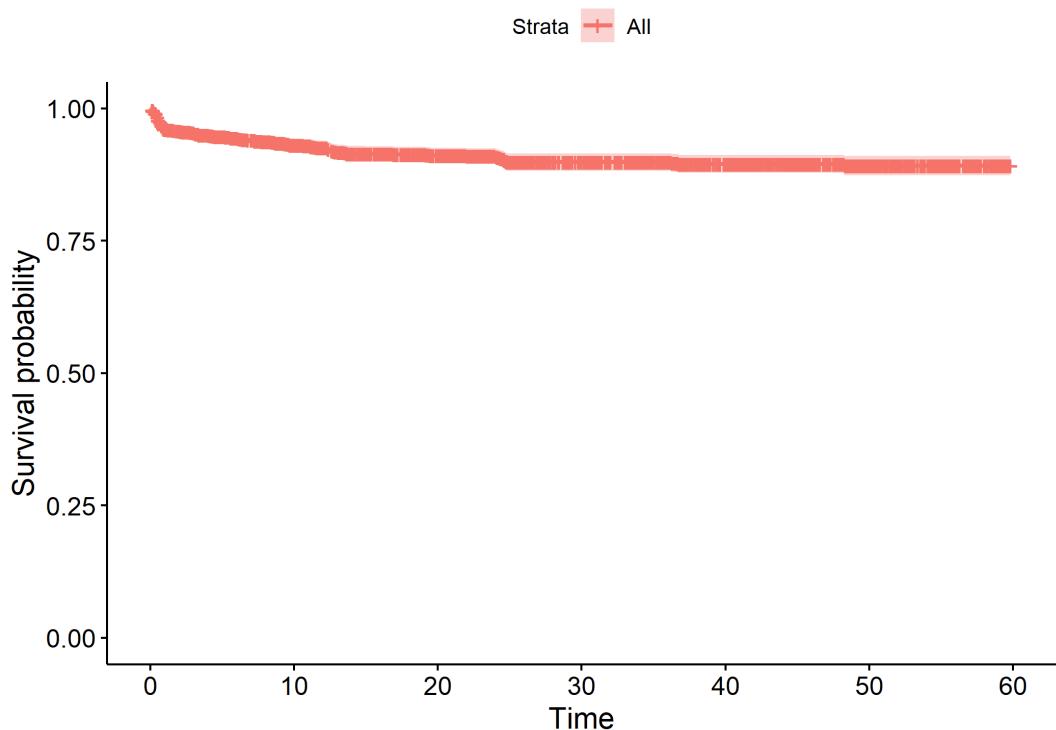


Figure 1. Courbe de survie de Kaplan-Meier

On peut facilement représenter à la place la courbe cumulée des évènements (l'inverse de la courbe de survie) et la table des effectifs en fonction du temps.

```
R> ggsurvplot(km_global, fun = "event", risk.table = TRUE, surv.scale = "per cent")
```

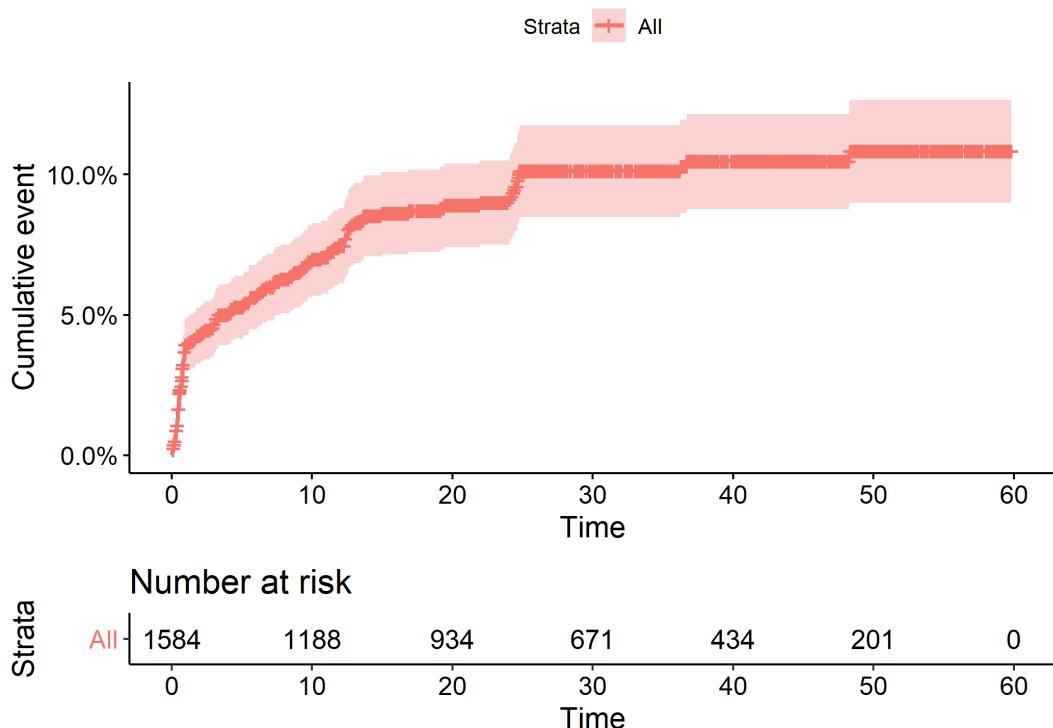


Figure 2. Courbe cumulée des évènements et table des effectifs

Pour comparer deux groupes (ici les filles et les garçons), il suffit d'indiquer la variable de comparaison à `survfit`.

```
R> km_sexe <- survfit(Surv(time, deces) ~ sexe, data = enfants)
km_sexe
```

```
Call: survfit(formula = Surv(time, deces) ~ sexe, data = enfants)

      n  events median 0.95LCL 0.95UCL
sexe=masculin 762      94      NA      NA      NA
sexe=féminin  822      48      NA      NA      NA
```

La fonction `survdiff` permet de calculer le test du logrank afin de comparer des courbes de survie. La mortalité infanto-juvénile diffère-t-elle significativement selon le sexe de l'enfant ?

```
R> survdiff(Surv(time, deces) ~ sexe, data = enfants)
```

Call:

```
survdiff(formula = Surv(time, deces) ~ sexe, data = enfants)
```

	N	Observed	Expected	(O-E)^2/E	(O-E)^2/V
sexe=masculin	762	94	66.2	11.6	21.8
sexe=féminin	822	48	75.8	10.2	21.8

Chisq= 21.8 on 1 degrees of freedom, p= 3e-06

Une fois encore, on aura recours à `ggsurvplot` pour représenter les courbes de survie.

```
R> ggsurvplot(km_sexe, conf.int = TRUE, risk.table = TRUE, pval = TRUE, data = enfants)
```

Warning: Vectorized input to `element\_text()` is not officially supported.  
Results may be unexpected or may change in future versions of ggplot2.

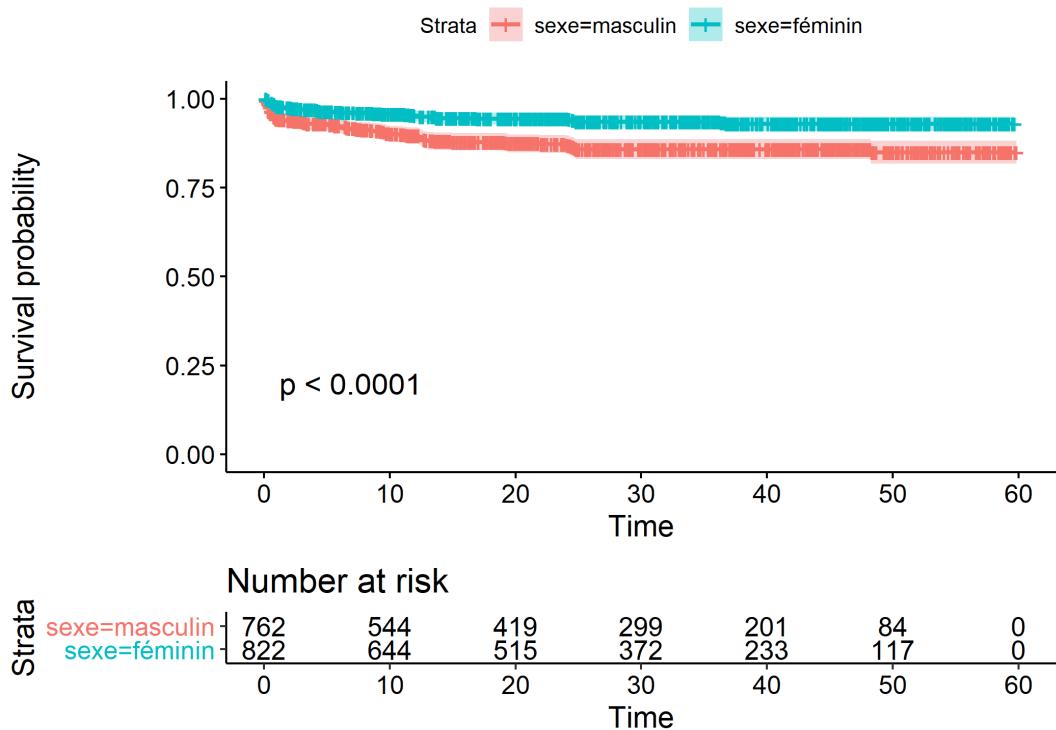


Figure 3. Courbes de Kaplan-Meier selon le sexe

## Modèle de Cox

Un modèle de Cox se calcule aisément avec `coxph {survival}`.

```
R> mod1 <- coxph(
  Surv(time, deces) ~ sexe + milieu + richesse +
  structure + educ2 + gpage_mere_naissance + rang_apres_ideal,
  data = enfants
)
mod1
```

Call:

```
coxph(formula = Surv(time, deces) ~ sexe + milieu + richesse +
  structure + educ2 + gpage_mere_naissance + rang_apres_ideal,
  data = enfants)
```

	coef
sexeféminin	-0.809568
milieu	0.656241
richessepauvre	-0.082223
richessemoyen	0.318645
richesseriche	0.353483
richesstrès riche	0.464590
structureun adulte	-0.150231
structuredeux adultes de même sexe	0.604592
structuretrois adultes ou plus avec lien de parenté	0.049430
structureadultes sans lien de parenté	-0.131369
educ2 primaire	-0.030251
educ2 secondaire ou plus	-0.203903
gpage_mere_naissance19 ou moins	-0.310248
gpage_mere_naissance30 et plus	-0.002586
rang_apres_idealoui	1.355106
	exp(coef)
sexeféminin	0.445050
milieu	1.927533
richessepauvre	0.921066
richessemoyen	1.375263
richesseriche	1.424019
richesstrès riche	1.591361
structureun adulte	0.860509
structuredeux adultes de même sexe	1.830506
structuretrois adultes ou plus avec lien de parenté	1.050672
structureadultes sans lien de parenté	0.876895
educ2 primaire	0.970202
educ2 secondaire ou plus	0.815541
gpage_mere_naissance19 ou moins	0.733265
gpage_mere_naissance30 et plus	0.997417
rang_apres_idealoui	3.877173
	se(coef)
sexeféminin	0.177806

milieu	0.269928
richessepauvre	0.250417
richessemoyen	0.247868
richesseriche	0.298425
richesstrès riche	0.428583
structureun adulte	0.600329
structuredeux adultes de même sexe	0.376445
structuretrois adultes ou plus avec lien de parenté	0.196666
structureadultes sans lien de parenté	0.305478
educ2 primaire	0.205751
educ2 secondaire ou plus	0.366889
gpage_mere_naissance19 ou moins	0.268062
gpage_mere_naissance30 et plus	0.191557
rang_apres_idealoui	0.602401
	<i>z</i>
sexeféminin	-4.553
milieu	2.431
richessepauvre	-0.328
richessemoyen	1.286
richesseriche	1.184
richesstrès riche	1.084
structureun adulte	-0.250
structuredeux adultes de même sexe	1.606
structuretrois adultes ou plus avec lien de parenté	0.251
structureadultes sans lien de parenté	-0.430
educ2 primaire	-0.147
educ2 secondaire ou plus	-0.556
gpage_mere_naissance19 ou moins	-1.157
gpage_mere_naissance30 et plus	-0.014
rang_apres_idealoui	2.250
	<i>p</i>
sexeféminin	5.29e-06
milieu	0.0151
richessepauvre	0.7427
richessemoyen	0.1986
richesseriche	0.2362
richesstrès riche	0.2784
structureun adulte	0.8024
structuredeux adultes de même sexe	0.1083
structuretrois adultes ou plus avec lien de parenté	0.8016
structureadultes sans lien de parenté	0.6672
educ2 primaire	0.8831
educ2 secondaire ou plus	0.5784
gpage_mere_naissance19 ou moins	0.2471
gpage_mere_naissance30 et plus	0.9892
rang_apres_idealoui	0.0245

```
Likelihood ratio test=38.16  on 15 df, p=0.0008546
n= 1584, number of events= 142
```

De nombreuses variables ne sont pas significatives. Voyons si nous pouvons, avec la fonction `step`, améliorer notre modèle par minimisation de l'AIC ou *Akaike Information Criterion* (voir la section Sélection de modèles, page 0 du chapitre sur la Régression logistique, page 507).

```
R> mod2 <- step(mod1)
```

```
Start: AIC=2027.07
Surv(time, deces) ~ sexe + milieu + richesse + structure + educ2 +
  gpage_mere_naissance + rang_apres_ideal

          Df      AIC
- structure        4 2022.0
- richesse         4 2022.7
- educ2            2 2023.4
- gpage_mere_naissance  2 2024.6
<none>              2027.1
- rang_apres_ideal    1 2028.6
- milieu            1 2031.3
- sexe              1 2047.1

Step: AIC=2022.01
Surv(time, deces) ~ sexe + milieu + richesse + educ2 + gpage_mere_naissance +
  rang_apres_ideal

          Df      AIC
- richesse         4 2017.0
- educ2            2 2018.2
- gpage_mere_naissance  2 2019.5
<none>              2022.0
- rang_apres_ideal    1 2023.4
- milieu            1 2025.6
- sexe              1 2042.2

Step: AIC=2017
Surv(time, deces) ~ sexe + milieu + educ2 + gpage_mere_naissance +
  rang_apres_ideal

          Df      AIC
- educ2            2 2013.3
- gpage_mere_naissance  2 2014.8
<none>              2017.0
- rang_apres_ideal    1 2018.0
```

```
- milieu           1 2018.8
- sexe            1 2037.6

Step: AIC=2013.28
Surv(time, deces) ~ sexe + milieu + gpage_mere_naissance + rang_apres_ideal

          Df   AIC
- gpage_mere_naissance 2 2011.2
<none>             2013.3
- rang_apres_ideal     1 2014.3
- milieu              1 2015.7
- sexe                1 2033.8

Step: AIC=2011.17
Surv(time, deces) ~ sexe + milieu + rang_apres_ideal

          Df   AIC
<none>             2011.2
- rang_apres_ideal   1 2012.3
- milieu              1 2013.9
- sexe                1 2031.6
```

On peut obtenir facilement les coefficients du modèle avec l'excellente fonction `tidy` de l'extension `broom`. Ne pas oublier de préciser `exponentiate = TRUE`. En effet, dans le cas d'un modèle de Cox, l'exponentiel des coefficients corresponds au ratio des risques instantannés ou *hazard ratio (HR)* en anglais.

```
R> library(broom, quietly = TRUE)
tidy(mod2, exponentiate = TRUE)
```

Pour représenter ces rapports de risque, on peut ici encore avoir recours à la fonction `ggcoef_model` de l'extension `GGally`.

```
R> library(GGally, quietly = TRUE)
```

```
Registered S3 method overwritten by 'GGally':
  method from
  +.gg   ggplot2
```

```
R> ggcoef_model(mod2, exponentiate = TRUE)
```

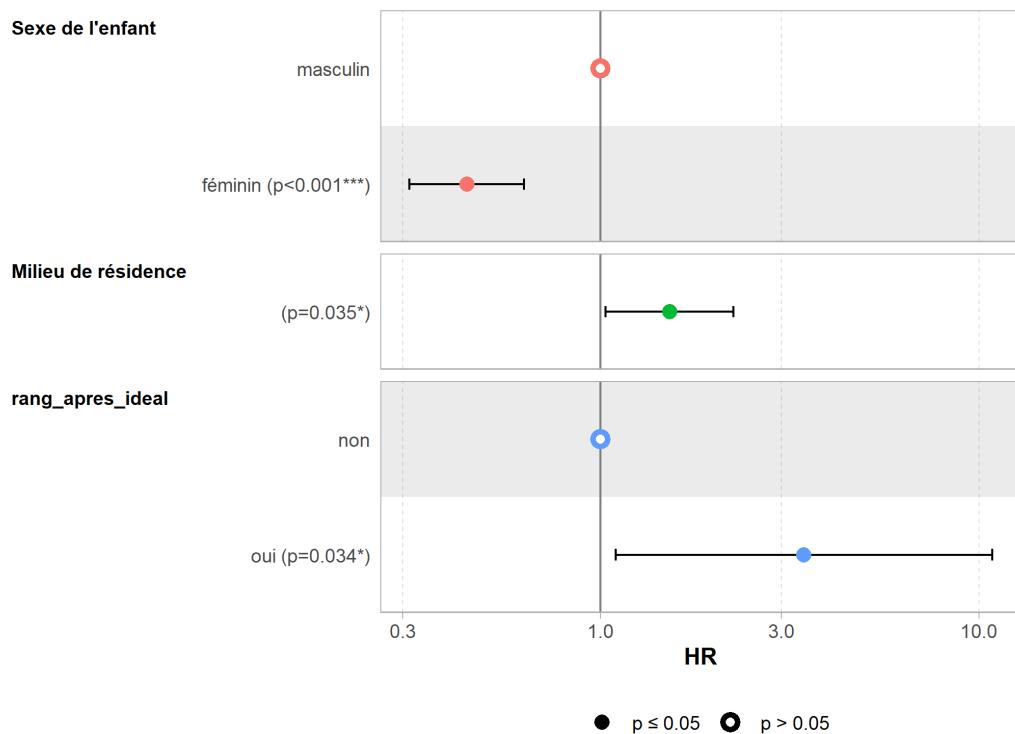


Figure 4. Coefficients du modèle avec ggcoef\_model

L'extension `survminer` fournit également une fonction `ggforest` qui permet de représenter de manière plus esthétique et complète les coefficients d'un modèle de Cox.

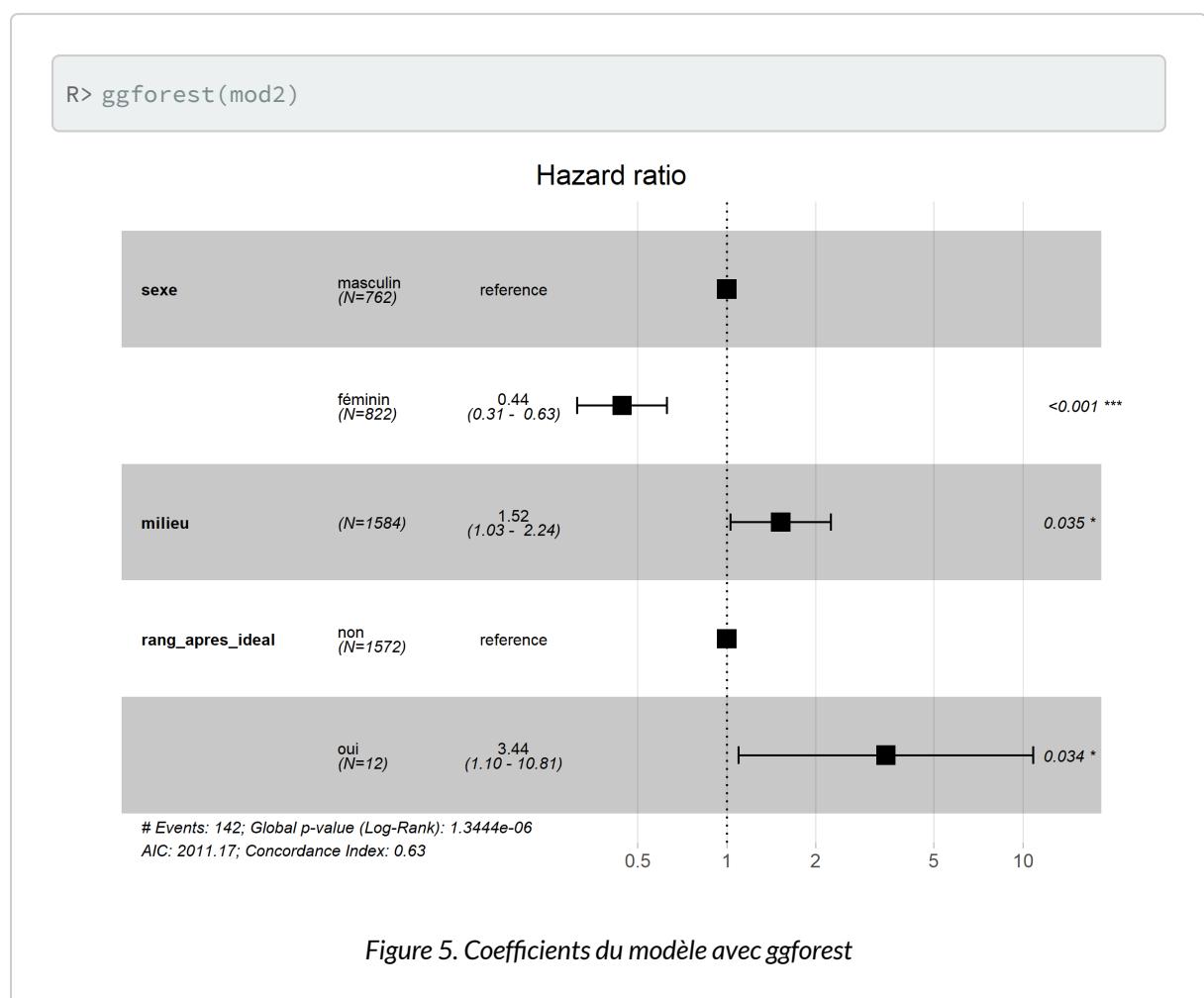


Figure 5. Coefficients du modèle avec ggforest

## Vérification de la validité du modèle

Un modèle de Cox n'est valable que sous l'hypothèse de la proportionnalité des risques relatifs. Selon cette hypothèse les résidus de Schoenfeld ne dépendent pas du temps. Cette hypothèse peut être testée avec la fonction `cox.zph`.

```
R> test <- cox.zph(mod2)
test
```

	chisq	df	p
sexé	0.505	1	0.48
milieu	0.136	1	0.71
rang_apres_ideal	0.163	1	0.69
GLOBAL	0.805	3	0.85

Une valeur de p inférieure à 5 % indique que l'hypothèse n'est pas vérifiée. Il apparaît que p est supérieur à 5 % globalement et pour chaque variable prise individuellement. Notre modèle est donc valide.

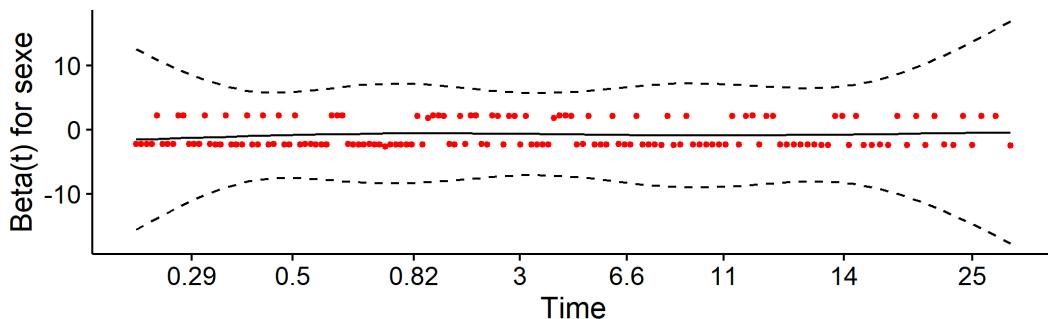
Il est possible de représenter la distribution des résidus de Schoenfeld à l'aide de `ggcoxph` de l'extension `survminer`, afin de voir si leur répartition change au cours du temps.

```
R> ggcoxph(test)
```

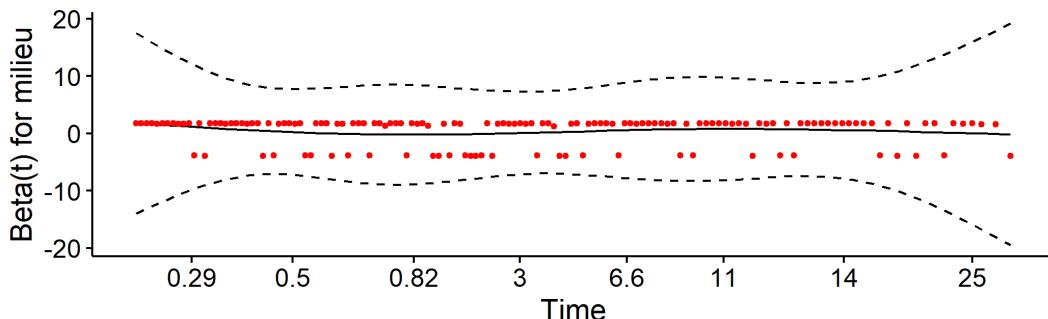
```
Warning in regularize.values(x, y, ties, missing(ties),  
na.rm = na.rm): collapsing to unique 'x' values
```

Global Schoenfeld Test p: 0.8484

Schoenfeld Individual Test p: 0.4773



Schoenfeld Individual Test p: 0.7127



Schoenfeld Individual Test p: 0.6868

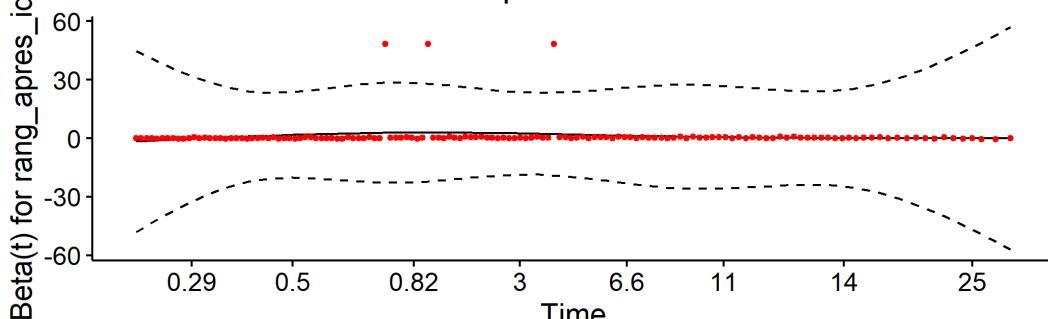


Figure 6. Résidus de Schoenfeld

## Données pondérées

Si vous utilisez des données pondérées avec un plan d'échantillonnage complexe (voir le chapitre dédié, page 501), vous pouvez utiliser les fonctions suivantes de l'extension `survey` :

- `svykm` pour estimer une courbe de survie de Kaplan-Meier ;
- `svycoxph` pour un modèle de Cox.

Dans les deux cas, pensez à ajouter l'option `se = TRUE` pour que les erreurs standards soient calculées (et que les intervalles de confiance puissent être générés).



# Analyse de séquences

L'analyse de séquences . . . . .	679
Charger TraMineR et récupérer les données . . . . .	681
Appariement optimal et classification . . . . .	685
Représentations graphiques . . . . .	689
Distribution de la typologie . . . . .	702
Pour aller plus loin . . . . .	703
Bibliographie . . . . .	703

## NOTE

La version originale de ce chapitre est une reprise, avec l'aimable autorisation de son auteur, d'un article de Nicolas Robette intitulé *L'analyse de séquences : une introduction avec le logiciel R et le package TraMineR* et publié sur le blog Quanti (<http://quanti.hypotheses.org/686/>).

Depuis les années 1980, l'étude quantitative des trajectoires biographiques (*life course analysis*) a pris une ampleur considérable dans le champ des sciences sociales. Les collectes de données micro-individuelles longitudinales se sont développées, principalement sous la forme de panels ou d'enquêtes rétrospectives. Parallèlement à cette multiplication des données disponibles, la méthodologie statistique a connu de profondes évolutions. L'analyse des biographies (*event history analysis*) — qui ajoute une dimension diachronique aux modèles économétriques mainstream — s'est rapidement imposée comme l'approche dominante : il s'agit de modéliser la durée des situations ou le risque d'occurrence des événements.

## L'analyse de séquences

Cependant, ces dernières années ont vu la diffusion d'un large corpus de méthodes descriptives d'analyse de séquences, au sein desquelles l'appariement optimal (*optimal matching*) occupe une place centrale<sup>1</sup>.

---

1. Pour une analyse des conditions sociales de la diffusion de l'analyse de séquences dans le champ des sciences sociales, voir Robette, 2012.

L'objectif principal de ces méthodes est d'identifier — dans la diversité d'un corpus de séquences constituées de séries d'états successifs — les régularités, les ressemblances, puis le plus souvent de construire des typologies de « séquences-types ». L'analyse de séquences constitue donc un moyen de décrire mais aussi de mieux comprendre le déroulement de divers processus.

La majeure partie des applications de l'analyse de séquences traite de trajectoires biographiques ou de carrières professionnelles. Dans ces cas, chaque trajectoire ou chaque carrière est décrite par une séquence, autrement dit par une suite chronologiquement ordonnée de « moments » élémentaires, chaque moment correspondant à un « état » déterminé de la trajectoire (par exemple, pour les carrières professionnelles : être en emploi, au chômage ou en inactivité). Mais on peut bien sûr imaginer des types de séquences plus originaux : Andrew Abbott<sup>2</sup>, le sociologue américain qui a introduit l'*optimal matching* dans les sciences scientifiques ou des séquences de pas de danses traditionnelles.

En France, les premiers travaux utilisant l'appariement optimal sont ceux de Claire Lemercier<sup>3</sup> sur les carrières des membres des institutions consulaires parisiennes au xix<sup>e</sup> siècle (Lemercier, 2005), et de Laurent Lesnard<sup>4</sup> sur les emplois du temps (Lesnard, 2008). Mais dès les années 1980, les chercheurs du Céreq construisaient des typologies de trajectoires d'insertion à l'aide des méthodes d'analyse des données « à la française » (analyse des correspondances, etc.)<sup>5</sup>. Au final, on dénombre maintenant plus d'une centaine d'articles de sciences sociales contenant ou discutant des techniques empruntées à l'analyse de séquences.

Pour une présentation des différentes méthodes d'analyse de séquences disponibles et de leur mise en œuvre pratique, il existe un petit manuel en français, publié en 2011 dernière aux éditions du Ceped (collection « Les clefs pour »<sup>6</sup>) et disponible en pdf<sup>7</sup> (Robette, 2011). De plus, un article récemment publié dans le *Bulletin de Méthodologie Sociologique* compare de manière systématique les résultats obtenus par les principales méthodes d'analyse de séquences (Robette & Bry, 2012). La conclusion en est qu'avec des données empiriques aussi structurées que celles que l'on utilise en sciences sociales, l'approche est robuste, c'est-à-dire qu'un changement de méthode aura peu d'influence sur les principaux résultats. Cependant, l'article tente aussi de décrire les spécificités de chaque méthode et les différences marginales qu'elles font apparaître, afin de permettre aux chercheurs de mieux adapter leurs choix méthodologiques à leur question de recherche.

Afin d'illustrer la démarche de l'analyse de séquences, nous allons procéder ici à la description « pas à pas » d'un corpus de carrières professionnelles, issues de l'enquête *Biographies et entourage* (Ined, 2000)<sup>8</sup>.

---

2. <http://home.uchicago.edu/~aabott/>

3. <http://lemercier.ouvaton.org/document.php?id=62>

4. [http://laurent.lesnard.free.fr/article.php3?id\\_article=22](http://laurent.lesnard.free.fr/article.php3?id_article=22)

5. Voir par exemple l'article d'Yvette Grelet (2002).

6. <http://www.ceped.org/?rubrique57>

7. [http://nicolas.robette.free.fr/Docs/Robette2011\\_Manuel\\_TypoTraj.pdf](http://nicolas.robette.free.fr/Docs/Robette2011_Manuel_TypoTraj.pdf)

8. Pour une analyse plus poussée de ces données, avec deux méthodes différentes, voir Robette & Thibault, 2008. Pour

Et pour ce faire, on va utiliser le logiciel **R**, qui propose la solution actuellement la plus complète et la plus puissante en matière d'analyse de séquences. Les méthodes d'analyse de séquences par analyses factorielles ou de correspondances ne nécessitent pas de logiciel spécifique : tous les logiciels de statistiques généralistes peuvent être utilisés (**SAS**, **SPSS**, **Stata**, **R**, etc.). En revanche, il n'existe pas de fonctions pour l'appariement optimal dans **SAS** ou **SPSS**. Certains logiciels gratuits implémentent l'appariement optimal (comme **Chesa**<sup>9</sup> ou **TDA**<sup>10</sup>) mais il faut alors recourir à d'autres programmes pour dérouler l'ensemble de l'analyse (classification, représentation graphique). **Stata** propose le module **sq**<sup>11</sup>, qui dispose d'un éventail de fonctions intéressantes. Mais c'est **R** et le package **TraMineR**<sup>12</sup>, développé par des collègues de l'Université de Genève (Gabadinho et al, 2011), qui fournit la solution la plus complète et la plus puissante à ce jour : on y trouve l'appariement optimal mais aussi d'autres algorithmes alternatifs, ainsi que de nombreuses fonctions de description des séquences et de représentation graphique.

## Charger TraMineR et récupérer les données

Tout d'abord, à quoi ressemblent nos données ? On a reconstruit à partir de l'enquête les carrières de 1000 hommes. Pour chacune, on connaît la position professionnelle chaque année, de l'âge de 14 ans jusqu'à 50 ans. Cette position est codée de la manière suivante : les codes 1 à 6 correspondent aux groupes socioprofessionnels de la nomenclature des PCS de l'INSEE 13 (agriculteurs exploitants ; artisans, commerçants et chefs d'entreprise ; cadres et professions intellectuelles supérieures ; professions intermédiaires ; employés ; ouvriers) ; on y a ajouté « études » (code 7), « inactivité » (code 8) et « service militaire » (code 9). Le fichier de données comporte une ligne par individu et une colonne par année : la variable *csp1* correspond à la position à 14 ans, la variable *csp2* à la position à 15 ans, etc. Par ailleurs, les enquêtés étant tous nés entre 1930 et 1950, on ajoute à notre base une variable « génération » à trois modalités, prenant les valeurs suivantes : 1="1930-1938" ; 2="1939-1945" ; 3="1946-1950". Au final, la base est constituée de 500 lignes et de  $37 + 1 = 38$  colonnes et se présente sous la forme d'un fichier texte au format **csv** (téléchargeable à <http://larmarange.github.io/analyse-R/data/trajpro.csv>).

Une fois **R** ouvert, on commence par installer les extensions nécessaires à ce programme (opération à ne réaliser que lors de leur première utilisation) et par les charger en mémoire. L'extension **TraMineR** propose de nombreuses fonctions pour l'analyse de séquences. L'extension **cluster** comprend un certain nombre de méthodes de classification automatique<sup>13</sup>.

---

une présentation de l'enquête, voir Lelièvre & Vivier, 2001.

9. <http://home.fsw.vu.nl/ch.elzinga/>

10. <http://steinhaus.stat.ruhr-uni-bochum.de/tda.html>

11. <http://www.stata-journal.com/article.html?article=st0111>

12. <http://mephisto.unige.ch/traminer/>

13. Pour une présentation plus détaillée, voir le chapitre sur la classification ascendante hiérarchique (CAH), page 595.

```
R> library(TraMineR)
```

```
TraMineR stable version 2.2-1 (Built: 2020-11-02)
```

```
Website: http://traminer.unige.ch
```

```
Please type 'citation("TraMineR")' for citation information.
```

```
R> library(cluster)
```

On importe ensuite les données, on recode la variable « génération » pour lui donner des étiquettes plus explicites. On jette également un coup d’œil à la structure du tableau de données :

```
-- Column specification -----
cols(
  .default = col_double()
)
# Use `spec()` for the full column specifications.
```

```
R> donnees <- read.csv("http://lamarange.github.io/analyse-R/data/trajpro.csv",
  header = T)
```

```
R> donnees$generation <- factor(donnees$generation, labels = c("1930-38", "1939-45",
  "1946-50"))
str(donnees)
```

```
tibble [1,000 x 38] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
$ csp1      : num [1:1000] 1 7 6 7 7 6 7 7 7 6 ...
$ csp2      : num [1:1000] 1 7 6 7 7 6 7 7 6 6 ...
$ csp3      : num [1:1000] 1 7 6 6 7 6 7 7 6 6 ...
$ csp4      : num [1:1000] 1 7 6 6 7 6 7 7 6 6 ...
$ csp5      : num [1:1000] 1 7 6 6 7 6 7 7 6 6 ...
$ csp6      : num [1:1000] 1 7 6 6 7 6 9 7 6 6 ...
$ csp7      : num [1:1000] 6 9 6 6 7 6 9 7 9 6 ...
$ csp8      : num [1:1000] 6 9 9 6 7 6 9 7 4 6 ...
$ csp9      : num [1:1000] 6 6 9 6 7 6 9 3 4 9 ...
$ csp10     : num [1:1000] 6 6 9 6 7 6 4 3 4 9 ...
$ csp11     : num [1:1000] 6 6 6 6 3 6 4 3 4 6 ...
$ csp12     : num [1:1000] 6 6 6 6 3 6 4 3 4 6 ...
$ csp13     : num [1:1000] 6 6 6 6 3 6 4 3 4 6 ...
$ csp14     : num [1:1000] 6 4 6 6 3 6 4 3 4 6 ...
```

```

$ csp15      : num [1:1000] 6 4 6 6 3 6 4 3 4 6 ...
$ csp16      : num [1:1000] 6 4 6 6 3 6 6 3 4 6 ...
$ csp17      : num [1:1000] 6 4 6 6 3 6 6 3 4 6 ...
$ csp18      : num [1:1000] 6 4 6 6 3 6 6 3 4 6 ...
$ csp19      : num [1:1000] 6 4 6 6 3 6 6 3 4 6 ...
$ csp20      : num [1:1000] 6 4 6 6 3 6 6 3 4 6 ...
$ csp21      : num [1:1000] 6 4 6 6 6 6 6 3 4 6 ...
$ csp22      : num [1:1000] 6 4 6 6 6 6 6 3 4 4 ...
$ csp23      : num [1:1000] 6 4 6 6 6 6 6 3 4 4 ...
$ csp24      : num [1:1000] 6 6 6 6 5 6 6 3 4 4 ...
$ csp25      : num [1:1000] 6 6 6 6 5 6 6 3 4 4 ...
$ csp26      : num [1:1000] 6 6 6 6 5 6 6 3 4 4 ...
$ csp27      : num [1:1000] 6 6 6 6 5 6 6 3 4 4 ...
$ csp28      : num [1:1000] 6 6 6 6 5 6 6 3 4 4 ...
$ csp29      : num [1:1000] 6 6 6 6 5 6 6 3 4 4 ...
$ csp30      : num [1:1000] 4 6 6 6 5 6 6 3 4 4 ...
$ csp31      : num [1:1000] 4 6 6 6 5 6 6 3 4 4 ...
$ csp32      : num [1:1000] 4 6 6 6 5 6 6 3 4 4 ...
$ csp33      : num [1:1000] 4 6 6 6 5 6 6 3 4 4 ...
$ csp34      : num [1:1000] 4 6 6 6 5 6 6 3 4 4 ...
$ csp35      : num [1:1000] 4 6 6 6 5 6 6 3 4 4 ...
$ csp36      : num [1:1000] 4 6 6 6 5 6 6 3 4 4 ...
$ csp37      : num [1:1000] 4 6 6 6 5 6 6 3 4 4 ...
$ generation: Factor w/ 3 levels "1930-38","1939-45",...: 2 1 1 3 2 3 1 1 2 1
...
- attr(*, "spec")=
.. cols(
..   csp1 = col_double(),
..   csp2 = col_double(),
..   csp3 = col_double(),
..   csp4 = col_double(),
..   csp5 = col_double(),
..   csp6 = col_double(),
..   csp7 = col_double(),
..   csp8 = col_double(),
..   csp9 = col_double(),
..   csp10 = col_double(),
..   csp11 = col_double(),
..   csp12 = col_double(),
..   csp13 = col_double(),
..   csp14 = col_double(),
..   csp15 = col_double(),
..   csp16 = col_double(),
..   csp17 = col_double(),
..   csp18 = col_double(),
..   csp19 = col_double(),
..   csp20 = col_double(),

```

```

..  csp21 = col_double(),
..  csp22 = col_double(),
..  csp23 = col_double(),
..  csp24 = col_double(),
..  csp25 = col_double(),
..  csp26 = col_double(),
..  csp27 = col_double(),
..  csp28 = col_double(),
..  csp29 = col_double(),
..  csp30 = col_double(),
..  csp31 = col_double(),
..  csp32 = col_double(),
..  csp33 = col_double(),
..  csp34 = col_double(),
..  csp35 = col_double(),
..  csp36 = col_double(),
..  csp37 = col_double(),
..  generation = col_double()
.. )

```

On a bien 1000 observations et 38 variables. On définit maintenant des *labels* pour les différents états qui composent les séquences et on crée un objet « séquence » avec `seqdef` :

```
R> labels <- c("agric", "acce", "cadr", "pint", "empl", "ouvr", "etud", "inact",
  "smil")
seq <- seqdef(donnees[, 1:37], states = labels)
```

[>] state coding:

	[alphabet]	[label]	[long label]
1	1	agric	agric
2	2	acce	acce
3	3	cadr	cadr
4	4	pint	pint
5	5	empl	empl
6	6	ouvr	ouvr
7	7	etud	etud

```

8 8      inact    inact
9 9      smil     smil
[>] 1000 sequences in the data set
[>] min/max sequence length: 37/37

```

## Appariement optimal et classification

Ces étapes préalables achevées, on peut comparer les séquences en calculant les dissimilarités entre paires de séquences. On va ici utiliser la méthode la plus répandue, l'appariement optimal (*optimal matching*). Cette méthode consiste, pour chaque paire de séquences, à compter le nombre minimal de modifications (substitutions, suppressions, insertions) qu'il faut faire subir à l'une des séquences pour obtenir l'autre. On peut considérer que chaque modification est équivalente, mais il est aussi possible de prendre en compte le fait que les « distances » entre les différents états n'ont pas toutes la même « valeur » (par exemple, la distance sociale entre emploi à temps plein et chômage est plus grande qu'entre emploi à temps plein et emploi à temps partiel), en assignant aux différentes modifications des « coûts » distincts. Dans notre exemple, on va créer avec `seqsubm` une « matrice des coûts de substitution » dans laquelle tous les coûts sont constants et égaux à 2<sup>14</sup>:

```
R> couts <- seqsubm(seq, method = "CONSTANT", cval = 2)
```

```
[>] creating 9x9 substitution-cost matrix using 2 as constant value
```

Ensuite, on calcule la matrice de distances entre les séquences (i.e contenant les « dissimilarités » entre les séquences) avec `seqdist`, avec un coût d'insertion/suppression (*indel*) que l'on fixe ici à 1 :

```
R> seq.om <- seqdist(seq, method = "OM", indel = 1, sm = couts)
```

```
[>] 1000 sequences with 9 distinct states
```

```
[>] checking 'sm' (size and triangle inequality)
```

```
[>] 818 distinct sequences
```

---

14. Le fonctionnement de l'algorithme d'appariement optimal – et notamment le choix des coûts – est décrit dans le chapitre 9 du manuel de **TraMineR** (<http://mephisto.unige.ch/pub/TraMineR/doc/TraMineR-Users-Guide.pdf>).

```
[>] min/max sequence lengths: 37/37  
[>] computing distances using the OM metric  
[>] elapsed time: 2.2 secs
```

### IMPORTANT

Ce cas de figure où tous les coûts de substitution sont égaux à 2 et le coût *indel* égal à 1 correspond à un cas particulier d'*optimal matching* que l’on appelle la *Longest Common Subsequence* ou LCS. Elle peut se calculer directement avec `seqdist` de la manière suivante :

```
R> seq.om <- seqdist(seq, method = "LCS")  
  
[>] 1000 sequences with 9 distinct states  
[>] creating a 'sm' with a substitution cost of 2  
[>] creating 9x9 substitution-cost matrix using 2 as constant value  
[>] 818 distinct sequences  
[>] min/max sequence lengths: 37/37  
[>] computing distances using the LCS metric  
[>] elapsed time: 2.07 secs
```

En l’absence d’hypothèses fortes sur les différents statuts auxquels correspond notre alphabet (données hiérarchisées, croisement de différentes dimensions...), nous vous recommandons d’utiliser prioritairement la métrique LCS pour calculer la distance entre les séquences.

On pourra trouver un exemple de matrice de coûts hiérarchisée dans le chapitre sur les trajectoires de soins, page 705.

Cette matrice des distances ou des dissimilarités entre séquences peut ensuite être utilisée pour une classification ascendante hiérarchique (CAH), qui permet de regrouper les séquences en un certain nombre de « classes » en fonction de leur proximité :

```
R> seq.dist <- hclust(as.dist(seq.om), method = "ward.D2")
```

Avec la fonction `plot`, il est possible de tracer l'arbre de la classification (dendrogramme).

```
R> plot(as.dendrogram(seq.dist), leaflab = "none")
```

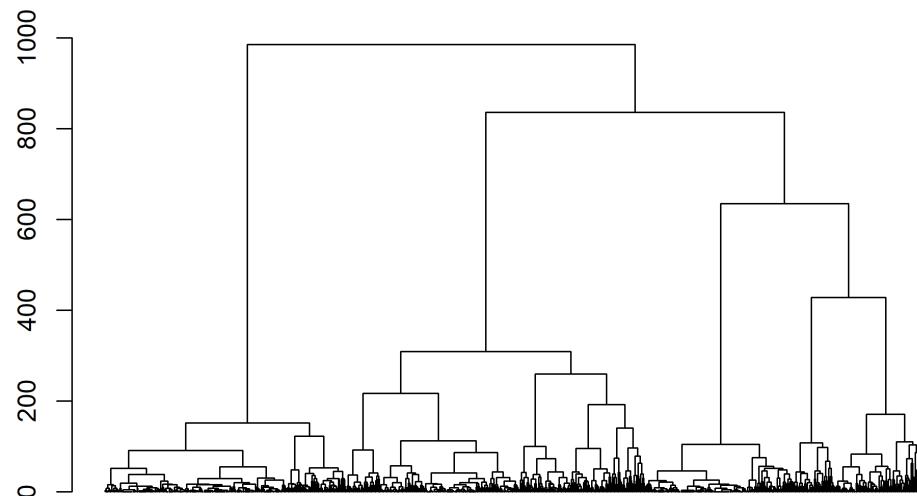


Figure 1. Dendrogramme de la classification des séquences

De même, on peut représenter les sauts d'inertie.

```
R> plot(sort(seq.dist$height, decreasing = TRUE)[1:20], type = "s", xlab = "nb de classes", ylab = "inertie")
```

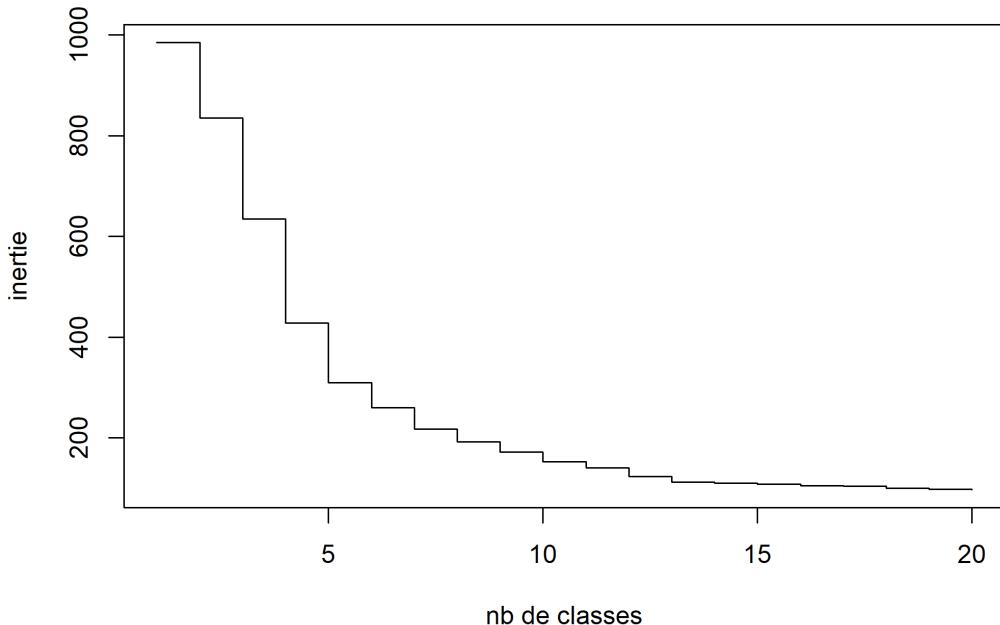


Figure 2. Sauts d'inertie de la classification des séquences

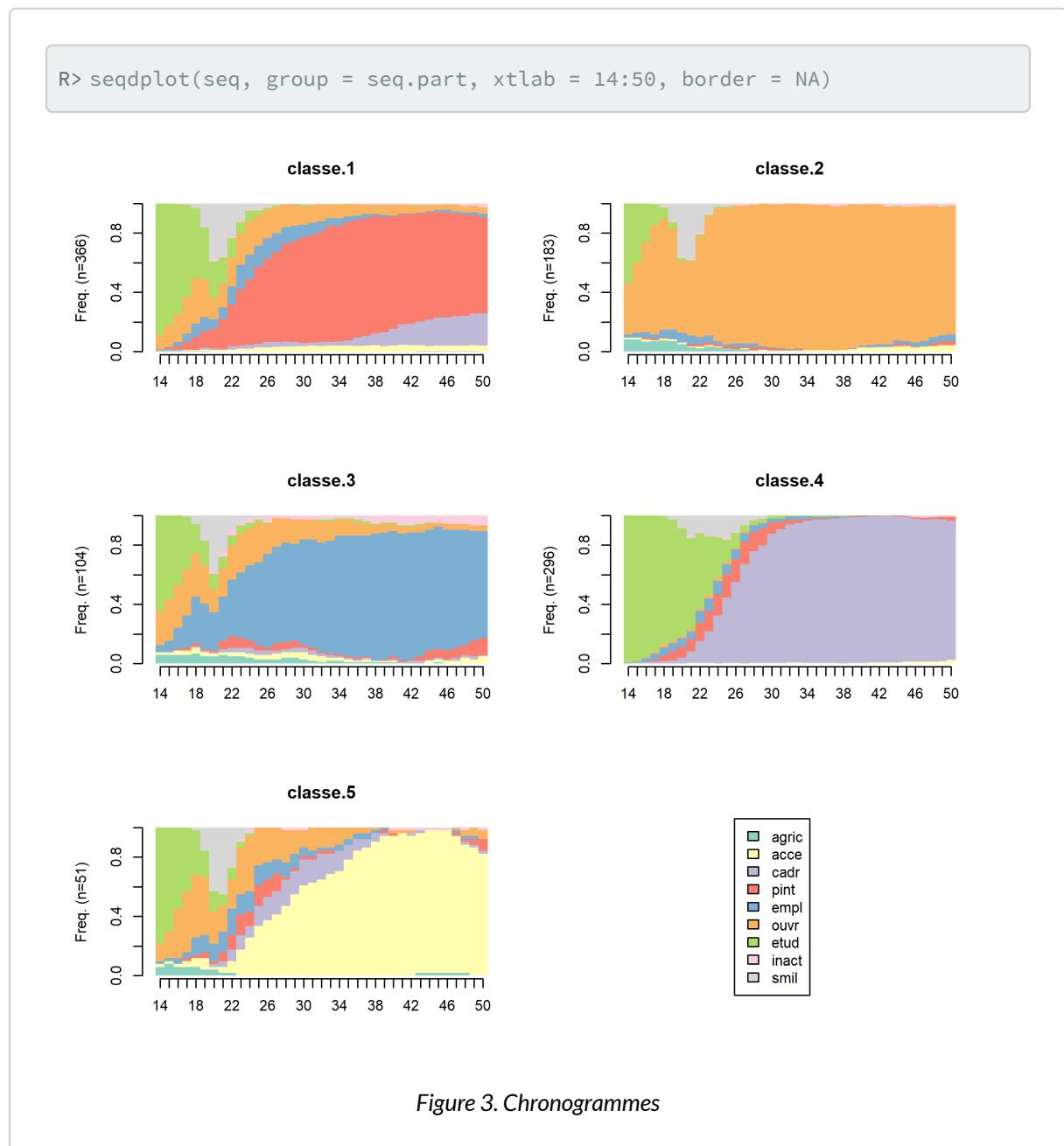
L'observation, sur ce dendrogramme ou sur la courbe des sauts d'inertie, des sauts d'inertie des dernières étapes de la classification peut servir de guide pour déterminer le nombre de classes que l'on va retenir pour la suite des analyses. Une première inflexion dans la courbe des sauts d'inertie apparaît au niveau d'une partition en 5 classes. On voit aussi une seconde inflexion assez nette à 7 classes. Mais il faut garder en tête le fait que ces outils ne sont que des guides, le choix devant avant tout se faire après différents essais, en fonction de l'intérêt des résultats par rapport à la question de recherche et en arbitrant entre exhaustivité et parcimonie.

On fait ici le choix d'une partition en 5 classes :

```
R> nbcl <- 5
seq.part <- cutree(seq.dist, nbcl)
seq.part <- factor(seq.part, labels = paste("classe", 1:nbcl, sep = ".") )
```

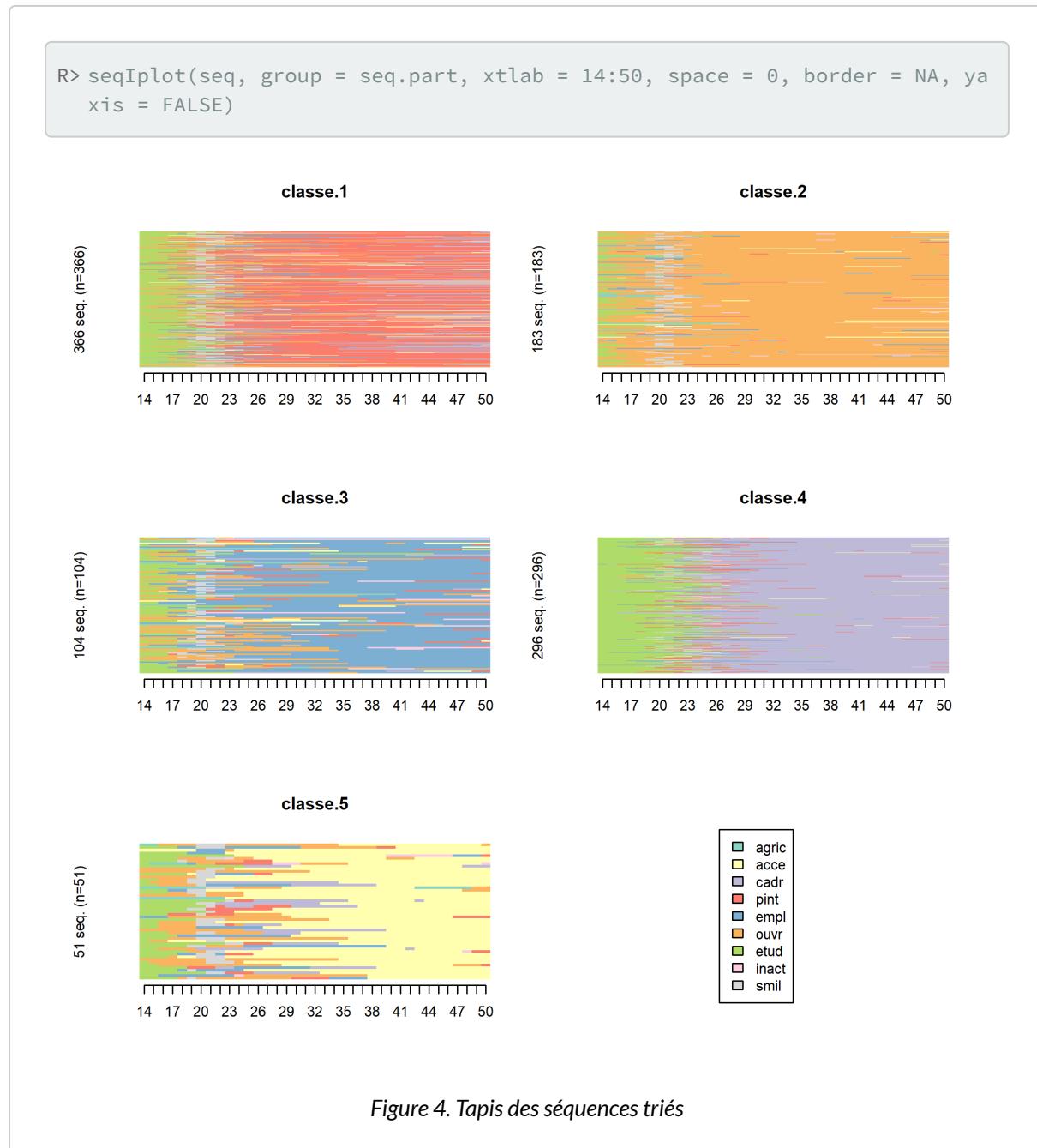
# Représentations graphiques

Pour se faire une première idée de la nature des classes de la typologie, il existe un certain nombre de représentations graphiques. Les chronogrammes (*state distribution plots*) présentent une série de coupes transversales : pour chaque âge, on a les proportions d'individus de la classe dans les différentes situations (agriculteur, étudiant, etc.). Ce graphique s'obtient avec `seqdplot` :



Chacune des classes semble caractérisée par un groupe professionnel principal : profession intermédiaire pour la classe 1, ouvrier pour la 2, employé pour la 3, cadre pour la 4 et indépendant pour la 5. Cependant, on aperçoit aussi des « couches » d'autres couleurs, indiquant que l'ensemble des carrières ne sont probablement pas stables.

Les « tapis » (*index plots*), obtenus avec `seqIplot`, permettent de mieux visualiser la dimension individuelle des séquences. Chaque segment horizontal représente une séquence, découpée en sous-segments correspondant aux différents états successifs qui composent la séquence.



Il est possible de trier les séquences pour rendre les tapis plus lisibles (on trie ici par *multidimensional scaling* à l'aide de la fonction `cmdscale`).

```
R> ordre <- cmdscale(as.dist(seq.om), k = 1)
seqIplot(seq, group = seq.part, sortv = ordre, xlab = 14:50, space = 0,
border = NA, yaxis = FALSE)
```

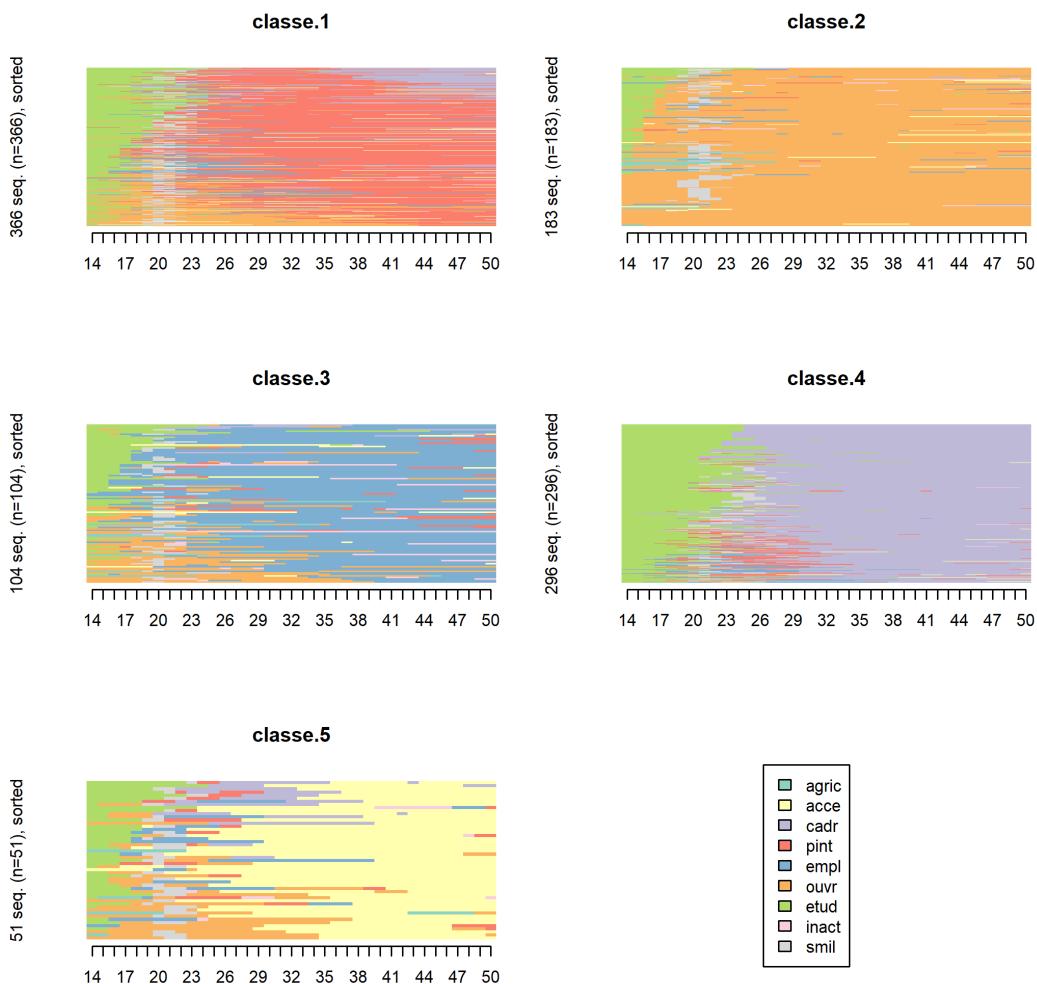


Figure 5. Tapis des séquences triés par multidimensional scaling

On voit mieux apparaître ainsi l'hétérogénéité de certaines classes. Les classes 1, 3 et 4, par exemple, semblent regrouper des carrières relativement stables (respectivement de professions intermédiaires, d'employés et de cadres) et des carrières plus « mobiles » commencées comme ouvrier (classes 1 et 3, en orange) ou comme profession intermédiaire (classe 4, en rouge). De même, la majorité des membres de la

dernière classe commencent leur carrière dans un groupe professionnel distinct de celui qu’ils occuperont par la suite (indépendants). Ces distinctions apparaissent d’ailleurs si on relance le programme avec un nombre plus élevé de classes (en remplaçant le 5 de la ligne `nbcl <- 5` par 7, seconde inflexion de la courbe des sauts d’inertie, et en exécutant de nouveau le programme à partir de cette ligne) : les stables et les mobiles se trouvent alors dans des classes distinctes.

Le package **JLutils**, disponible sur [GitHub](#), propose une fonction `seq_heatmap` permettant de représenter le tapis de l’ensemble des séquences selon l’ordre du dendrogramme.

Pour installer **JLutils**, on aura recours au package **devtools** et à sa fonction `install_github` :

```
R> library(devtools)
    install_github("larmarange/JLutils")
```

On peut ensuite générer le graphique ainsi :

```
R> library(JLutils)
```

```
Loading required package: ggplot2
```

```
Loading required package: plyr
```

```
Loading required package: magrittr
```

```
Registered S3 method overwritten by 'GGally':  
method from  
+.gg   ggplot2
```

```
R> seq_heatmap(seq, seq.dist, labCol = 14:50)
```

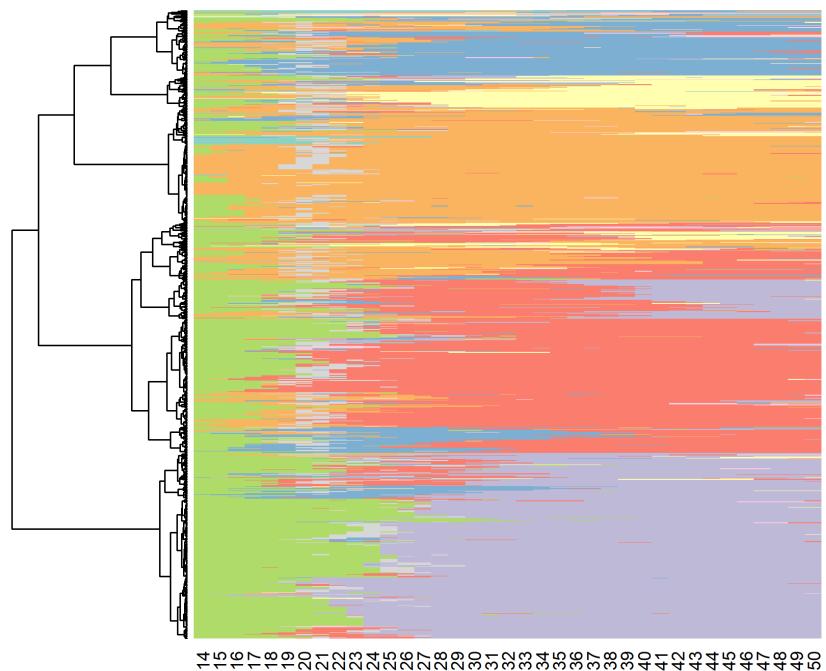


Figure 6. Tapis des séquences trié selon le dendrogramme

**NOTE**

Il est possible de reproduire un tapis de séquence avec `ggplot2`. Outre le fait que cela fournit plus d'options de personnalisation du graphique, cela permet également à ce que la hauteur de chaque classe sur le graphique soit proportionnelle aux nombre d'individus.

En premier lieu, on va ajouter à notre fichier de données des identifiants individuels, la typologie créée et l'ordre obtenu par *multidimensional scaling*.

```
R> donnees$id <- row.names(donnees)
donnees$classe <- seq.part
donnees$ordre <- rank(ordre, ties.method = "random")
```

Ensuite, il est impératif que nos données soient dans un format long et *tidy*, c'est-à-dire avec une ligne par individu et par pas de temps. Pour cela on aura recours à la fonction `gather` (voir le chapitre dédié, page 299).

```
R> library(tidyr)

Attaching package: 'tidyr'

The following object is masked from 'package:magrittr':
  extract
```

```
R> long <- donnees %>% gather(csp1:csp37, key = annee, value = csp)
```

On va mettre en forme la variable `csp` sous forme de facteur, récupérer l'année grâce à la fonction `str_sub` de l'extension `stringr` (voir le chapitre sur la manipulation de texte, page 0) et recalculer l'âge.

```
R> long$csp <- factor(long$csp, labels = c("agriculteur", "art./com./chefs",
  "cadres", "prof. int.", "employés", "ouvriers", "étudiants", "inactifs", "s
  erv. militaire"))
library(stringr)
long$annee <- as.integer(str_sub(long$annee, 4))
long$age <- long$annee + 13
```

Il n'y a plus qu'à faire notre graphique grâce à `geom_raster` qui permet de colorier chaque pixel. Techniquement, pour un tapis de séquence, il s'agit de représenter le temps sur l'axe horizontal et les

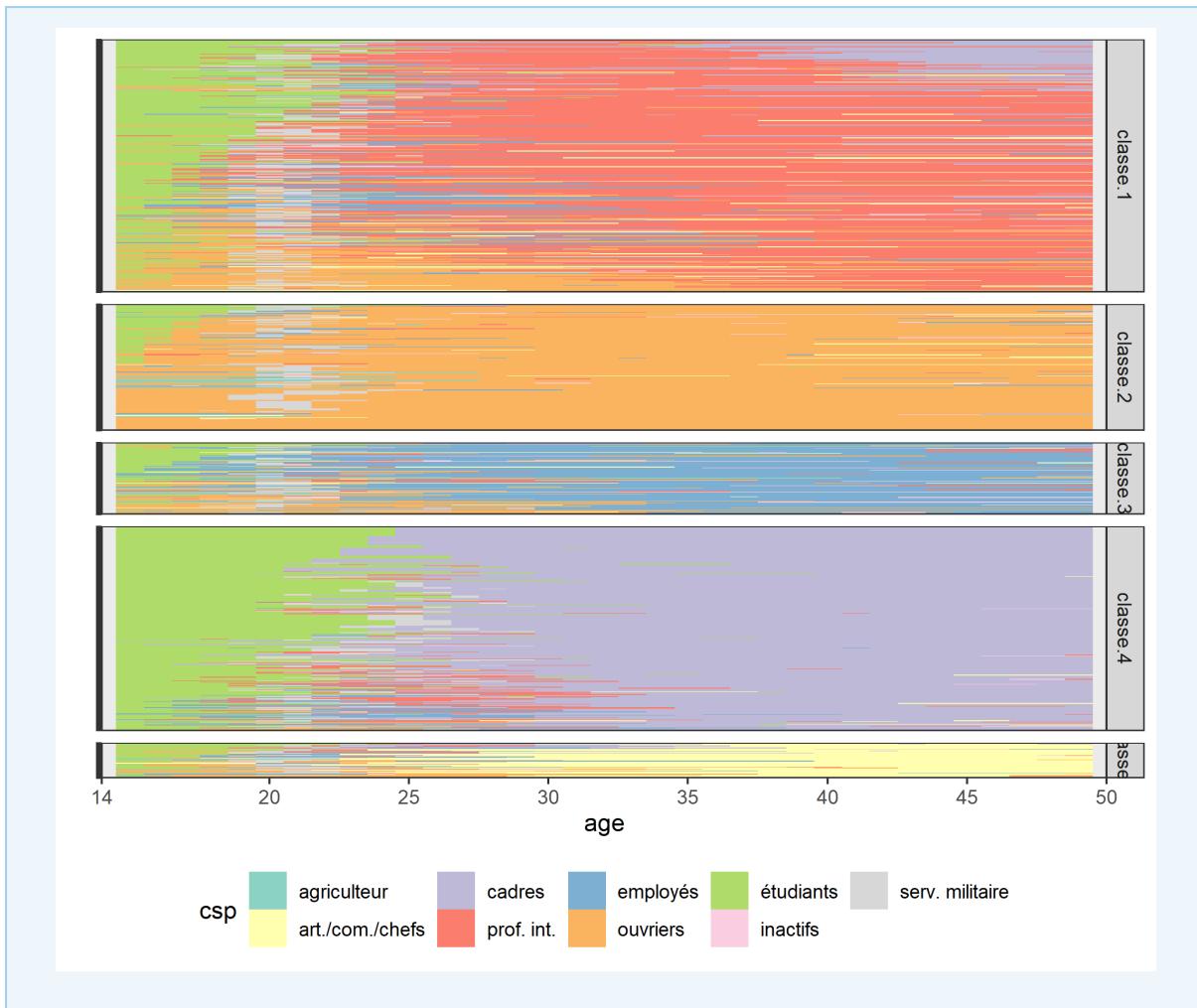
individus sur l'axe vertical. Petite astuce : plutôt que d'utiliser `id` pour l'axe vertical, nous utilisons `ordre` afin de trier les observations. Par ailleurs, il est impératif de transformer au passage `ordre` en facteur afin que `ggplot2` puisse recalculer proprement et séparément les axes pour chaque facette<sup>15</sup>, à condition de ne pas oublier l'option `scales = "free_y"` dans l'appel à `facet_grid`. Les autres commandes ont surtout pour vocation d'améliorer le rendu du graphique (voir le chapitre dédié à `ggplot2`, page 789).

```
R> library(ggplot2)
ggplot(long) +
  aes(x = age, y = factor(ordre), fill = csp) +
  geom_raster() +
  ylab("") +
  scale_y_discrete(label = NULL) +
  theme_bw() +
  theme(legend.position = "bottom") +
  scale_fill_brewer(palette = "Set3") +
  facet_grid(classe ~ ., scales = "free_y", space = "free_y") +
  scale_x_continuous(limits = c(14, 50), breaks = c(14, 20, 25, 30, 35, 40, 45, 50), expand = c(0, 0))
```

Warning: Removed 2000 rows containing missing values  
(geom\_raster).

---

15. Essayez le même code mais avec `y = ordre` au lieu de `y = factor(ordre)` et vous comprendrez tout l'intérêt de cette astuce.



La distance des séquences d'une classe au centre de cette classe, obtenue avec `disscenter`, permet de mesurer plus précisément l'homogénéité des classes. Nous utilisons ici `aggregate` pour calculer la moyenne par classe :

```
R> aggregate(disscenter(as.dist(seq.om)), group = seq.part, list(seq.part), mean)
```

Cela nous confirme que les classes 1, 3 et 5 sont nettement plus hétérogènes que les autres, alors que la classe 2 est la plus homogène.

D'autres représentations graphiques existent pour poursuivre l'examen de la typologie. On peut visualiser les 10 séquences les plus fréquentes de chaque classe avec `seqfplot`.

```
R> seqfplot(seq, group = seq.part)
```

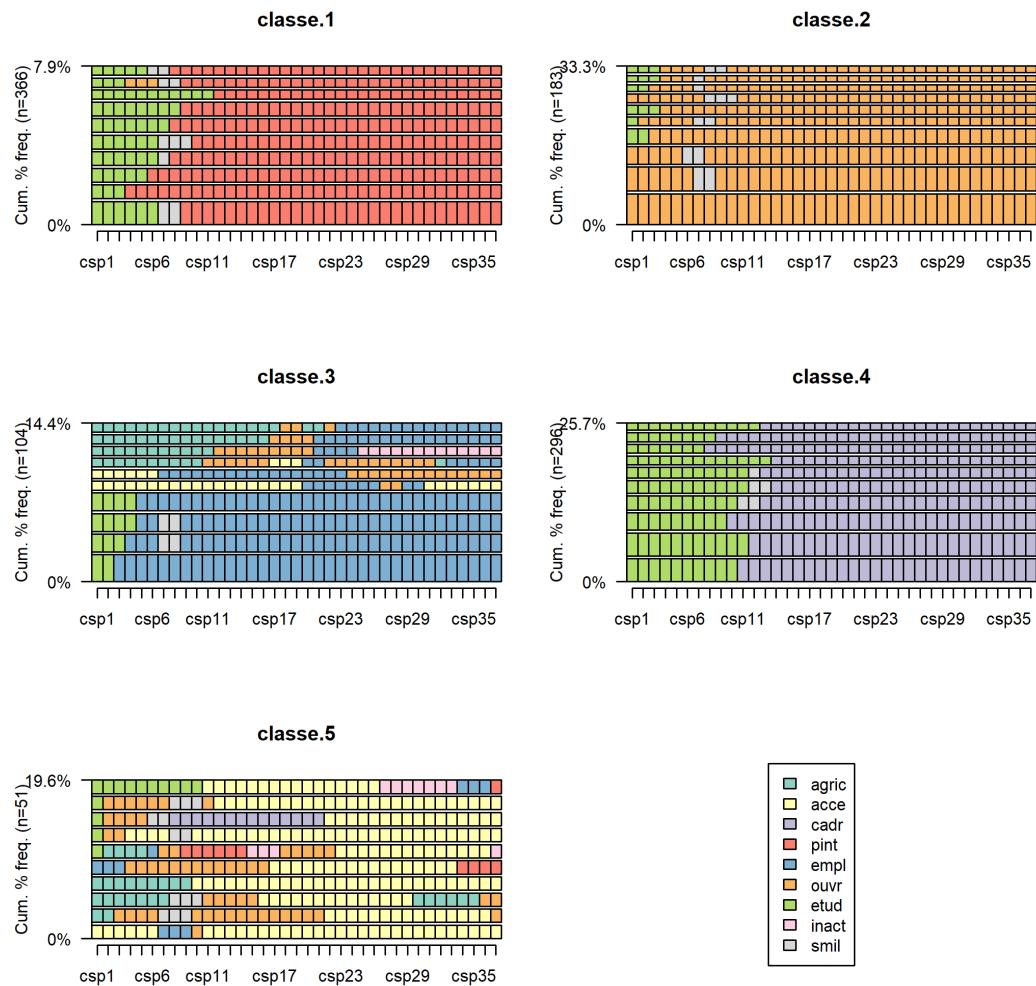


Figure 7. Séquences les plus fréquentes de chaque classe

On peut aussi visualiser avec `seqmsplot` l'état modal (celui qui correspond au plus grand nombre de séquences de la classe) à chaque âge.

```
R> seqmsplot(seq, group = seq.part, xtlab = 14:50, main = "classe")
```

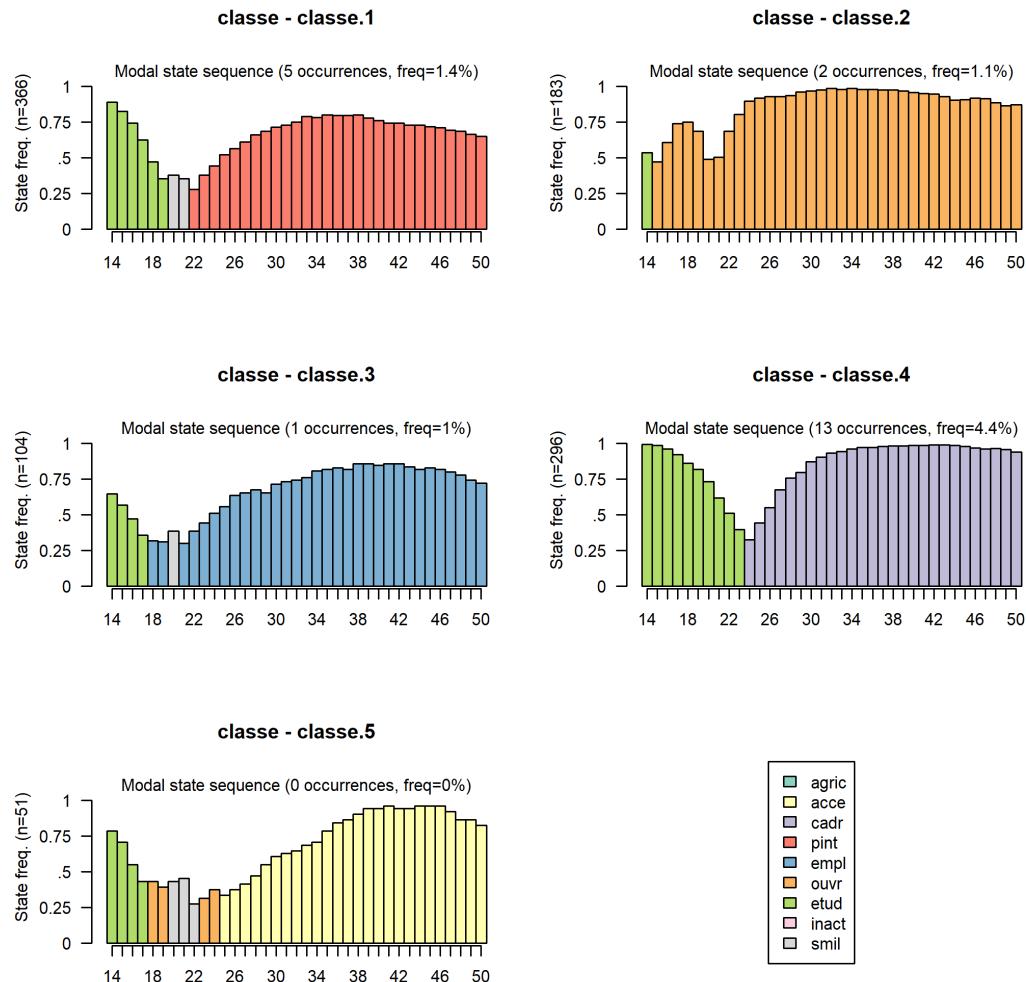


Figure 8. Statut modal à chaque âge

On peut également représenter avec `seqmtpplot` les durées moyennes passées dans les différents états.

```
R> seqmtplot(seq, group = seq.part)
```

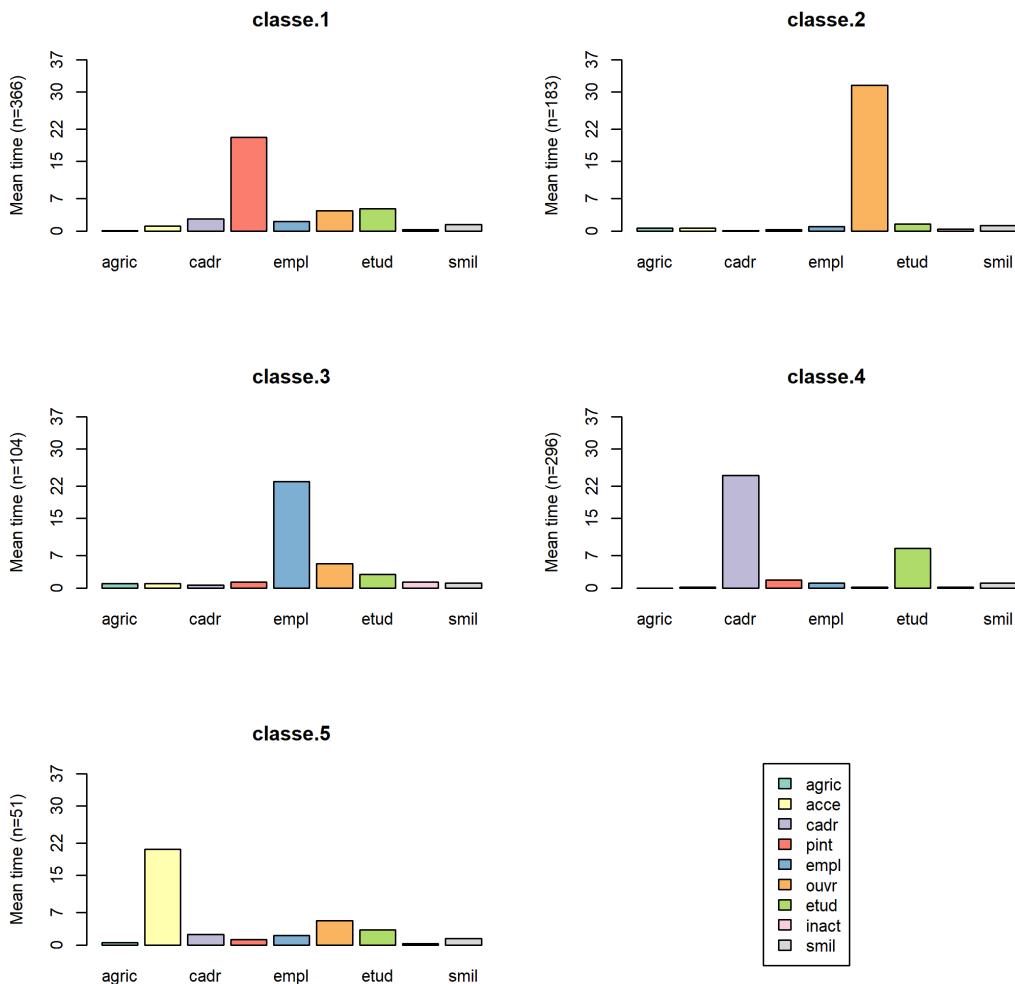


Figure 9. Durée moyenne dans chaque statut

La fonction `seqrplot` cherche à identifier des séquences «représentatives» de chaque classe. Plusieurs méthodes sont proposées (voir `seqrep`). La méthode `dist` cherche à identifier des séquences centrales à chaque classe, c'est-à-dire situées à proximité du centre de la classe. Selon l'hétérogénéité de la classe, plusieurs séquences «représentatives» peuvent être renvoyées. ATTENTION : il faut être prudent dans l'interprétation de ces séquences centrales de la classe dans la mesure où elles ne rendent pas toujours compte de ce qui se passe dans la classe et où elles peuvent induire en erreur quand la classe est assez hétérogène. Il faut donc les considérer tout en ayant en tête l'ensemble du tapis de séquence pour voir si elles sont effectivement de bonnes candidates.

```
R> seqrplot(seq, group = seq.part, dist.matrix = seq.om, criterion = "dist")
```

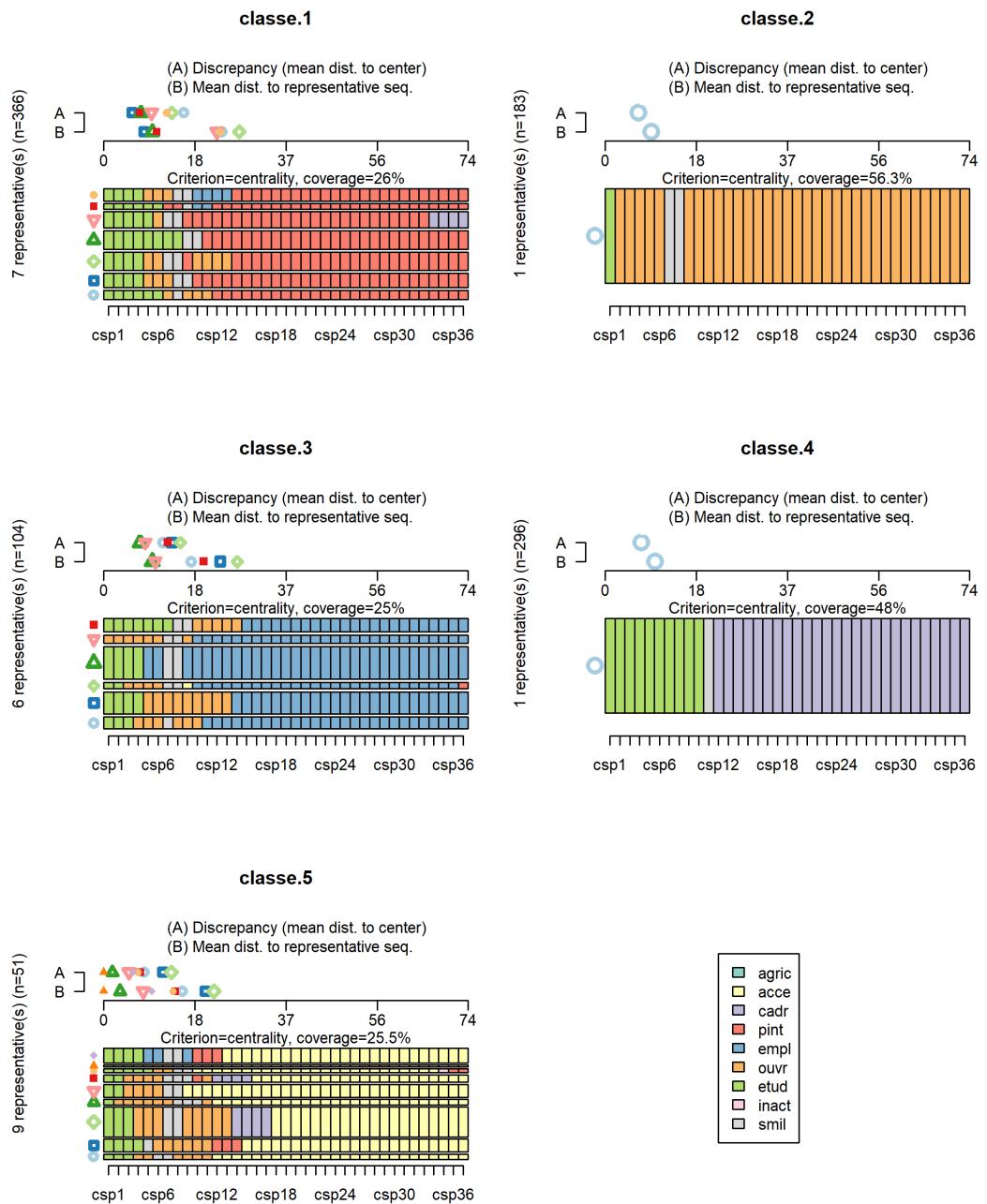


Figure 10. Séquences «représentatives» de chaque classe

Enfin, l'entropie transversale décrit l'évolution de l'homogénéité de la classe. Pour un âge donné, une entropie proche de 0 signifie que tous les individus de la classe (ou presque) sont dans la même situation. À l'inverse, l'entropie est de 1 si les individus sont dispersés dans toutes les situations. Ce type de graphique produit par `seqHtplot` peut être pratique pour localiser les moments de transition, l'insertion professionnelle ou une mobilité sociale ascendante.

```
R> seqHtplot(seq, group = seq.part, xtlab = 14:50)
```

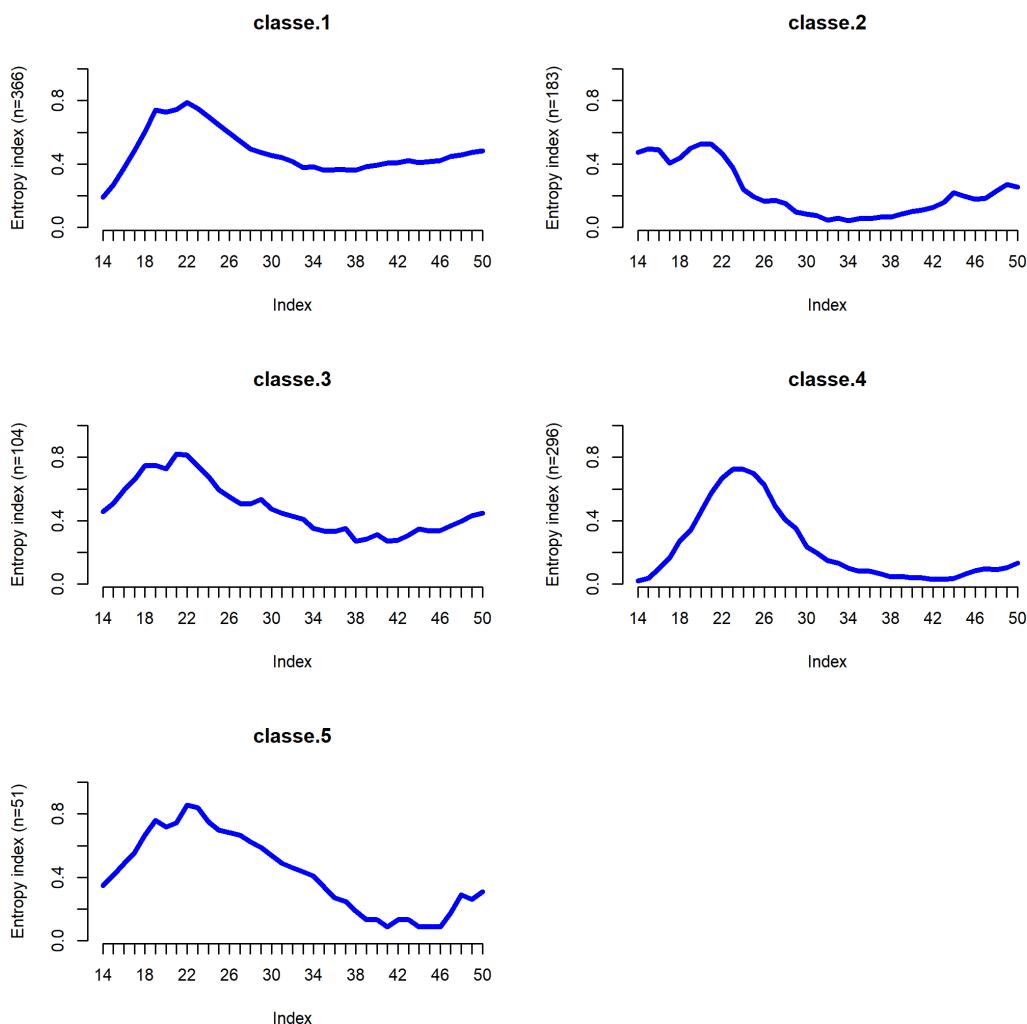


Figure 11. Entropie transversale

## Distribution de la typologie

On souhaite maintenant connaître la distribution de la typologie (en effectifs et en pourcentages) :

```
R> library(questionr)
  freq(seq.part)
```

On poursuit ensuite la description des classes en croisant la typologie avec la variable *generation* :

```
R> cprop(table(seq.part, donnees$generation))
```

seq.part	1930-38	1939-45	1946-50	Ensemble
classe.1	35.6	32.5	40.8	36.6
classe.2	19.7	18.3	17.0	18.3
classe.3	6.5	13.9	11.2	10.4
classe.4	31.8	29.2	27.9	29.6
classe.5	6.5	6.1	3.0	5.1
Total	100.0	100.0	100.0	100.0

```
R> chisq.test(table(seq.part, donnees$generation))
```

Pearson's Chi-squared test

```
data: table(seq.part, donnees$generation)
X-squared = 18.518, df = 8, p-value = 0.01766
```

Le lien entre le fait d'avoir un certain type de carrières et la cohorte de naissance est significatif à un seuil de 15 %. On constate par exemple l'augmentation continue de la proportion de carrières de type « professions intermédiaires » (classe 1) et, entre les deux cohortes les plus anciennes, l'augmentation de la part des carrières de type « employés » (classe 3) et la baisse de la part des carrières de type « cadres » (classe 4).

Bien d'autres analyses sont envisageables : croiser la typologie avec d'autres variables (origine sociale, etc.), construire l'espace des carrières possibles, étudier les interactions entre trajectoires familiales et professionnelles, analyser la variance des dissimilarités entre séquences en fonction de plusieurs variables « explicatives<sup>16</sup> »...

---

16. L'articulation entre méthodes « descriptives » et méthodes « explicatives » est un prolongement possible de l'analyse de séquences. Cependant, l'analyse de séquences était envisagée par Abbott comme une alternative à la sociologie

Mais l'exemple proposé est sans doute bien suffisant pour une première introduction !

## Pour aller plus loin

En premier lieu, la lecture du manuel d'utilisation de **TraMineR**, intitulé *Mining sequence data in R with the TraMineR package: A user's guide* et écrit par Alexis Gabadinho, Gilbert Ritschard, Matthias Studer et Nicolas S. Muller, est fortement conseillée. Ce manuel ne se contente pas de présenter l'extension, mais aborde également la théorie sous-jacente de l'analyse de séquences, les différents formats de données, les différences approches (séquences de statut ou séquences de transitions par exemple), etc.

Pour une initiation en français, on pourra se référer à l'ouvrage de Nicolas Robette *Explorer et décrire les parcours de vie : les typologies de trajectoires* sorti en 2011 aux éditions du Ceped.

L'extension **WeightedCluster** de Matthias Studer est un excellent complément à **TraMineR**. Il a également écrit un *manuel de la librairie WeightedCluster : un guide pratique pour la création de typologies de trajectoires en sciences sociales avec R*.

Enfin, l'extension **TraMineRextras** (<https://cran.r-project.org/package=TraMineRextras>) contient des fonctions complémentaires à **TraMineR**, plus ou moins en phase de test.

## Bibliographie

- Abbott A., 2001, *Time matters. On theory and method*, The University of Chicago Press.
- Abbott A., Hrycak A., 1990, « Measuring ressemblance in sequence data: an optimal matching analysis of musicians' careers», *American journal of sociology*, (96), p.144-185. <http://www.jstor.org/stable/10.2307/2780695>
- Abbott A., Tsay A., 2000, « Sequence analysis and optimal matching methods in sociology: Review and prospect », *Sociological methods & research*, 29(1), p.3-33. <http://smr.sagepub.com/content/29/1/3.short>
- Gabadinho, A., Ritschard, G., Müller, N.S. & Studer, M., 2011, « Analyzing and visualizing state sequences in R with TraMineR », *Journal of Statistical Software*, 40(4), p.1-37. <http://archive-ouverte.unige.ch/downloader/vital/pdf/tmp/4hff8pe6uhukqjavvgaluqmjq2/out.pdf>
- Grelet Y., 2002, « Des typologies de parcours. Méthodes et usages », *Document Génération* 92, (20), 47 p. [http://www.cmh.greco.ens.fr/programs/Grelet\\_typolparc.pdf](http://www.cmh.greco.ens.fr/programs/Grelet_typolparc.pdf)
- Lelièvre É., Vivier G., 2001, « Évaluation d'une collecte à la croisée du quantitatif et du qualitatif : l'enquête Biographies et entourage », *Population*, (6), p.1043-1073. [http://www.persee.fr/web/revues/home/prescript/article/pop\\_0032-4663\\_2001\\_num\\_56\\_6\\_7217](http://www.persee.fr/web/revues/home/prescript/article/pop_0032-4663_2001_num_56_6_7217)
- Lemercier C., 2005, « Les carrières des membres des institutions consulaires parisiennes au XIX<sup>e</sup>

---

quantitative *mainstream*, i.e le « paradigme des variables » et ses hypothèses implicites souvent difficilement tenables (Abbott, 2001). Une bonne description solidement fondée théoriquement vaut bien des « modèles explicatifs » (Savage, 2009).

- siècle », *Histoire et mesure*, XX (1-2), p.59-95. <http://histoiredemesure.revues.org/786>
- Lesnard L., 2008, « Off-Scheduling within Dual-Earner Couples: An Unequal and Negative Externality for Family Time », *American Journal of Sociology*, 114(2), p.447-490. [http://laurent.lesnard.free.fr/IMG/pdf/lesnard\\_2008\\_off-scheduling\\_within\\_dual-earner\\_couples-2.pdf](http://laurent.lesnard.free.fr/IMG/pdf/lesnard_2008_off-scheduling_within_dual-earner_couples-2.pdf)
  - Lesnard L., Saint Pol T. (de), 2006, « Introduction aux Méthodes d'Appariement Optimal (Optimal Matching Analysis) », *Bulletin de Méthodologie Sociologique*, 90, p.5-25. <http://bms.revues.org/index638.html>
  - Robette N., 2011, *Explorer et décrire les parcours de vie : les typologies de trajectoires*, Ceped (Les Clefs pour), 86 p. [http://nicolas.robette.free.fr/Docs/Robette2011\\_Manuel\\_TypoTraj.pdf](http://nicolas.robette.free.fr/Docs/Robette2011_Manuel_TypoTraj.pdf)
  - Robette N., 2012, « Du prosélytisme à la sécularisation. Le processus de diffusion de l'Optimal Matching Analysis », document de travail. [http://nicolas.robette.free.fr/Docs/Proselytisme\\_secularisation\\_NRobette.pdf](http://nicolas.robette.free.fr/Docs/Proselytisme_secularisation_NRobette.pdf)
  - Robette N., Bry X., 2012, « Harpoon or bait? A comparison of various metrics to fish for life course patterns », *Bulletin de Méthodologie Sociologique*, 116, p.5-24. [http://nicolas.robette.free.fr/Docs/Harpoon\\_maggot\\_RobetteBry.pdf](http://nicolas.robette.free.fr/Docs/Harpoon_maggot_RobetteBry.pdf)
  - Robette N., Thibault N., 2008, « L'analyse exploratoire de trajectoires professionnelles : analyse harmonique qualitative ou appariement optimal ? », *Population*, 64(3), p.621-646. <http://www.cairn.info/revue-population-2008-4-p-621.htm>
  - Savage M., 2009, « Contemporary Sociology and the Challenge of Descriptive Assemblage », *European Journal of Social Theory*, 12(1), p.155-174. <http://est.sagepub.com/content/12/1/155.short>

# Trajectoires de soins : un exemple de données longitudinales

Première description des données . . . . .	706
Évolution de la cascade de soins au cours du temps . . . . .	712
Analyse de survie classique . . . . .	714
Une première analyse de séquences sur l'ensemble du fichier . . . . .	722
Une seconde analyse de séquences limitées aux 18 premiers mois . . . . .	729
Facteurs associés à l'appartenance à chaque groupe . . . . .	742
Modèle mixte à classes latentes . . . . .	752
Modèle à observations répétées . . . . .	763
Modèle de survie multi-états . . . . .	770

Dans ce chapitre, nous allons aborder plusieurs méthodes d'analyse à partir d'un jeu de données longitudinales. Tout d'abord, importons les données dans R avec la commande suivante :

```
R> load(url("http://lamarange.github.io/analyse-R/data/care_trajectories.RData"))
```

```
R> class(care_trajectories)
```

```
[1] "data.table" "data.frame"
```

Nous obtenons un objet appelé `care_trajectories`. La fonction `class` nous montre qu'il s'agit d'un tableau de données au format `data.table` (voir le chapitre dédié, page 251). Chargeons donc cette extension ainsi que le `tidyverse`.

```
R> library(tidyverse, quietly = TRUE)
  library(data.table, quietly = TRUE)
```

## Première description des données

Jetons un premier regard aux données.

```
R> head(care_trajectories)
```

<b>id</b>	<b>month</b>	<b>care_status</b>	<b>sex</b>	<b>age</b>	<b>education</b>	<b>wealth</b>	<b>distance_clinic</b>
3	0	D	1	1	2	2	1
3	1	D	1	1	2	2	1
3	2	D	1	1	2	2	1
3	3	D	1	1	2	2	1
3	4	D	1	1	2	2	1
3	5	D	1	1	2	2	1

Il apparaît que les données sont dans un format «long» et *tidy* (voir le chapitre sur **tidyR**, page 299 pour une présentation du concept de *tidy data*), avec une ligne par individu et par pas de temps. Il apparaît également que les données sont stockées sous formes de vecteurs labellisés (voir le chapitre dédié aux vecteurs labellisés, page 123). Nous aurons donc besoin de l’extension **labelled**.

```
R> library(labelled)
```

Pour une description des variables, on pourra avoir recours à **describe** de **questionr**.

```
R> library(questionr)
  describe(care_trajectories, freq.n.max = 10)
```

```
[49365 obs. x 8 variables] data.table data.frame

$id: patient identifier
integer: 3 3 3 3 3 3 9 9 13 13 ...
min: 3 - max: 9998 - NAs: 0 (0%) - 2929 unique values
```

```

$month: month(s) since diagnosis
numeric: 0 1 2 3 4 5 0 1 0 1 ...
min: 0 - max: 50 - NAs: 0 (0%) - 51 unique values

$care_status: care status
labelled character: "D" ...
NAs: 0 (0%) - 4 unique values
4 value labels: [D] diagnosed, but not in care [C] in care, but not on treatment
[t] on treatment, but infection not suppressed [S] on treatment and suppressed
infection

n      %
[D] diagnosed, but not in care      25374 51.4
[C] in care, but not on treatment    5886 11.9
[T] on treatment, but infection not suppressed 4596 9.3
[S] on treatment and suppressed infection 13509 27.4
Total                            49365 100.0

$sex: sex
labelled double: 1 1 1 1 1 1 1 1 0 0 ...
min: 0 - max: 1 - NAs: 0 (0%) - 2 unique values
2 value labels: [0] male [1] female

n      %
[0] male    17781 36
[1] female  31584 64
Total      49365 100

$age: age group
labelled double: 1 1 1 1 1 1 2 2 2 2 ...
min: 1 - max: 3 - NAs: 0 (0%) - 3 unique values
3 value labels: [1] 16-29 [2] 30-59 [3] 60+

n      %
[1] 16-29 16911 34.3
[2] 30-59 29365 59.5
[3] 60+    3089  6.3
Total    49365 100.0

$education: education level
labelled double: 2 2 2 2 2 2 3 3 2 2 ...
min: 1 - max: 3 - NAs: 0 (0%) - 3 unique values
3 value labels: [1] primary [2] secondary [3] higher

n      %
[1] primary   10417 21.1
[2] secondary 19024 38.5

```

```
[3] higher      19924 40.4
Total          49365 100.0

$wealth: wealth group (assets score)
labelled double: 2 2 2 2 2 2 2 2 1 1 ...
min: 1 - max: 3 - NAs: 0 (0%) - 3 unique values
3 value labels: [1] low [2] middle [3] high

n      %
[1] low    15432 31.3
[2] middle 20769 42.1
[3] high   13164 26.7
Total      49365 100.0

$distance_clinic: distance to nearest clinic
labelled double: 1 1 1 1 1 1 2 2 2 2 ...
min: 1 - max: 2 - NAs: 0 (0%) - 2 unique values
2 value labels: [1] less than 10 km [2] 10 km or more

n      %
[1] less than 10 km 26804 54.3
[2] 10 km or more   22561 45.7
Total              49365 100.0
```

Dans cette étude, on a suivi des patients à partir du moment où ils ont été diagnostiqués pour une pathologie grave et chronique et on a suivi leurs parcours de soins chaque mois à partir du diagnostic. La variable `status` contient le statut dans les soins de chaque individu pour chaque mois de suivi :

- **D**: s'il n'est pas actuellement suivi dans une clinique, soit que la personne n'est pas encore entrée en clinique après le diagnostic, soit qu'elle a quitté la clinique et qu'elle est donc sortie des soins ;
- **C**: indique que le patient est entré en soins (il est suivi dans une clinique) mais il n'a pas encore commencé le traitement, ou bien il a arrêté le traitement mais est toujours suivi en clinique ;
- **T**: la personne est sous traitement mais l'infection n'est pas «supprimée» ou «contrôlée», soit que le traitement n'a pas encore eu le temps de faire effet, soit qu'il n'est plus efficace ;
- **S**: la personne est suivie en clinique, sous traitement et son infection est «supprimée» / «contrôlée», indiquant que le traitement est efficace et produit son effet. Cette étape ultime du parcours de soins est celle dans laquelle on souhaite maintenir les individus le plus longtemps possible.

Il est important de noter que nous avons ici des statuts **hiérarchiquement ordonnés** ( $D < C < T < S$ ), ce qui aura son importance pour les choix méthodologiques que nous aurons à faire.

Nous disposons également d'autres variables (âge, sexe, niveau d'éducation...) qui sont ici dépendantes du temps, c'est-à-dire que le cas échéant, elles peuvent varier d'un mois à l'autre en cas de changement.

Avant de démarrer les analyses, francisons certaines de ces variables.

```
R> var_label(care_trajectories$sex) <- "Sexe"  
val_labels(care_trajectories$sex) <- c(homme = 0, femme = 1)  
var_label(care_trajectories$age) <- "Âge"  
var_label(care_trajectories$education) <- "Education"  
val_labels(care_trajectories$education) <- c(  
    primaire = 1,  
    secondaire = 2,  
    supérieur = 3  
)
```

Le fichier contient 49 365 lignes, ce qui ne veut pas dire qu'il y a ce nombre d'invidus suivis au cours du temps, puisque plusieurs lignes correspondent à un même individu. On peut obtenir le nombre d'individus différents assez facilement avec la commande :

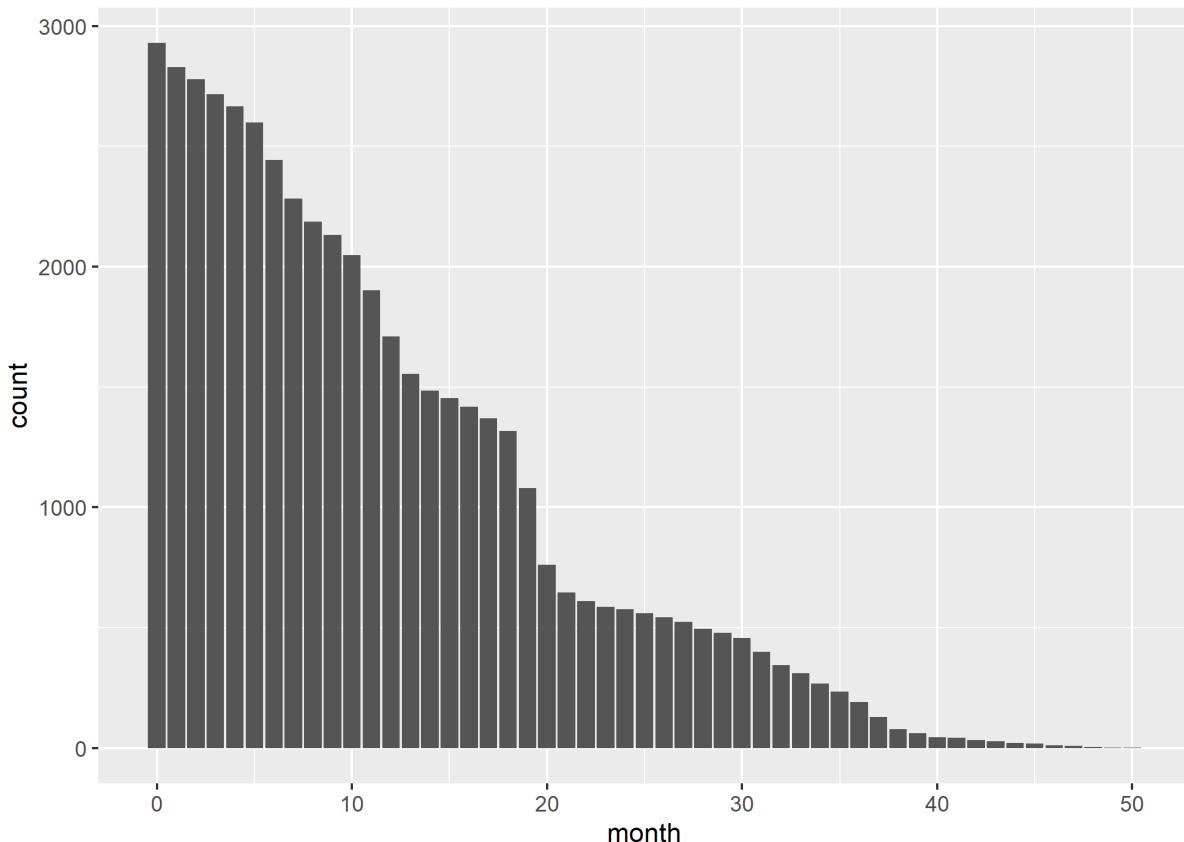
```
R> length(unique(care_trajectories$id))
```

```
[1] 2929
```

**Précision :** dans ce fichier, tous les individus ne sont pas suivis pendant la même durée, car ils n'ont pas tous été diagnostiqués au même moment. Cependant, il n'y a pas de « trous » dans le suivi (ce qui serait le cas si certains individus sortaient de l'observation pendant quelques mois puis re-rentraient dans la cohorte de suivi).

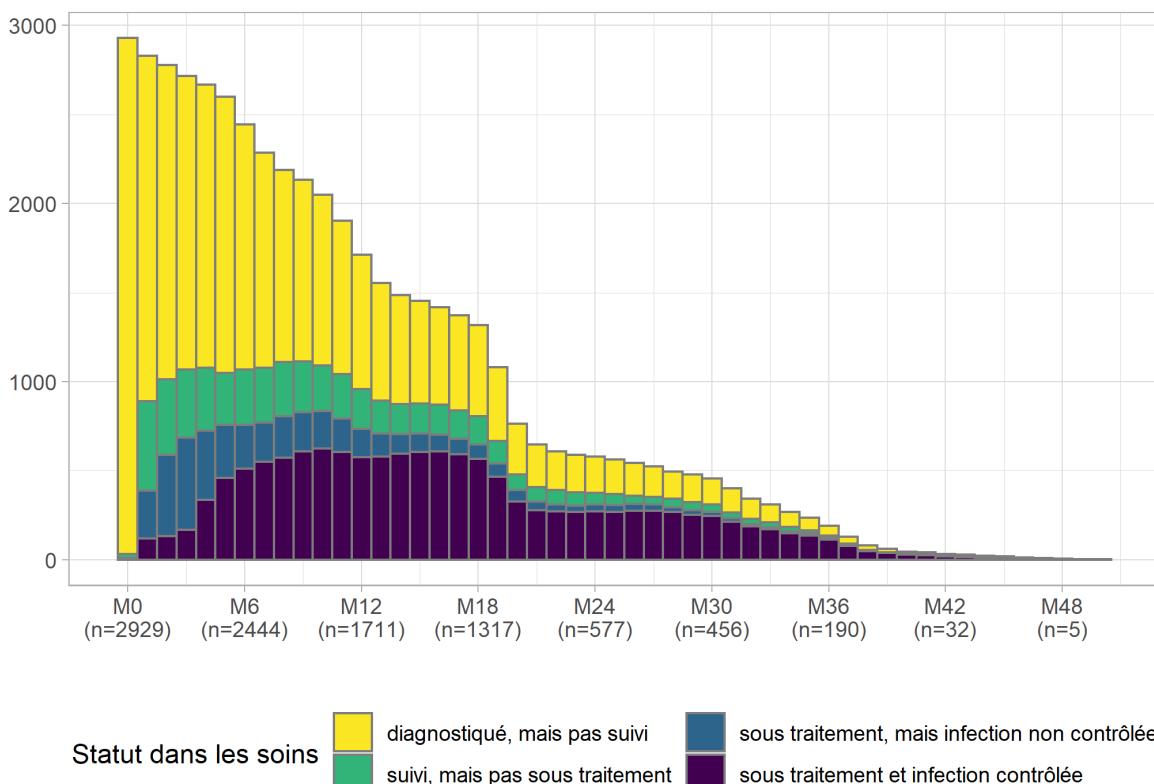
Avant d'aller plus avant, il nous faut avoir une idée du nombre d'individus observé au cours du temps, ce que l'on peut obtenir avec :

```
R> ggplot(care_trajectories) +
  aes(x = month) +
  geom_bar()
```



Améliorons ce graphique en y ajoutant la distribution selon le statut dans les soins chaque mois, en améliorant l’axe du temps (tous les 6 mois est plus facile à lire) et en y ajoutant un titre et des étiquettes appropriées. Afin de disposer d’une palette de couleurs à fort contraste, nous allons utiliser l’extension [viridis](#). Enfin, nous allons utiliser une petite astuce pour indiquer les effectifs sur l’axe horizontal. Au passage, nous allons également franciser les étiquettes de la variable `care_status` avec [val\\_labels](#) (notez aussi le recours à `to_factor` dans `aes` qui nous permet de transformer à la volée la variable en facteur, format attendu par `ggplot2` pour les variables catégorielles). On se référera au chapitre dédié à `ggplot2`, page 789 pour plus de détails sur les différentes fonctions de cette extension graphique.

```
R> library(viridis)
n <- care_trajectories[month %in% (0:8*6), .(n = .N), by = month]$n
etiquettes <- paste0("M", 0:8*6, "\n(n=", n, ")")
val_labels(care_trajectories$care_status) <- c(
  "diagnostiqué, mais pas suivi" = "D",
  "suivi, mais pas sous traitement" = "C",
  "sous traitement, mais infection non contrôlée" = "T",
  "sous traitement et infection contrôlée" = "S"
)
ggplot(care_trajectories) +
  aes(x = month, fill = to_factor(care_status)) +
  geom_bar(color = "gray50", width = 1) +
  scale_x_continuous(breaks = 0:8*6, labels = etiquettes) +
  ggtitle("Distribution du statut dans les soins chaque mois") +
  xlab("") + ylab("") +
  theme_light() +
  theme(legend.position = "bottom") +
  labs(fill = "Statut dans les soins") +
  scale_fill_viridis(discrete = TRUE, direction = -1) +
  guides(fill = guide_legend(nrow = 2))
```

**Distribution du statut dans les soins chaque mois**

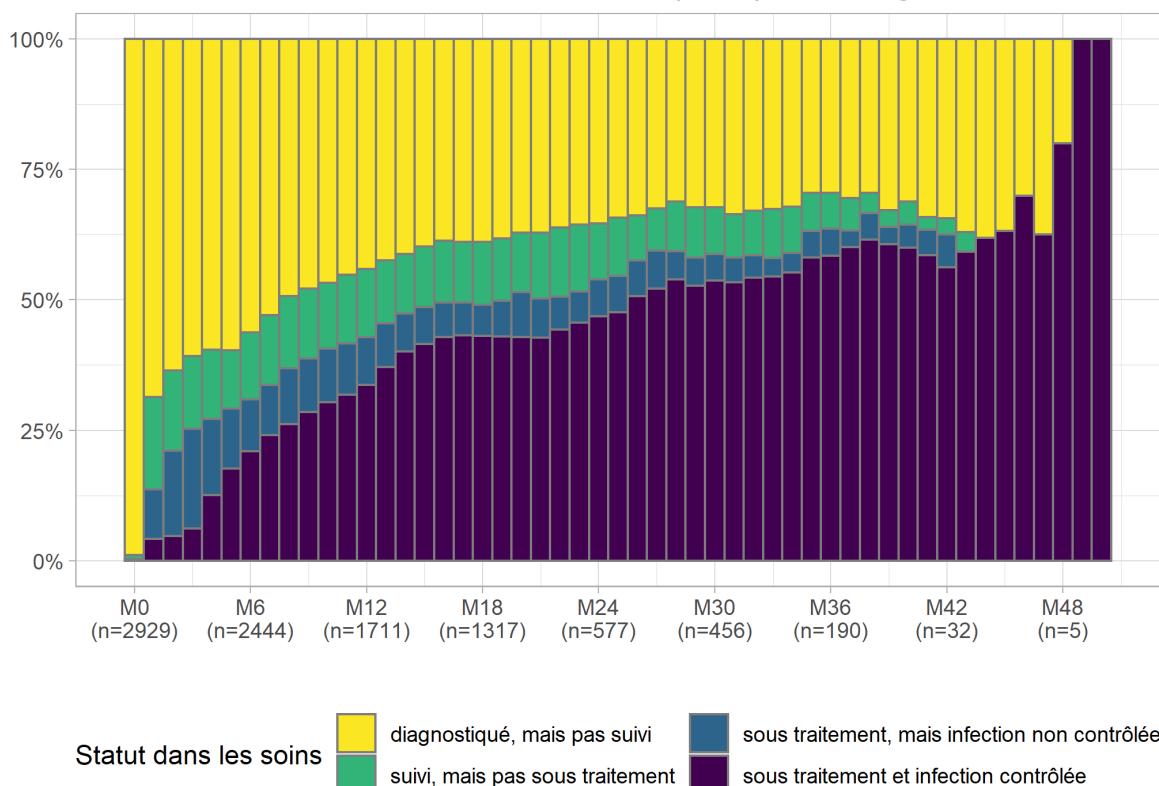
On s’aperçoit qu’une majorité des personnes suivies ne l’ont été que peu de temps, avec une décroissance rapide des effectifs.

## Évolution de la cascade de soins au cours du temps

On nomme communément «cascade de soins» la proportion d’individus dans chaque statut à un moment du temps donné. On peut facilement obtenir celle-ci à partir du code du graphique précédent en ajoutant l’option `position = fill` à `geom_bar`.

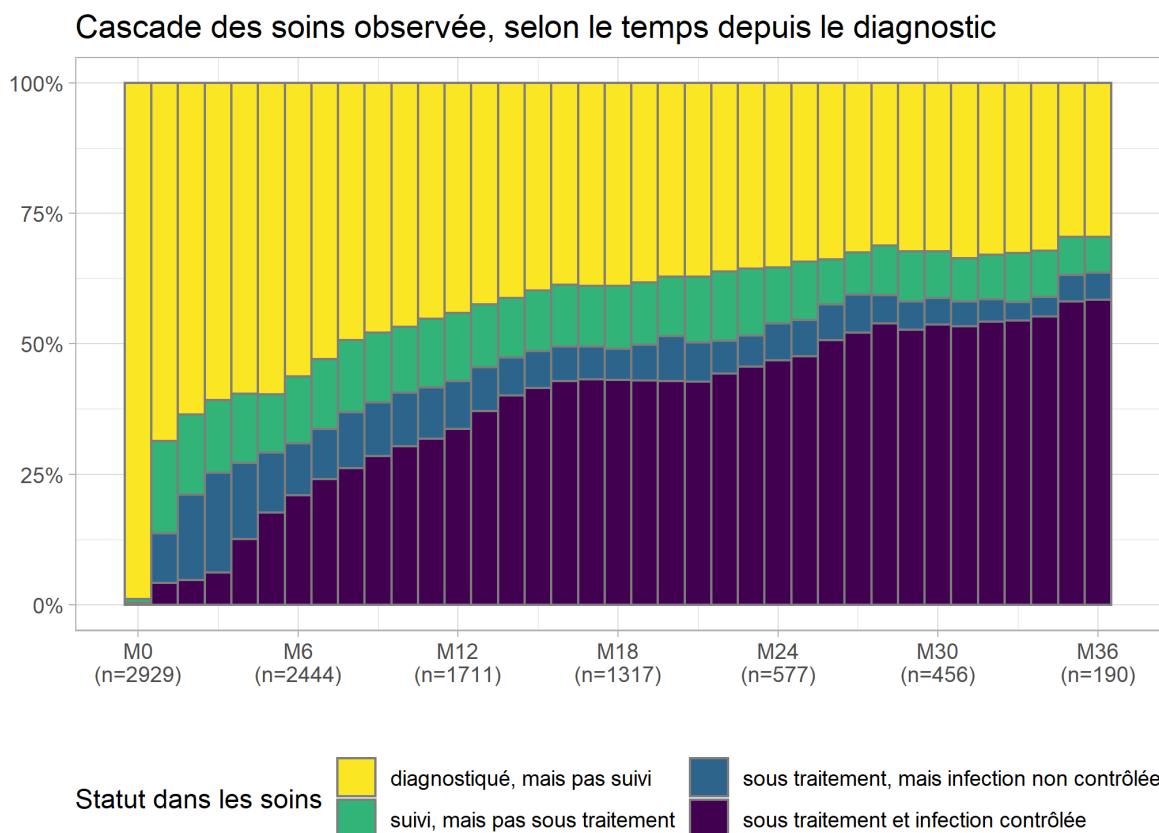
```
R> ggplot(care_trajectories) +
  aes(x = month, fill = to_factor(care_status)) +
  geom_bar(color = "gray50", width = 1, position = "fill") +
  scale_x_continuous(breaks = 0:8*6, labels = etiquettes) +
  scale_y_continuous(labels = scales::percent) +
  ggtitle("Cascade des soins observée, selon le temps depuis le diagnostic") +
  xlab("") + ylab("") +
  theme_light() +
  theme(legend.position = "bottom") +
  labs(fill = "Statut dans les soins") +
  scale_fill_viridis(discrete = TRUE, direction = -1) +
  guides(fill = guide_legend(nrow = 2))
```

Cascade des soins observée, selon le temps depuis le diagnostic



Les effectifs sont très faibles au-delà de 36 mois et il serait préférable de couper la cascade au-delà de M36, ce que l'on peut faire aisément ne gardant que les lignes correspondantes de `care_trajectories`.

```
R> casc_obs <- ggplot(care_trajectories[month <= 36]) +
  aes(x = month, fill = to_factor(care_status)) +
  geom_bar(color = "gray50", width = 1, position = "fill") +
  scale_x_continuous(breaks = 0:8*6, labels = etiquettes) +
  scale_y_continuous(labels = scales::percent) +
  ggttitle("Cascade des soins observée, selon le temps depuis le diagnostic") +
  xlab("") + ylab("") +
  theme_light() +
  theme(legend.position = "bottom") +
  labs(fill = "Statut dans les soins") +
  scale_fill_viridis(discrete = TRUE, direction = -1) +
  guides(fill = guide_legend(nrow = 2))
casc_obs
```



## Analyse de survie classique

L'analyse de survie constitue l'approche statistique la plus fréquente pour appréhender des données biographiques. Dans sa version classique, l'analyse de survie modélise le temps mis pour vivre un événement particulier à partir d'un événement origine.

Dans notre exemple, l'événement d'origine commun à tous les individus est le diagnostic VIH. Les personnes suivies peuvent vivre trois événements principaux :

- entrée en soins (passage de **D** à **C**) ;
- initiation du traitement (passage de **C** à **T**) ;
- contrôle de l'infection (passage de **T** à **S**).

En toute rigueur, il faudrait également considérer les transitions inverses (sortie de soins, arrêt du traitement, échec virologique). De même, il est possible que certains aient vécus plusieurs transitions successives (entrée en soin, initiation du traitement, sortie de soins et arrêt du traitement, nouvelle entrée en soins...).

Pour le moment, contentons-nous de regarder la **première** entrée en soins, la **première** initiation du traitement et la **première** atteinte du contrôle de l'infection et de calculer la date de ces trois événements, dans un fichier `ind` contenant une ligne par individu.

```
R> ind <- care_trajectories[month == 0]
  ind$diagnostic <- 0
  ind <- merge(
    ind,
    care_trajectories[
      care_status %in% c("C", "T", "S"),
      .(entree_soins = min(month)),
      by = id
    ],
    by = "id",
    all.x = TRUE
  )
  ind <- merge(
    ind,
    care_trajectories[
      care_status %in% c("T", "S"),
      .(initiation_tt = min(month)),
      by = id
    ],
    by = "id",
    all.x = TRUE
  )
  ind <- merge(
    ind,
    care_trajectories[
      care_status == "S",
      .(controle = min(month)),
      by = id
    ],
    by = "id",
    all.x = TRUE
  )
```

Il nous faut également la durée de suivi par individu.

```
R> ind <- merge(
  ind,
  care_trajectories[, .(suivi = max(month)), by = id],
  by = "id",
  all.x = TRUE
)
```

Pour faciliter la suite des analyses, nous allons nous créer une petite fonction qui, en fonction de la date d’origine et la date d’événement retenues, calculera la courbe de Kaplan-Meier correspondante (voir le chapitre sur l’analyse de suivie, page 651 pour le calcul de la courbe de survie et celui dédié à l’écriture de fonctions).

```
R> km <- function(date_origine, date_evenement, nom) {
  library(survival)

  # ne garder que les observations avec date d'origine
  tmp <- ind[!is.na(ind[[date_origine]]), ]

  # pre-remplir la variable time avec duree de suivi
  # depuis date d'origine
  tmp$time <- tmp$suivi - tmp[[date_origine]]
  # et considerer que l'événement n'a pas été vécu
  tmp$event <- FALSE

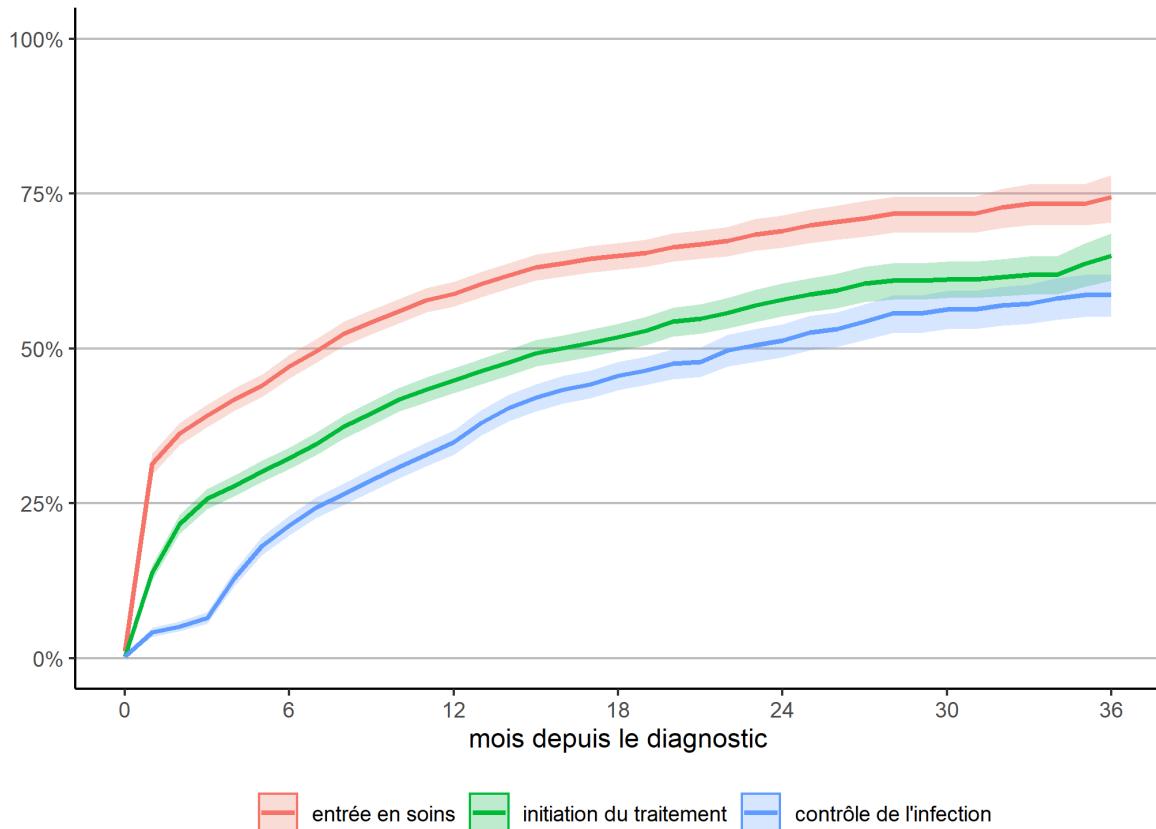
  # si date_eveement documentée, événement vécu
  tmp[!is.na(tmp[[date_evenement]]), ]$event <- TRUE
  tmp[tmp$event == TRUE, ]$time <-
    tmp[tmp$event == TRUE, ][[date_evenement]] -
    tmp[tmp$event == TRUE, ][[date_origine]]

  kaplan <- survfit(Surv(time, event) ~ 1, data = tmp)
  res <- broom::tidy(kaplan, conf.int = TRUE)
  res$nom <- nom
  res
}
```

Une première approche consiste à regarder la survenue de chacun des trois événements mentionnés plus haut en fonction du temps depuis le diagnostic.

```
R> depuis_diag <- dplyr::bind_rows(
  km("diagnostic", "entree_soins", "entrée en soins"),
  km("diagnostic", "initiation_tt", "initiation du traitement"),
  km("diagnostic", "controle", "contrôle de l'infection")
)

g_diag <- ggplot(data = depuis_diag) +
  aes(x = time, y = 1 - estimate,
      color = as_factor(nom), fill = as_factor(nom),
      ymin = 1 - conf.high, ymax = 1 - conf.low) +
  geom_ribbon(alpha = .25, mapping = aes(color = NULL)) +
  geom_line(size = 1) +
  theme_classic() +
  theme(
    legend.position = "bottom",
    panel.grid.major.y = element_line(colour = "grey")
  ) +
  scale_x_continuous(breaks = 0:6*6, limits = c(0, 36)) +
  scale_y_continuous(labels = scales::percent, limits = c(0, 1)) +
  xlab("mois depuis le diagnostic") +
  ylab("") + labs(color = "", fill = "")
g_diag
```

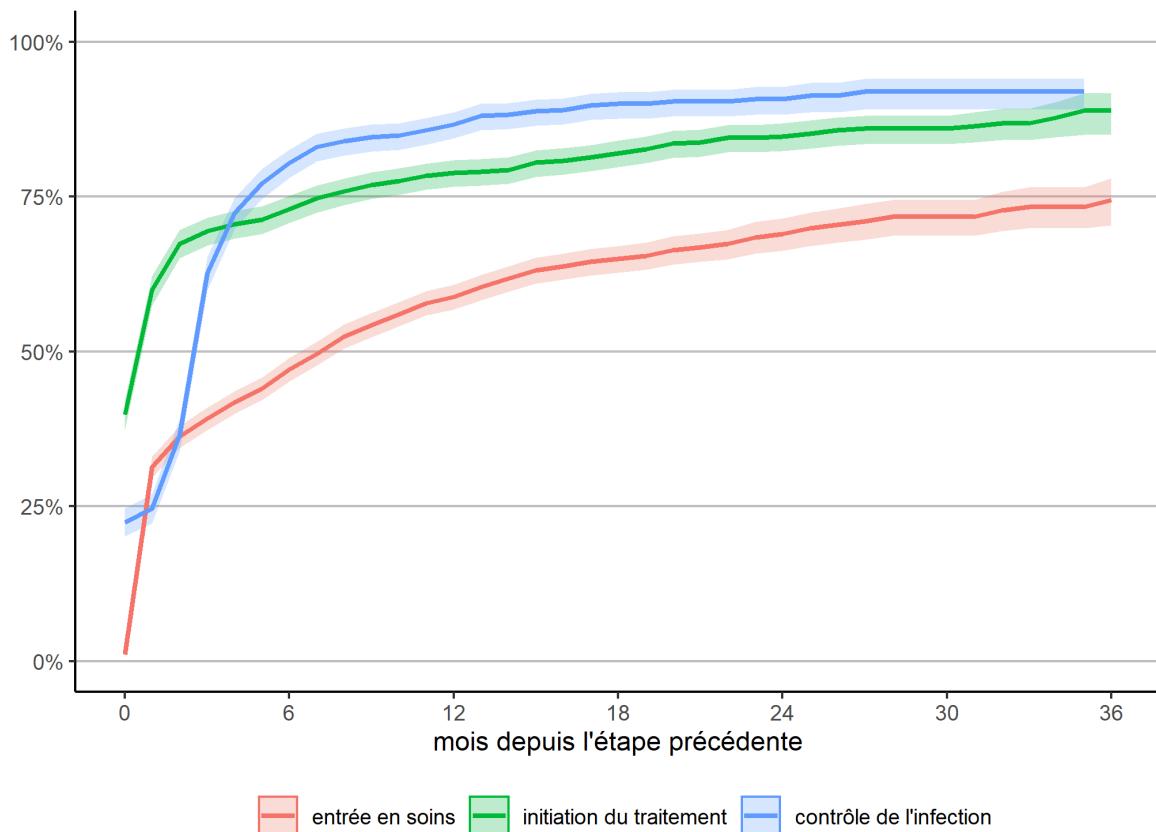


Ce graphique ressemble à la cascade des soins observée que nous avions calculée plus haut, à quelques différences près :

- avec la méthode de Kaplan-Meier, la censure à droite, i.e. le fait que tous les individus n'ont pas la même durée de suivi, est correctement prise en compte et la courbe est corrigée en conséquence ;
- par contre, les transitions inverses ne sont pas considérées : lorsqu'un individu a atteint une étape, on ne regarde pas s'il en ressort.

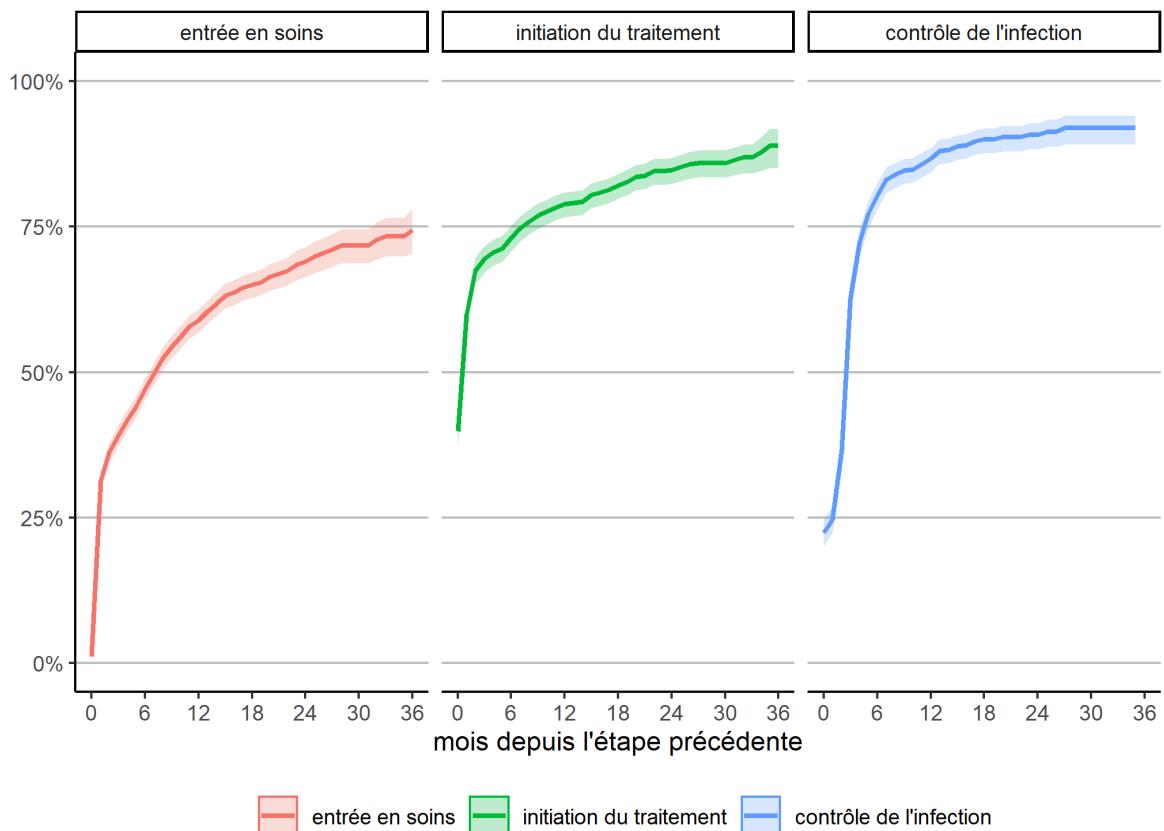
Une autre manière d'appréhender nos trajectoires est de considérer le temps requis pour atteindre une étape une fois la précédente étape atteinte. Ce qu'on obtient facilement en adaptant légèrement notre code précédent.

```
R> depuis_prec <- dplyr::bind_rows(  
  km("diagnostic", "entree_soins", "entrée en soins"),  
  km("entree_soins", "initiation_tt", "initiation du traitement"),  
  km("initiation_tt", "controle", "contrôle de l'infection")  
)  
  
g_prec <- ggplot(data = depuis_prec) +  
  aes(x = time, y = 1 - estimate,  
      color = as_factor(nom), fill = as_factor(nom),  
      ymin = 1 - conf.high, ymax = 1 - conf.low) +  
  geom_ribbon(alpha = .25, mapping = aes(color = NULL)) +  
  geom_line(size = 1) +  
  theme_classic() +  
  theme(  
    legend.position = "bottom",  
    panel.grid.major.y = element_line(colour = "grey")  
) +  
  scale_x_continuous(breaks = 0:6*6, limits = c(0, 36)) +  
  scale_y_continuous(labels = scales::percent, limits = c(0, 1)) +  
  xlab("mois depuis l'étape précédente") +  
  ylab("") + labs(color = "", fill = "")  
  
g_prec
```



Attention : cette représentation graphique peut éventuellement prêter à confusion dans sa lecture car l'échelle de temps n'est pas tout à fait la même pour chaque courbe, dans la mesure où la date d'origine diffère pour chacune. Dès lors, il peut être plus pertinent de présenter chaque courbe l'une à côté de l'autre.

```
R> g_prec + facet_grid(~ as_factor(nom))
```



Ici, on voit plus clairement que l'étape où il y a le plus de «perdition» est celle de l'entrée en soins, moins des trois quarts des personnes diagnostiquées étant venu en clinique au moins une fois dans les trois ans suivant le diagnostic. Par contre, l'initiation du traitement une fois entré en clinique et le contrôle de l'infection une fois le traitement initié sont beaucoup plus rapides.

Pour aller plus loin avec les outils de l'analyse de survie classique, il serait possible de faire des analyses bivariées (Kaplan-Meier) ou multivariées (Cox) pour chacune de ces étapes. Cependant, il serait plus intéressant de trouver une approche statistique permettant de considérer dans un même modèle l'ensemble des transitions possibles.

## Une première analyse de séquences sur l'ensemble du fichier

L'analyse de séquences permet d'appréhender l'ensemble de la trajectoire de soins à travers la succession des états dans lesquels se trouvent les patients observés.

Nous allons donc réaliser une analyse de séquences (voir le chapitre dédié, page 679) sur l'ensemble de notre fichier. Pour cela, il va falloir préalable que nous transformions nos données actuellement dans un format «long» en un tableau «large», c'est-à-dire avec une ligne par individu et une variable différentes par pas de temps. On peut réaliser cela facilement avec `pivot_wider` de `tidyverse` (voir le chapitre dédié à `tidyverse`, page 0). Il faut noter que le résultat ne sera pas un tableau `data.table`, d'où le recours à `setDT` pour convertir le résultat.

```
R> library(tidyverse)
large <- care_trajectories %>%
  dplyr::select(id, m = month, care_status) %>%
  pivot_wider(names_from = m, values_from = care_status, names_prefix = "m")
setDT(large)
head(large)
```

<b>id</b>	<b>m0</b>	<b>m1</b>	<b>m2</b>	<b>m3</b>	<b>m4</b>	<b>m5</b>	<b>m6</b>	<b>m7</b>	<b>m8</b>	<b>m9</b>	<b>m10</b>	<b>m11</b>	<b>m12</b>	<b>m13</b>
3	D	D	D	D	D	D	NA	NA	NA	NA	NA	NA	NA	NA
9	D	D	NA	NA	NA	NA								
13	D	D	D	D	D	D	D	NA	NA	NA	NA	NA	NA	NA
15	D	D	D	D	T	T	T	C	D	D	D	D	D	D
18	D	D	S	S	S	S	S	S	S	S	S	S	S	S
21	D	D	D	D	D	D	NA	NA	NA	NA	NA	NA	NA	NA

On utilise `seqdef` de `TraMineR` pour créer nos séquences, avec les arguments `alphabet` pour forcer l'ordre de l'alphabet, `states` pour spécifier des étiquettes courtes à chaque état et `cpal` pour indiquer le code couleur de chaque état (et être raccord avec nos graphiques précédents).

```
R> library(TraMineR, quietly = TRUE)
```

```
TraMineR stable version 2.2-1 (Built: 2020-11-02)
```

```
Website: http://traminer.unige.ch
```

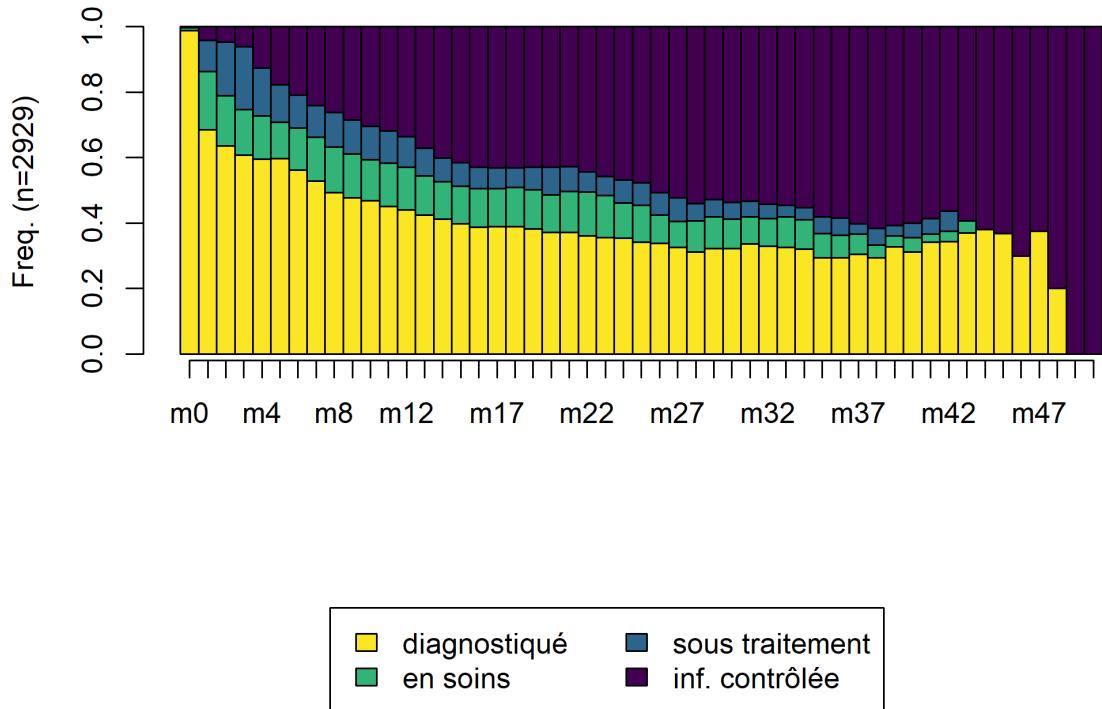
```
Please type 'citation("TraMineR")' for citation information.
```

```
R> seq_all <- seqdef(
  large[, m0:m50],
  id = large$id,
  alphabet = c("D", "C", "T", "S"),
  states = c("diagnostiqué", "en soins", "sous traitement", "inf. contrôlée"),
  cpal = viridis(4, direction = -1)
)
```

```
[>] found missing values ('NA') in sequence data
[>] preparing 2929 sequences
[>] coding void elements with '%' and missing values with '*'
[>] state coding:
[alphabet] [label] [long label]
1 D diagnostiqué diagnostiqué
2 C en soins en soins
3 T sous traitement sous traitement
4 S inf. contrôlée inf. contrôlée
[>] 2929 sequences in the data set
[>] min/max sequence length: 1/51
```

On peut retrouver la cascade de soins avec `seqdplot`.

```
R> seqdplot(seq_all, legend.prop = .25)
```



Nous allons maintenant calculer une matrice des distances entre individus par *optimal matching*. Dans le cas présent, nos différents status sont hiérarchiquement ordonnés. Il n'est donc pas raisonnable de penser que les coûts sont constants entre les différents statuts, puisqu'en un sens, passer directement de **D** à **T** peut être considéré comme être passé d'abord de **D** à **C** puis de **C** à **T**. Nous allons donc faire une matrice de coûts hiérarchisée. `seqcost` nous permet de produire une matrice de coûts constants, que nous allons ensuite modifier manuellement. Pour le coût *indel*, le plus simple est de considérer la moitié du coût de substitution maximum.

```
R> couts <- seqcost(seq_all, method = "CONSTANT")
```

```
[>] creating 4x4 substitution-cost matrix using 2 as constant value
```

```
R> couts
```

```
$indel
```

```
[1] 1
```

```
$sm
      diagnostiqué-> en soins->
diagnostiqué->          0      2
en soins->                2      0
sous traitement->        2      2
inf. contrôlée->         2      2
      sous traitement-> inf. contrôlée->
diagnostiqué->          2      2
en soins->                2      2
sous traitement->        0      2
inf. contrôlée->         2      0
```

```
R> couts$sm[1, ] <- c(0, 1, 2, 3)
  couts$sm[2, ] <- c(1, 0, 1, 2)
  couts$sm[3, ] <- c(2, 1, 0, 1)
  couts$sm[4, ] <- c(3, 2, 1, 0)
  couts$indel <- max(couts$sm) / 2
  couts
```

```
$indel
[1] 1.5
```

```
$sm
      diagnostiqué-> en soins->
diagnostiqué->          0      1
en soins->                1      0
sous traitement->        2      1
inf. contrôlée->         3      2
      sous traitement-> inf. contrôlée->
diagnostiqué->          2      3
en soins->                1      2
sous traitement->        0      1
inf. contrôlée->         1      0
```

```
R> dist_all <- seqdist(seq_all, method = "OM", sm = couts$sm, indel = couts$inde
l)
```

```
[>] 2929 sequences with 4 distinct states
```

```
[>] checking 'sm' (size and triangle inequality)
```

```
[>] 1370 distinct sequences
[>] min/max sequence lengths: 1/51
[>] computing distances using the OM metric
[>] elapsed time: 3.86 secs
```

Calculons le dendrogramme et représentons le avec le tapis de séquence grâce à `seq_heatmap` de l'extension **JLutils**. Pour rappel, cette extension est seulement disponible sur [GitHub](#). On l'installera donc (ou on la mettra à jour) avec la commande `devtools::install_github("lamarange/JLutils")`.

```
R> arbre_all <- hclust(as.dist(dist_all), method = "ward.D2")
library(JLutils, quietly = TRUE)
```

---

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, summarise
```

The following object is masked from 'package:purrr':

`compact`

Attaching package: 'magrittr'

The following object is masked from 'package:purrr':

`set_names`

The following object is masked from 'package:tidyverse':

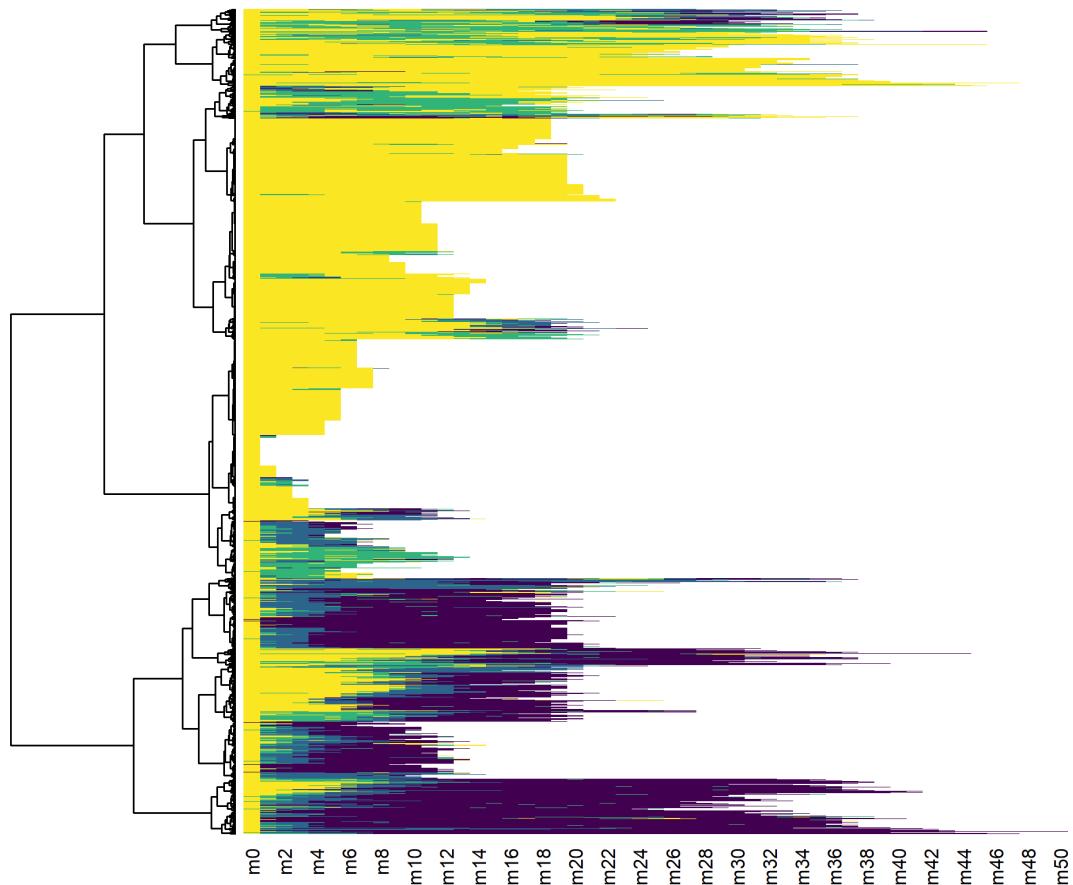
```
extract
```

```
Attaching package: 'JLutils'
```

```
The following object is masked from 'package:formattable':
```

```
percent
```

```
R> seq_heatmap(seq_all, arbre_all)
```



Il apparaît que les différentes séquences sont principalement regroupées en fonction de leur longueur. En effet, pour passer d'une séquence courte à une séquence longue il faut «ajouter» des statuts pour compléter la séquence ce qui induit *de facto* une distance élevée (en raison du coût *indel*). Dès lors, lorsque l'on travaille avec des séquences aux longueurs très disparates, une classification ascendante hiérarchique va produire une typologie de séquences courtes et de séquences longues, ce qui n'est pas forcément ce que l'on recherche.

Dans notre exemple, nous pouvons considérer que les séquences courtes ne sont pas pertinentes à retenir dans l'analyse car l'observation n'est pas assez longue pour voir le parcours de soins des patients. Une solution consiste à ne retenir que les individus observés au moins *n* mois et analyser leur trajectoire sur seulement *n* mois, ce qui permet de n'avoir que des séquences de même longueur. Dès lors, la distance entre deux séquences ne dépendra plus que des différences de parcours. On serait tenté de prendre un *n* élevé pour avoir ainsi des parcours de soins longs. Mais dans ce cas là, l'analyse ne se fera que sur un tout petit nombre d'individus et on manquera de puissance. Si, à l'inverse, on prends un *n* petit, nous aurons des effectifs élevés mais les séquences seront peut-être trop courtes pour mettre en évidence la variété des trajectoires. Il faut dès lors trouver un compromis entre ces deux contraintes.

Si l'on regarde notre premier graphique montrant le nombre d'observations au cours du temps, il apparaît une sorte de point d'inflexion au niveau de M18 avec un brusque décrochage. D'un autre côté, 18 mois offre un minimum de durée d'observations pour espérer voir émerger des trajectoires plus complexes.

## Une seconde analyse de séquences limitées aux 18 premiers mois

Reprendons notre analyse en se limitant aux individus observés au moins 18 mois (soit 19 status entre M0 et M18) et en se limitant aux 18 premiers mois pour modéliser les séquences. La fonction `seqlength` permet de récupérer la longueur de chaque séquence.

```
R> large$seq_length <- seqlength(seq_all)
large_m18 <- large[seq_length >= 19, id:m18]
seq_m18 <- seqdef(
  large_m18[, m0:m18],
  id = large_m18$id,
  alphabet = c("D", "C", "T", "S"),
  states = c("diagnostiqué", "en soins", "sous traitement", "inf. contrôlée"),
  cpal = viridis(4, direction = -1)
)
```

[>] state coding:

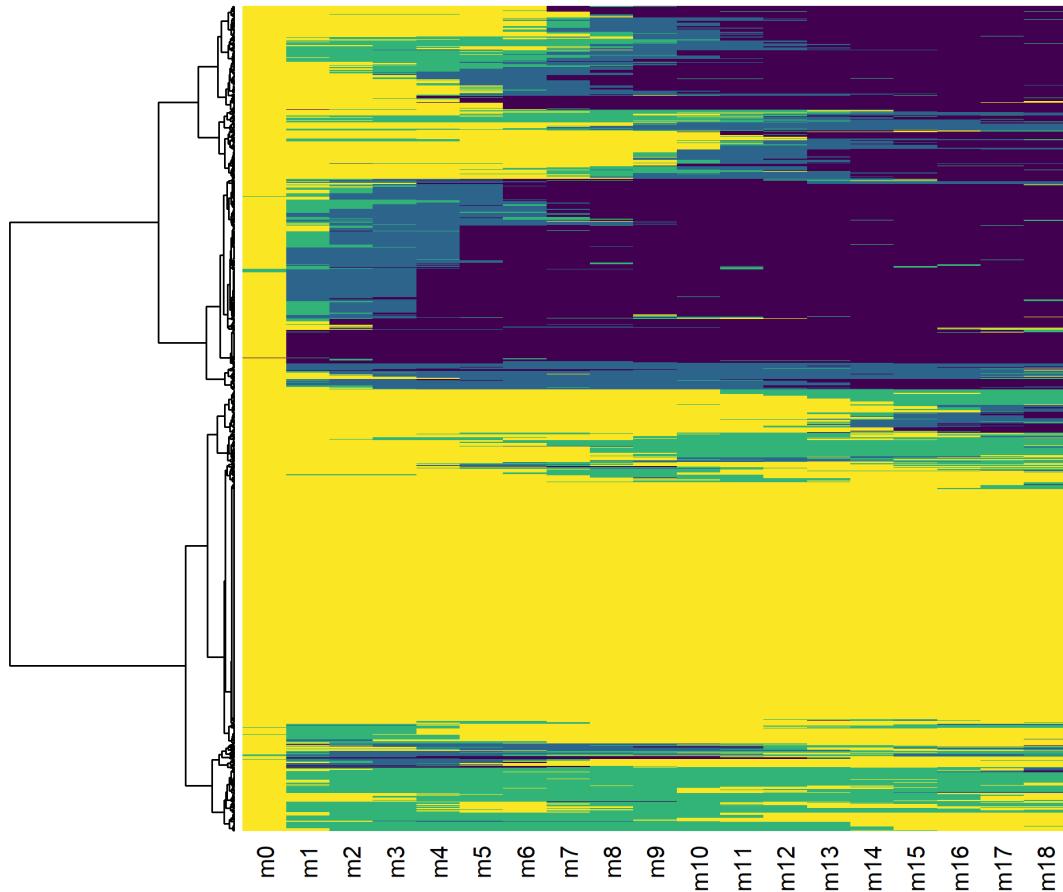
[alphabet]	[label]	[long label]
------------	---------	--------------

1	D	diagnostiqué	diagnostiqué
2	C	en soins	en soins
3	T	sous traitement	sous traitement
4	S	inf. contrôlée	inf. contrôlée
[>] 1317 sequences in the data set			
[>] min/max sequence length: 19/19			

```
R> dist_m18 <- seqdist(seq_m18, method = "OM", sm = couts$sm, indel = couts$indep)
```

[>]	1317 sequences with 4 distinct states
[>]	checking 'sm' (size and triangle inequality)
[>]	578 distinct sequences
[>]	min/max sequence lengths: 19/19
[>]	computing distances using the OM metric
[>]	elapsed time: 0.4 secs

```
R> arbre_m18 <- hclust(as.dist(dist_m18), method = "ward.D2")
seq_heatmap(seq_m18, arbre_m18)
```



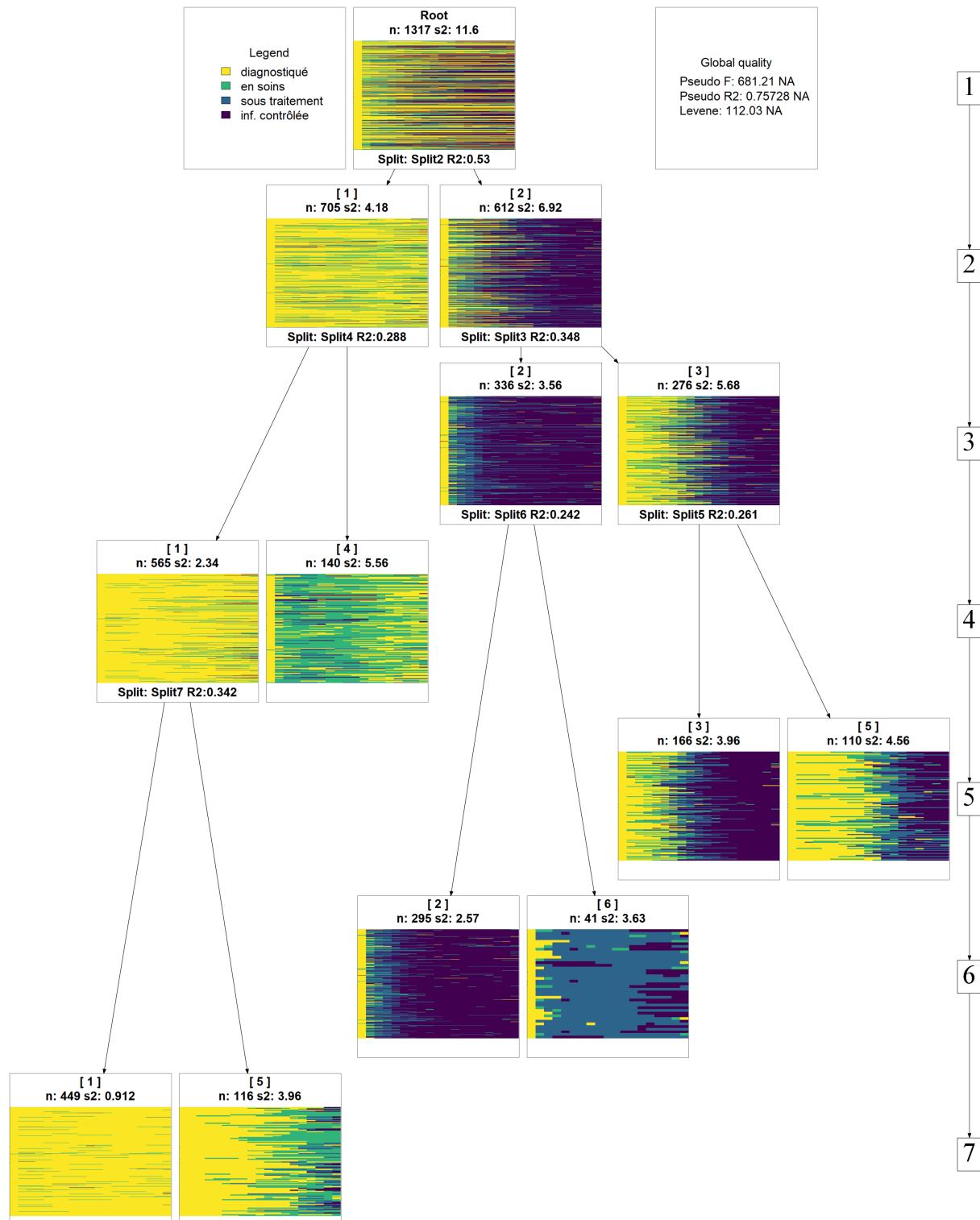
Reste maintenant à décider du nombre de classes à retenir. Encore une fois, c'est un équilibre à trouver entre le niveau de détails voulus et le niveau de simplification requis pour permettre l'analyse.

Pour faciliter ce choix, on peut avoir recours à la fonction `as.seqtree` de l'extension **WeightedCluster**, couplée à la fonction `seqtreedisplay`. **ATTENTION** : pour que le graphique puisse être produit, il faut que le logiciel libre **GraphViz** (<https://graphviz.gitlab.io/>) soit installé sur votre PC. On peut également installer **GraphViz** avec le code ci-dessous :

```
R> if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install("Rgraphviz")
```

La combinaison des deux fonctions va permettre de représenter l’évolution des catégories au fur-et-à-mesure que l’on coupe le dendrogramme plus bas. On peut choisir le type de graphique utilisé avec l’argument `type` (voir l’aide de `seqplot`) et le nombre maximum de clusters avec `nclust`.

```
R> library(WeightedCluster, quietly = TRUE)
seqtree_m18 <- as.seqtree(arbre_m18, seqdata = seq_m18, diss = dist_m18, ncluster = 7)
seqtreedisplay(seqtree_m18, type="I", border=NA, show.depth=TRUE)
```



Représentation graphique produite par seqtreedisplay

Afin d'éviter de multiplier les sous-groupes, nous n'allons conserver que 4 catégories.

```
R> large_m18$typo_cah <- cutree(arbre_m18, 4)
```

### NOTE

Il est aussi possible de se baser sur divers indicateurs statistiques sur la «qualité» de chaque partition. Pour cela, on pourra par exemple avoir recours à la fonction `as.clustrange` de l'extension `WeightedCluster`. Essayez par exemple les commandes ci-après :

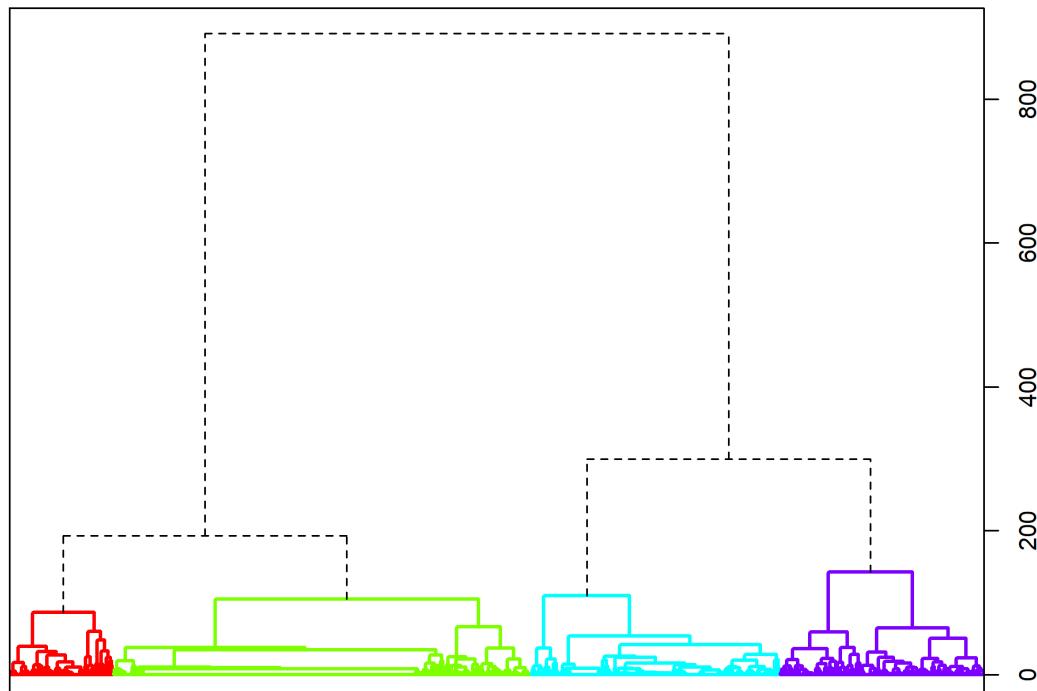
```
R> nc <- as.clustrange(arbre_m18, dist_m18)
  summary(nc, max.rank = 3)
  plot(nc, norm = "zscore")
```

Pour plus d'informations, voir le [manuel de la librairie WeightedCluster](#), chapitre 7.

On peut représenter le découpage du dendrogramme avec `A2Rplot` fournie par `JLutils`

```
R> A2Rplot(arbre_m18, k = 4, show.labels = FALSE)
```

**Colored Dendrogram ( 4 groups)**



Comme expliqué par Matthias Studer dans le [manuel de la librairie WeightedCluster](#), plusieurs critiques peuvent être adressées aux procédures hiérarchiques, en particulier le fait que la fusion de deux groupes se fait en maximisant un critère local.

L’algorithme PAM pour [Partitioning Around Medoids](#) suit une autre logique que les algorithmes hiérarchiques et vise à obtenir la meilleure partition d’un ensemble de données en un nombre prédéfini de groupes. Il a l’avantage de maximiser un critère global et non uniquement un critère local. Par contre, le nombre de classes doit être fixé à l’avance.

Ayant décidé de retenir 4 classes au regard de notre classification ascendante hiérarchique, nous pouvons voir si l’algorithme PAM permet d’améliorer nos 4 classes. Nous allons utiliser la fonction [wCKMedoids](#) de l’extension [WeightedCluster](#) en lui indiquant comme partition initiale celle obtenue avec la classification hiérarchique.

```
R> pam_m18 <- wCKMedoids(dist_m18, k = 4, initialclust = arbre_m18)
    large_m18$typo_pam <- pam_m18$clustering
```

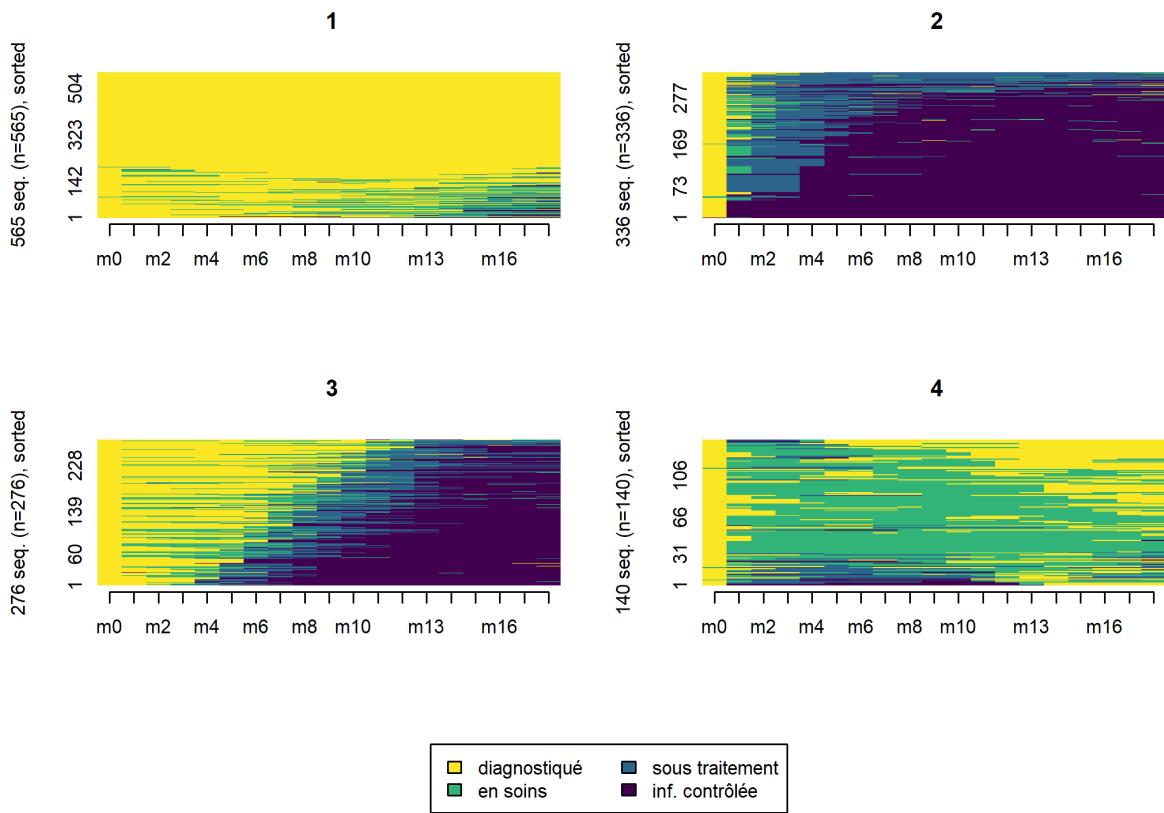
Un tableau croisé nous permet de voir que les deux typologies restent proches.

```
R> table(large_m18$typo_cah, large_m18$typo_pam)
```

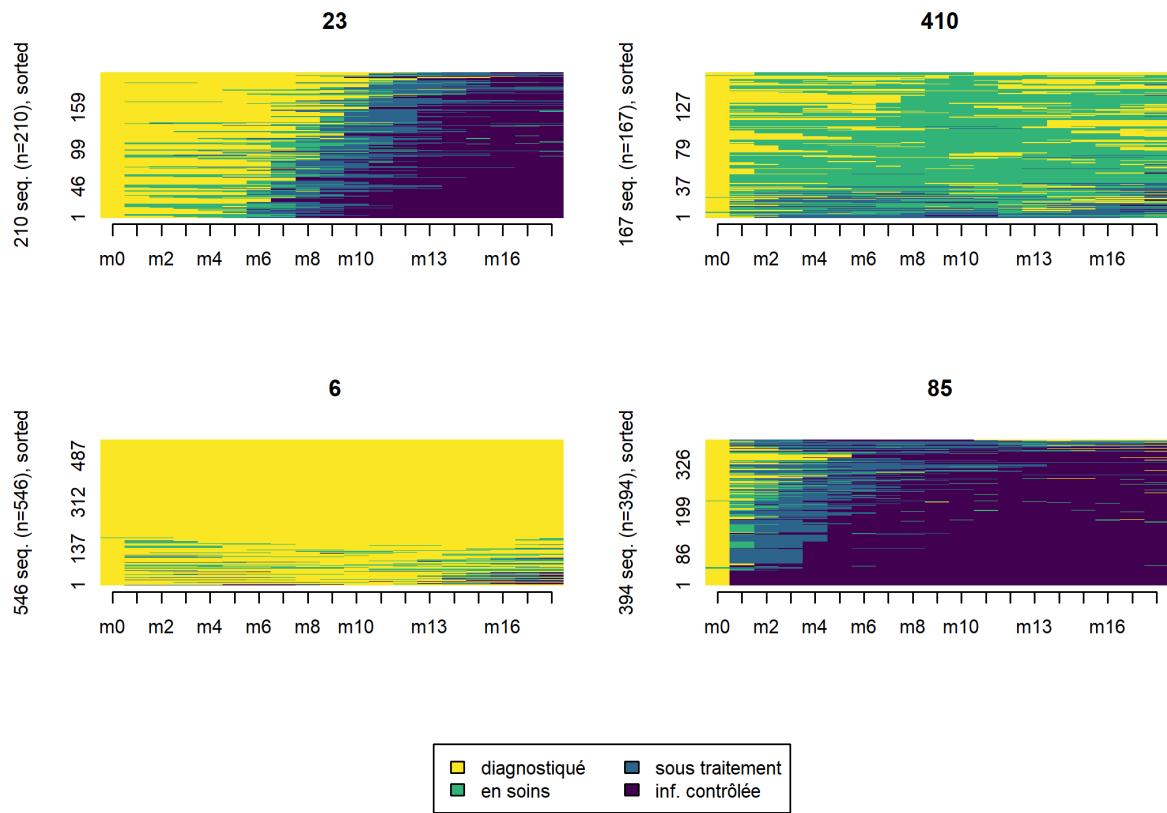
/	6	23	85	410
1	522	4	0	39
2	0	3	330	3
3	1	203	59	13
4	23	0	5	112

Regardons les tapis de séquence des deux typologies.

```
R> large_m18$ordre_cmd <- cmdscale(as.dist(dist_m18), k = 1)
  seqIplot(seq_m18, group = large_m18$typo_cah, sortv = large_m18$ordre_cmd)
```



```
R> seqIplot(seq_m18, group = large_m18$typo_pam, sortv = large_m18$ordre_cmd)
```



Comme on le voit les deux typologies obtenues sont très proches. Suivant le cas, à vous de choisir celle qui semble la plus pertinente d'un point de vue sociologique. Il existe également divers indicateurs statistiques pour mesurer la qualité d'une partition (voir le [manuel de la librairie WeightedCluster](#) de Matthias Studer). Ils peuvent être calculés avec la fonction `wcClusterQuality`. Comparons les deux typologies obtenues.

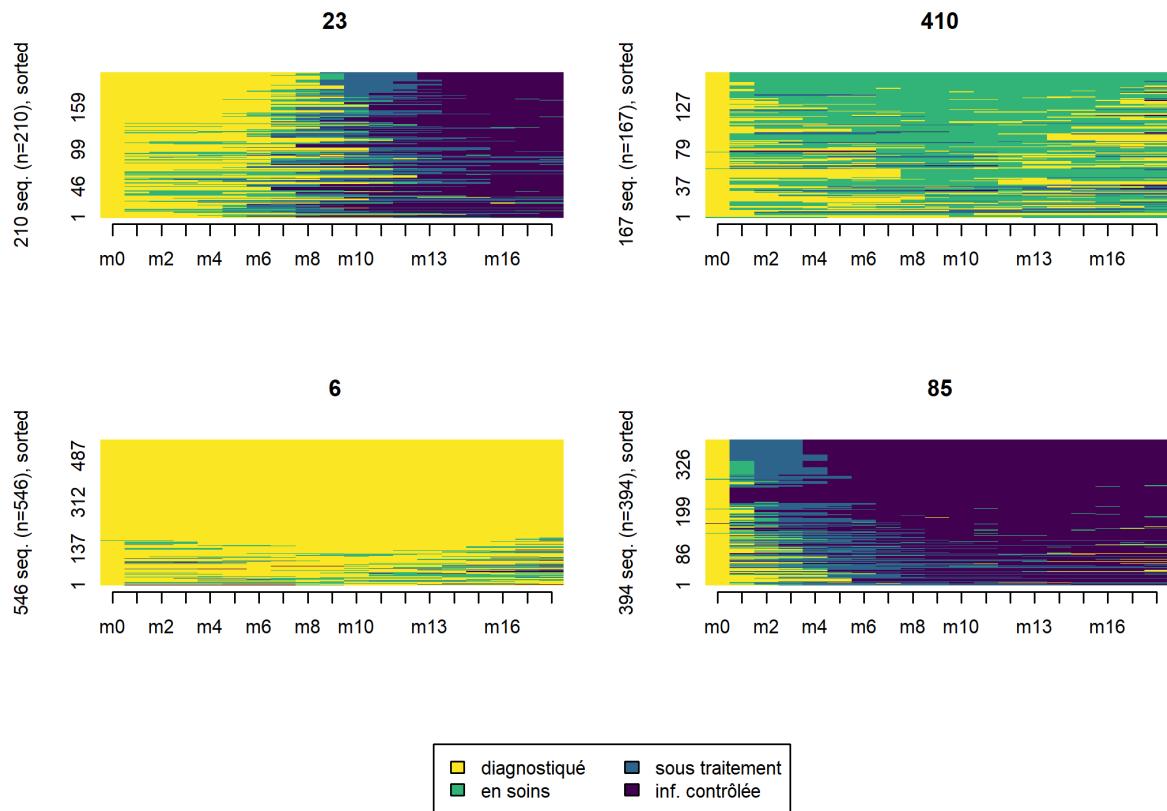
```
R> tab <- tibble(
  stat = names(wcClusterQuality(dist_m18, large_m18$typo_cah)$stats),
  cah = wcClusterQuality(dist_m18, large_m18$typo_cah)$stats,
  pam = wcClusterQuality(dist_m18, large_m18$typo_pam)$stats
)
gt:::gt(tab) %>% gt:::fmt_number(2:3, decimals = 2, sep_mark = " ")
```

stat	cah	pam
PBC	0.70	0.71
HG	0.93	0.95
HGSD	0.93	0.95
ASW	0.53	0.57
ASWw	0.53	0.57
CH	938.97	1 023.12
R2	0.68	0.70
CHsq	3 351.21	3 980.60
R2sq	0.88	0.90
HC	0.03	0.02

Selon ces indicateurs calculés, l'approche PAM obtiendrait une partition légèrement de meilleure qualité que celle obtenue par CAH.

L'extension **WeightedCluster** fournit aussi une fonction `wcSilhouetteObs` permettant de mesurer la «silhouette» de chaque séquence. Plus cette métrique est élevée et proche de 1, plus la séquence est proche du centre de classe et «caractéristique» de la classe. On peut utiliser cette métrique pour classer les séquences sur le tapis de séquences.

```
R> large_m18$sil <- wcSilhouetteObs(dist_m18, large_m18$typo_pam)
seqIplot(seq_m18, group = large_m18$typo_pam, sortv = large_m18$sil)
```



Nous voyons émerger quatre groupes distincts :

- les *rapides*, qui entrent en soins et initient le traitement dès les premiers mois suivant le diagnostic ;
- les *lents*, qui entrent en soins et initient le traitement plus tardivement, après M6 ;
- les *inaboutis*, qui entrent en soins mais n'initient pas le traitement ;
- les *hors soins*, qui ne sont pas entrés en soins où n'y sont pas restés.

Le graphique obtenu avec `seqIplot` affiche visuellement chaque groupe avec la même hauteur, pouvant laisser accroire que chaque groupe a le même poids dans l'échantillon. Pour produire une représentation graphique des tapis de séquences plus «correcte», où chaque la hauteur de chaque groupe correspondrait à son poids dans l'échantillon, nous allons passer par `ggplot2`. Un tapis de séquences peut-être vu comme un «raster» et dès lors représenté avec `geom_raster`. Pour travailler avec `ggplot2`, nos données doivent être au format `tidy`, c'est-à-dire avec une ligne par point d'observation, soit une ligne par personne et par jour. Nous allons donc repartir du fichier `care_trajectories`. Le mois d'observation indiquera la position en abscisse. Quant à la position en ordonnée, il faudra que nous la calculions, séparément pour chaque groupe, afin d'éviter des «lignes vides» dans le graphique.

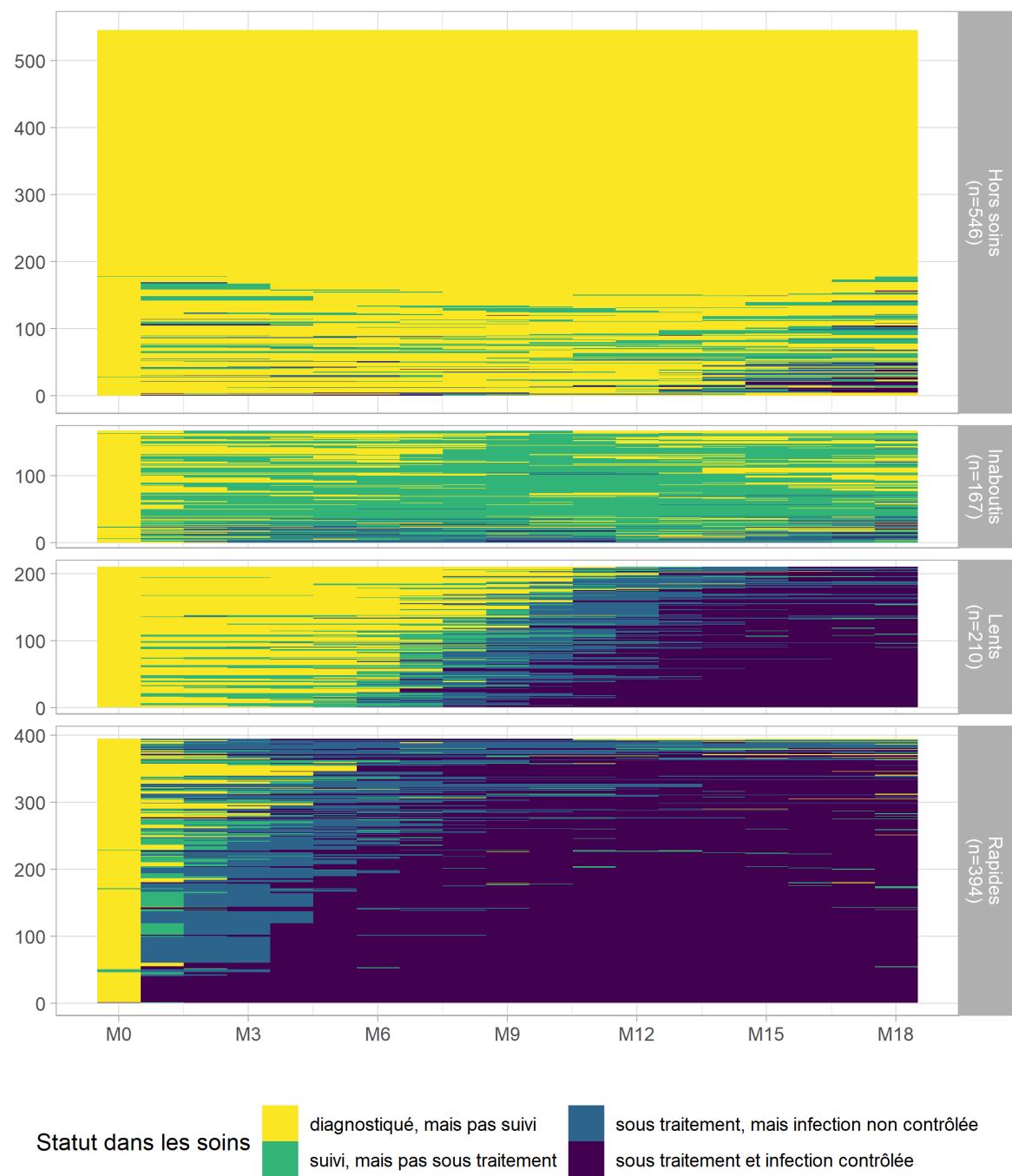
```
R> # nommer les groupes
large_m18$groupe <- factor(
  large_m18$typo_pam,
  c(85, 23, 410, 6),
  c("Rapides", "Lents", "Inaboutis", "Hors soins")
)

# calculer le rang des individus dans chaque groupe
setorder(large_m18, "ordre_cmd")
large_m18[, rang_cmd := 1:N, by = groupe]

# créer un fichier long
long_m18 <- care_trajectories[id %in% large_m18$id & month <= 18]
long_m18 <- merge(
  long_m18,
  large_m18[, .(id, groupe, rang_cmd)],
  by = "id",
  all.x = TRUE
)
long_m18$care_statusF <- to_factor(long_m18$care_status)

# calculer les effectifs par groupe
tmp <- large_m18[, .(n = N), by = groupe]
tmp[, groupe_n := paste0(groupe, "\n(n=", n, ")")]
long_m18 <- merge(
  long_m18,
  tmp[, .(groupe, groupe_n)],
  by = "groupe",
  all.x = TRUE
)
```

```
R> # graphique des tapis de séquences
ggplot(long_m18) +
  aes(x = month, y = rang_cmd, fill = care_statusF) +
  geom_raster() +
  facet_grid(groupe_n ~ ., space = "free", scales = "free") +
  scale_x_continuous(breaks = 0:6*3, labels = paste0("M", 0:6*3)) +
  scale_y_continuous(breaks = 0:5*100, minor_breaks = NULL) +
  xlab("") + ylab("") +
  theme_light() +
  theme(legend.position = "bottom") +
  labs(fill = "Statut dans les soins") +
  scale_fill_viridis(discrete = TRUE, direction = -1) +
  guides(fill = guide_legend(nrow = 2))
```



## Facteurs associés à l'appartenance à chaque groupe

Une fois les différents groupes de trajectoires identifiés, il est courant de vouloir regarder si certains

facteurs influencent l'appartenance à un groupe plutôt qu'un autre. Dans nos données d'exemple, nous nous intéresserons aux variables suivantes : sexe, groupe d'âges et niveau d'éducation.

Ces différentes variables sont renseignées pour chaque mois dans le tableau de données `care_trajectories`, en tenant compte des éventuels changements au cours du temps. Ici, notre analyse est menée au niveau individuel. Nous allons donc récupérer la valeur de ces différentes variables au moment du diagnostic, à savoir à M0.

```
R> large_m18 <- merge(
  large_m18,
  care_trajectories[
    month == 0,
    .(id, sex, age, education)
  ],
  by = "id",
  all.x = TRUE
)
```

La fonction `tbl_summary` de l'extension `gtsummary` permet de produire aisément une série de tableaux croisés. À noter le recours à `add_p()` pour ajouter les p-valeurs au test du Chi<sup>2</sup>.<sup>1</sup> La fonction `add_overall` permet d'ajouter une colonne avec l'ensemble de l'échantillon. Notez que nous avons utiliser `unlabelled` de l'extension `labelled` pour convertir en amont les vecteurs labellisés en facteurs.

---

1. Pour plus de détails sur les tableaux croisés et le test du Chi<sup>2</sup>, voir le chapitre sur la statisque bivariée, page 381 et le chapitre sur les tests de comparaison, page 493.

```
R> library(gtsummary)
unlabelled(large_m18) %>%
 tbl_summary(by = "groupe", include = c("groupe", "sex", "age", "education")) %>%
  add_p() %>%
  add_overall(last = TRUE)
```

Caractéristique	Rapides, N = 394 <sup>1</sup>	Lents, N = 210 <sup>1</sup>	Inaboutis, N = 167 <sup>1</sup>	Hors soins, N = 546 <sup>1</sup>	p-value <sup>2</sup>	Total, N = 1 317 <sup>1</sup>
Sexe						<0,001
homme	104 (26%)	61 (29%)	42 (25%)	243 (45%)		450 (34%)
femme	290 (74%)	149 (71%)	125 (75%)	303 (55%)		867 (66%)
Âge						<0,001
16-29	96 (24%)	71 (34%)	62 (37%)	235 (43%)		464 (35%)
30-59	269 (68%)	128 (61%)	91 (54%)	281 (51%)		769 (58%)
60+	29 (7,4%)	11 (5,2%)	14 (8,4%)	30 (5,5%)		84 (6,4%)
Education						0,002
primaire	63 (16%)	47 (22%)	30 (18%)	123 (23%)		263 (20%)
secondaire	126 (32%)	83 (40%)	61 (37%)	212 (39%)		482 (37%)
supérieur	205 (52%)	80 (38%)	76 (46%)	211 (39%)		572 (43%)

<sup>1</sup> Statistique présentée: n (%)<sup>2</sup> Test statistique réalisé: test du khi-deux d'indépendance

Une manière de présenter ces mêmes données de manière plus visuelle consiste à réaliser un diagramme en barres cumulées (voir le chapitre sur les graphiques bivariés, page 450). Ici, nous utilisons une boucle

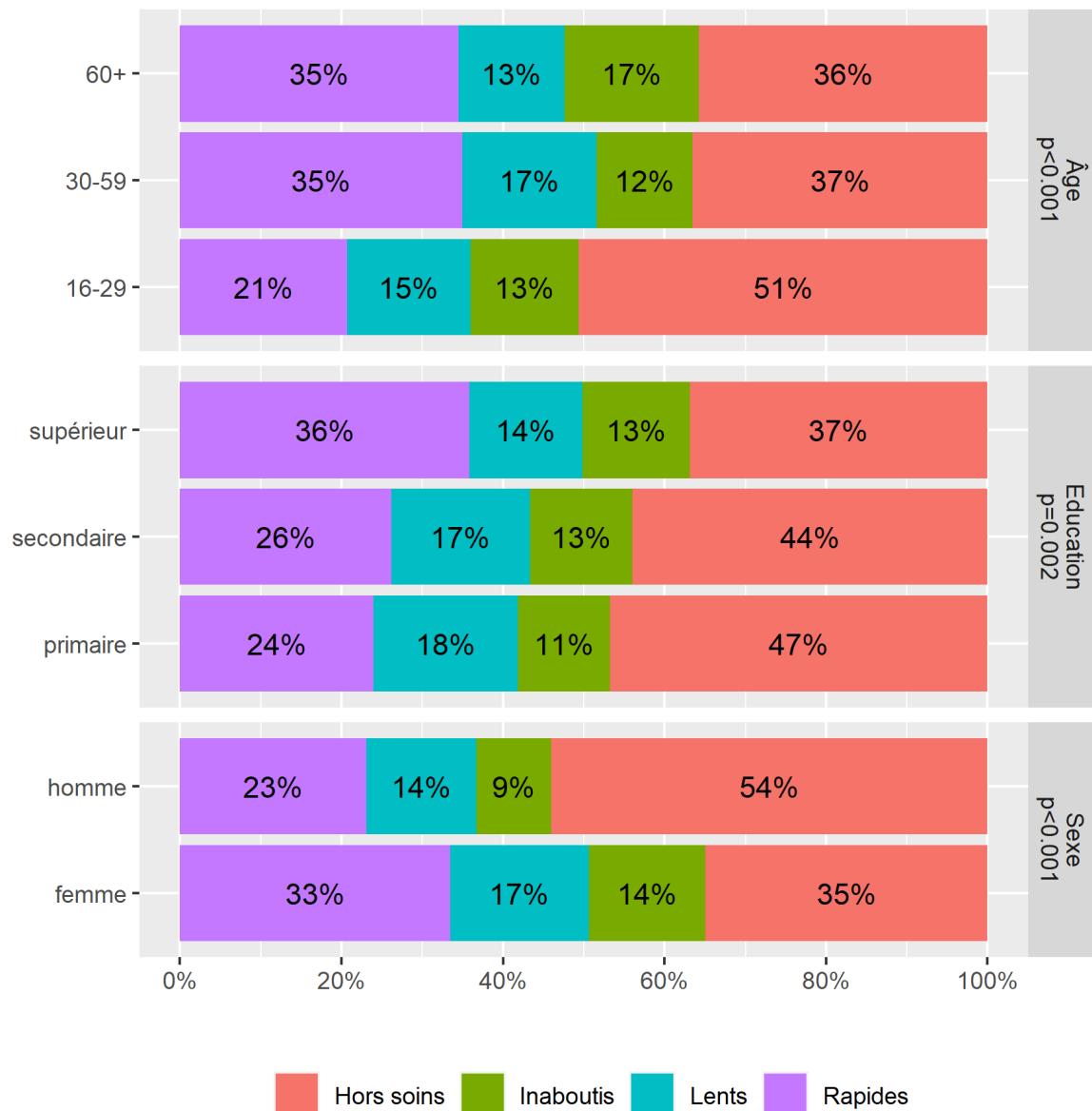
`for` (voir le chapitre sur les structures conditionnelles, page 945) pour calculer les différents tableaux croisés et les fusionner avec `bind_rows` (voir la section concaténation de tables, page 0 du chapitre dédié à `dplyr`). Nous en profitons également pour calculer le test du Chi<sup>2</sup> et l'afficher avec le nom de la variable sur le graphique.

Note : pour afficher les proportions sur le graphique, aurons recours recours à la statistique `stat_prop` de l'extension `GGally`.

```
R> res <- tibble()
  explanatory <- c(
    "sex" = "Sexe",
    "age" = "Âge",
    "education" = "Education"
  )
  for (v in names(explanatory)) {
    tmp <- tibble:::as_tibble(table(large_m18$groupe, to_factor(large_m18[[v]])), .name_repair = "unique")
    names(tmp) <- c("groupe", "level", "n")
    test <- chisq.test(large_m18$groupe, to_factor(large_m18[[v]]))
    tmp$var <- paste0(
      explanatory[v],
      "\n",
      scales::pvalue(test$p.value, add_p = TRUE)
    )
    res <- bind_rows(res, tmp)
  }

# stat_prop() a besoin d'un facteur
res$level <- factor(res$level)

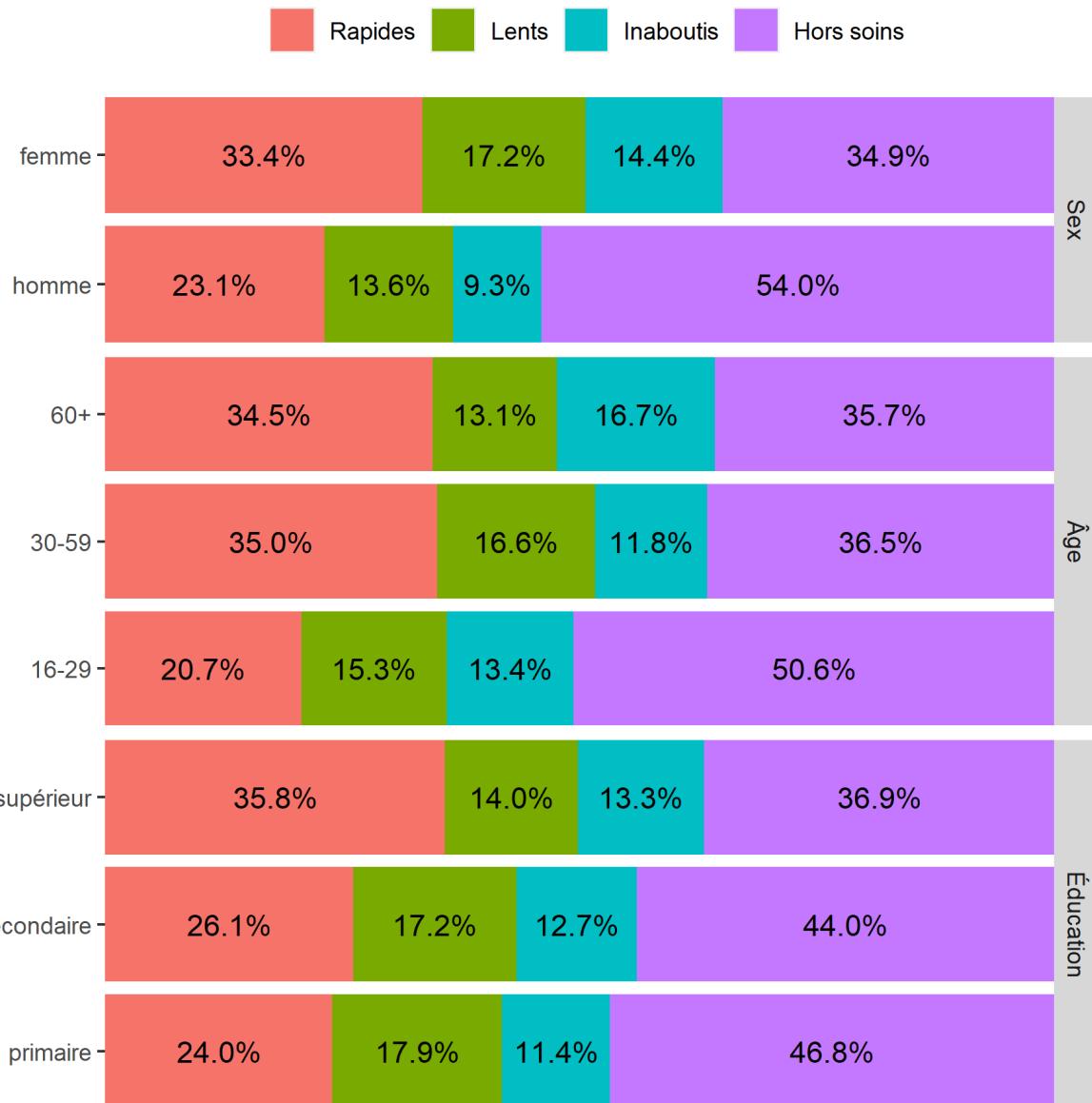
library(GGally)
ggplot(res) +
  aes(x = level, fill = groupe, weight = n) +
  geom_bar(position = "fill") +
  geom_text(
    aes(by = level, label = scales::percent(..prop.., accuracy = 1)),
    stat = "prop", position = position_fill(.5)
  ) +
  facet_grid(var ~ ., scales = "free", space = "free") +
  scale_y_continuous(labels = scales::percent, breaks = 0:5/5) +
  coord_flip() +
  theme(legend.position = "bottom") +
  xlab("") + ylab("") + labs(fill = "")
```



Un graphique similaire peut s'obtenir très facilement en ayant recours à la fonction `ggbivariate` de l'extension **GGally**<sup>2</sup>.

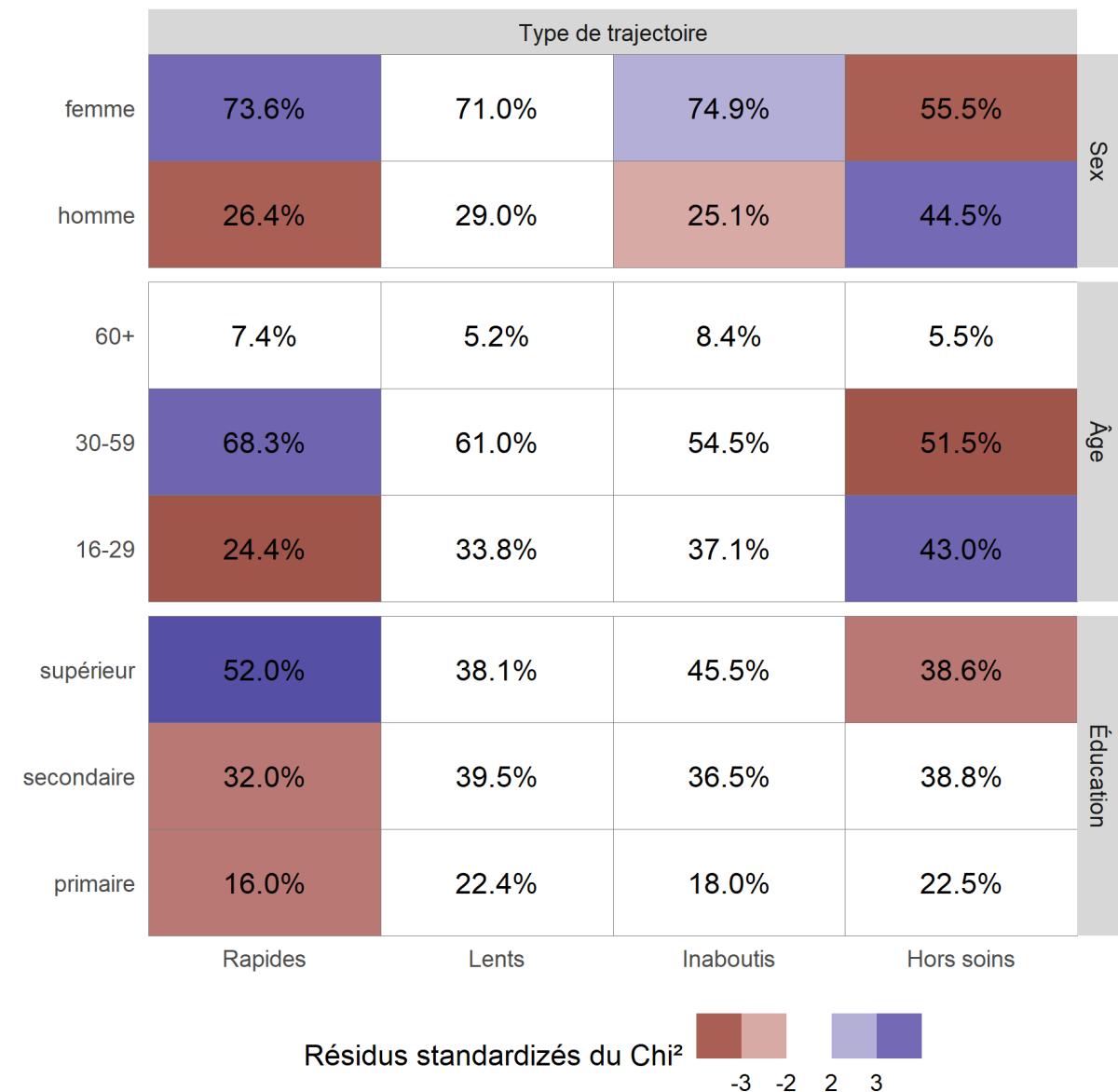
2. À noter, `ggbivariate` accepte également des variables continues.

```
R> library(GGally)
ggbivariate(
  unlabelled(large_m18),
  outcome = "groupe",
  explanatory = c("sex", "age", "education"),
  columnLabelsY = c("Sex", "Âge", "Éducation")
) + labs(fill = "")
```



Pour mieux visualiser les relations entre les variables, on peut avoir recours à la fonction `ggtabule` de l'extension **GGally** qui permet de représenter les résidus du Chi<sup>2</sup>.

```
R> library(GGally)
ggtable(
  unlabelled(large_m18),
  columnsX = "groupe",
  columnsY = c("sex", "age", "education"),
  cells = "col.prop",
  fill = "std.resid",
  columnLabelsX = "Type de trajectoire",
  columnLabelsY = c("Sex", "Âge", "Éducation"),
  legend = 1
) +
  labs(fill = "Résidus standardisés du Chi2") +
  theme(legend.position = "bottom")
```



On considère qu'une cellule est surreprésentée si son résidu standardisé du Chi<sup>2</sup> est supérieur à 2 ou 3, et qu'elle est sous-représentée si le résidu est inférieur à -2 ou -3.

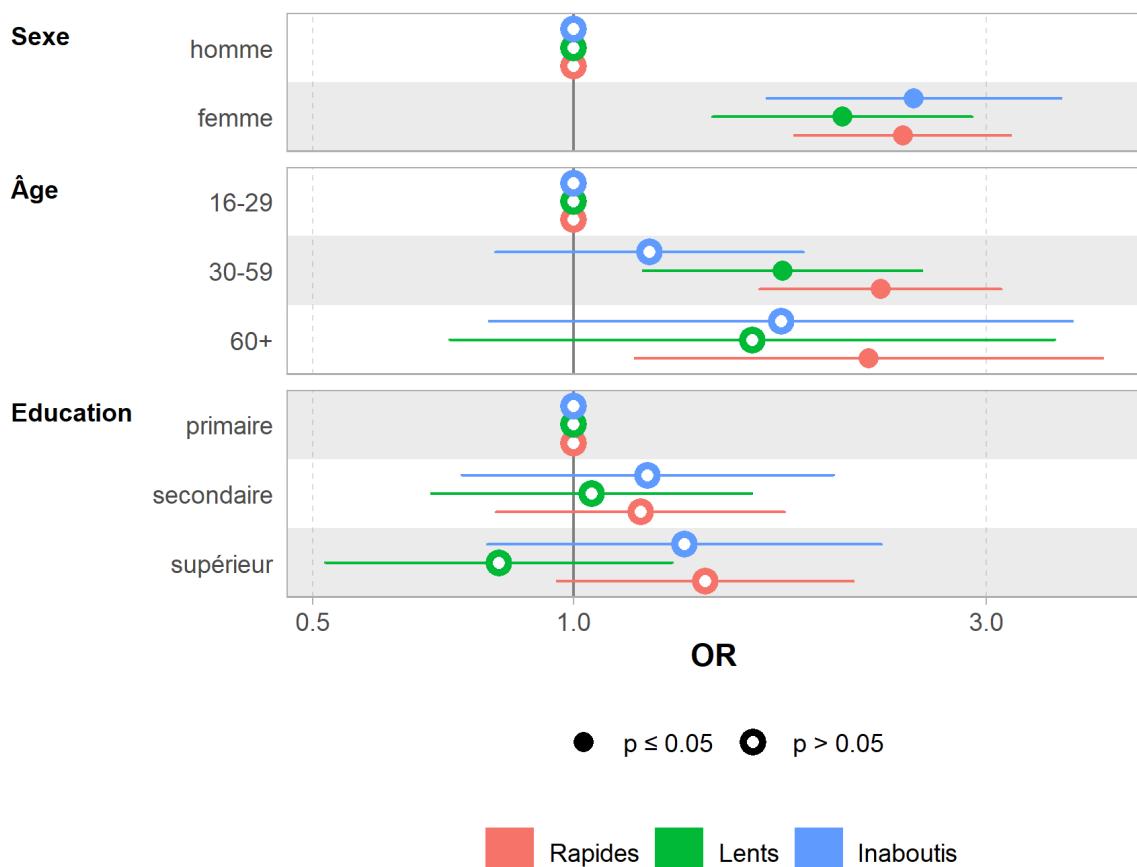
On peut ainsi noter que les hommes, les jeunes et ceux vivant à plus de 10 kilomètres d'une clinique sont plus souvent dans le groupe «Hors soins». Inversement, les femmes et les plus éduqués sont plus souvent dans le groupe des «Rapides». Résultat inattendu, les ménages les plus riches sont moins souvent dans les groupes ayant initiés un traitement («Rapides» et «Lents»), ce résultat pouvant s'expliquer en partie par le fait que les plus aisés peuvent plus facilement accéder à des soins dans le secteur privé, et donc «faussement» apparaître «Hors soins», car seuls les soins reçus dans le secteur public ont été mesurés dans cette étude.»»

Pour affiner les résultats, on aura recours à un modèle multivarié en exécutant une régression logistique

multinomiale avec `multinom` de l'extension `nnet` (pour plus de détails, voir le chapitre dédié, page 0).

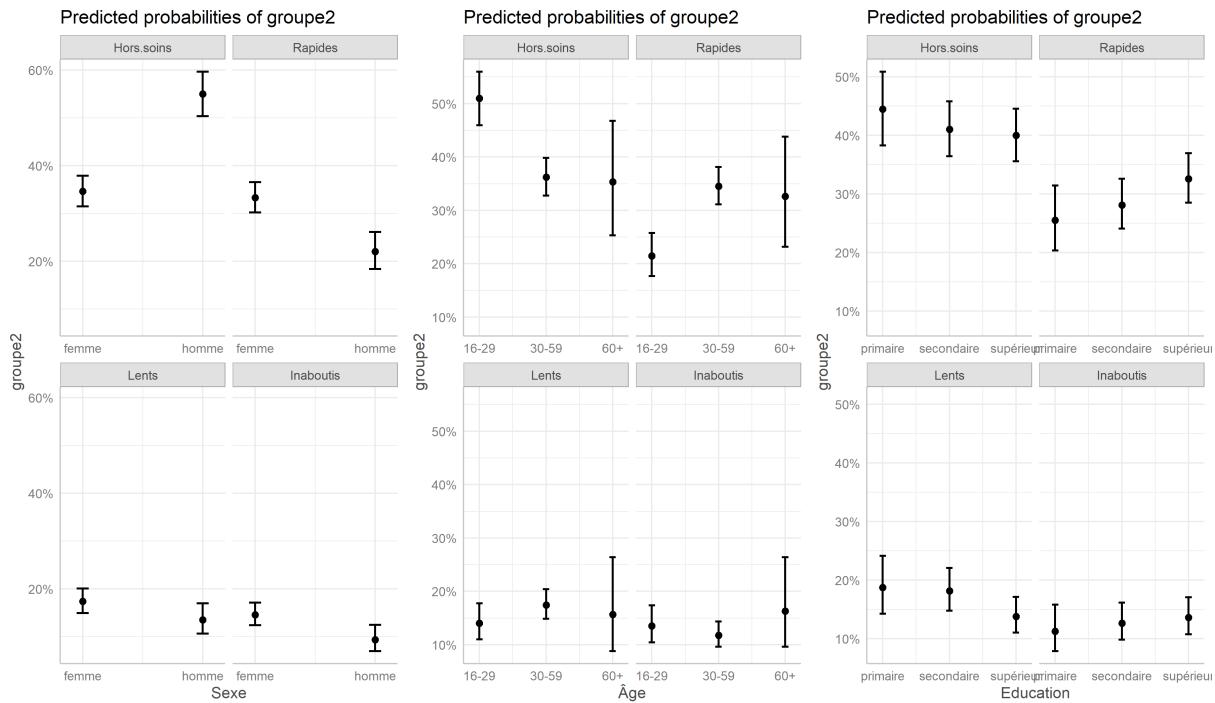
```
R> library(nnet)
  large_m18$groupe2 <- relevel(large_m18$groupe, "Hors soins")
  regm <- multinom(
    groupe2 ~ sex + age + education,
    data = to_factor(large_m18)
  )
```

```
R> library(GGally)
  ggcoef_multinom(regm, exponentiate = TRUE)
```



Nous pouvons représenter les effets des variables du modèle avec la fonction `ggeffect` de `ggeffects`.

```
R> library(ggeffects)
cowplot:::plot_grid(plotlist = plot(ggeffect(regm)), ncol = 3)
```



## Modèle mixte à classes latentes

Un autre type d'approche envisageable pour identifier des classes de trajectoires est celle des modèles mixtes à classes latentes. Ce type de modèles peut prendre en compte une grande variété d'indicateurs, continus, binaires ou ordinaux. On peut y intégrer des co-variables et il n'est pas nécessaire de disposer du même nombre d'observations par individu.

Nous n'aborderons que brièvement ici ce type de modèles complexes. Sous R, ils peuvent être réalisés via l'extension [lcmm](#) et sa fonction homonyme [lcmm](#).

Commençons par préparer les données.

```
R> care_trajectories$num_status <- as.integer(to_factor(care_trajectories$care_status))
care_trajectories[, sexF := to_factor(sex)]
care_trajectories[, ageF := to_factor(age)]
care_trajectories[, educationF := to_factor(education)]
```

Ici, nous allons modéliser le statut dans les soins en fonction du temps. Il faut indiquer au modèle, via le paramètre `ng` le nombre de groupes ou classes latentes souhaité. Ici, nous avons retenu 4 en lien

avec les résultats de notre analyse de séquences. L'argument `link = "thresholds"` permet d'indiquer que notre variable d'intérêt est ordinaire. Les modèles `lcmm` peuvent également prendre en compte des variables continues.

**Attention :** le temps de calcul de ce type de modèle peut être long (plusieurs heures dans notre exemple), suivant le nombre de paramètres, le nombre d'observations et la puissance de votre machine.

```
R> library(lcmm)
mod4 <- lcmm(
  num_status ~ month, random = ~ month, subject = 'id',
  mixture = ~ month, ng = 4, iddiag = TRUE, data = care_trajectories,
  link = "thresholds"
)
```

Voyons comment se présentent les résultats.

```
R> summary(mod4)
```

```
General latent class mixed model
fitted by maximum likelihood method

lcmm(fixed = num_status ~ month, mixture = ~month, random = ~month,
      subject = "id", ng = 4, iddiag = TRUE, link = "thresholds",
      data = care_trajectories)

Statistical Model:
  Dataset: care_trajectories
  Number of subjects: 2929
  Number of observations: 49365
  Number of latent classes: 4
  Number of parameters: 15
  Link function: thresholds

Iteration process:
  Convergence criteria satisfied
  Number of iterations: 52
  Convergence criteria: parameters= 5.2e-10
                        : likelihood= 5.4e-07
                        : second derivatives= 3.5e-07

Goodness-of-fit statistics:
  maximum log-likelihood: -26612.74
  AIC: 53255.48
  BIC: 53345.22

Discrete posterior log-likelihood: -26612.74
```

```
Discrete AIC: 53255.48
```

```
Mean discrete AIC per subject: 9.0911  
Mean UACV per subject: 9.1045  
Mean discrete LL per subject: -9.0859
```

Maximum Likelihood Estimates:

Fixed effects in the class-membership model:  
(the class of reference is the last class)

	coef	Se	Wald	p-value
intercept class1	0.00832	0.04503	0.185	0.85334
intercept class2	-0.42002	0.12990	-3.233	0.00122
intercept class3	-0.02992	0.09675	-0.309	0.75712

Fixed effects in the longitudinal model:

	coef	Se	Wald
intercept class1 (not estimated)	0		
intercept class2	-1.49825	0.08163	-18.355
intercept class3	-0.41418	0.05228	-7.923
intercept class4	-2.50344	0.09211	-27.179
month class1	0.21757	0.00332	65.447
month class2	0.01032	0.00506	2.039
month class3	0.17720	0.00338	52.373
month class4	0.00449	0.00512	0.876

p-value

intercept class1 (not estimated)	0.00000
intercept class2	0.00000
intercept class3	0.00000
intercept class4	0.00000
month class1	0.00000
month class2	0.04142
month class3	0.00000
month class4	0.38090

Variance-covariance matrix of the random-effects:

	intercept	month
intercept	4.39542	
month	0.00000	0.12653

Residual standard error (not estimated) = 1

Parameters of the link function:

	coef	Se	Wald	p-value
thresh. parm1	0.94803	0.04753	19.945	0.00000
thresh. parm2	1.10727	0.00652	169.922	0.00000
thresh. parm3	1.08980	0.00727	149.816	0.00000

On dispose d'un AIC et d'un BIC. Ainsi, une stratégie possible pour déterminer le nombre de classes consiste à calculer un modèle différent pour chaque nombre de classes envisagé puis à retenir le modèle ayant le plus faible AIC ou BIC.

Pour chaque observation, le modèle a calculé la probabilité qu'elle appartienne à chacune des 4 classes identifiées. La fonction `postprob` fournit des statistiques sur cette classification.

```
R> postprob(mod4)
```

```
Posterior classification:
  class1 class2 class3 class4
N 844.00 153.00 424.00 1508.00
% 28.82  5.22 14.48 51.49

Posterior classification table:
--> mean of posterior probabilities in each class
    prob1  prob2  prob3  prob4
class1 0.5945 0.0735 0.2476 0.0844
class2 0.0924 0.7257 0.0881 0.0938
class3 0.1576 0.1103 0.6561 0.0760
class4 0.1522 0.2052 0.1865 0.4561

Posterior probabilities above a threshold (%):
  class1 class2 class3 class4
prob>0.7 23.58 54.90 38.68  6.90
prob>0.8 15.88 48.37 30.19  4.97
prob>0.9 11.26 33.99 21.93  3.51
```

Les classes et les probabilités d'appartenance à chacune sont disponibles aisément.

```
R> head(mod4$pprob)
```

<b>id</b>	<b>class</b>	<b>prob1</b>	<b>prob2</b>	<b>prob3</b>	<b>prob4</b>
3	4	0.1686629	0.2218569	0.2016242	0.4078560
9	4	0.2123759	0.2051987	0.2329009	0.3495245
13	4	0.1590213	0.2260205	0.1879610	0.4269972
15	1	0.9992106	0.0000000	0.0000004	0.0007890
18	1	0.5616344	0.0472977	0.3478822	0.0431858
21	4	0.1686629	0.2218569	0.2016242	0.4078560

Récupérons la classe dans notre fichier de données.

```
R> care_trajectories <- merge(
  care_trajectories,
  mod4$pprob %>% dplyr::select(id, mod4_class = class),
  by = "id",
  all.x = TRUE
)
```

Améliorons les intitulés des classes et ajoutons le nombre d'individu par classes.

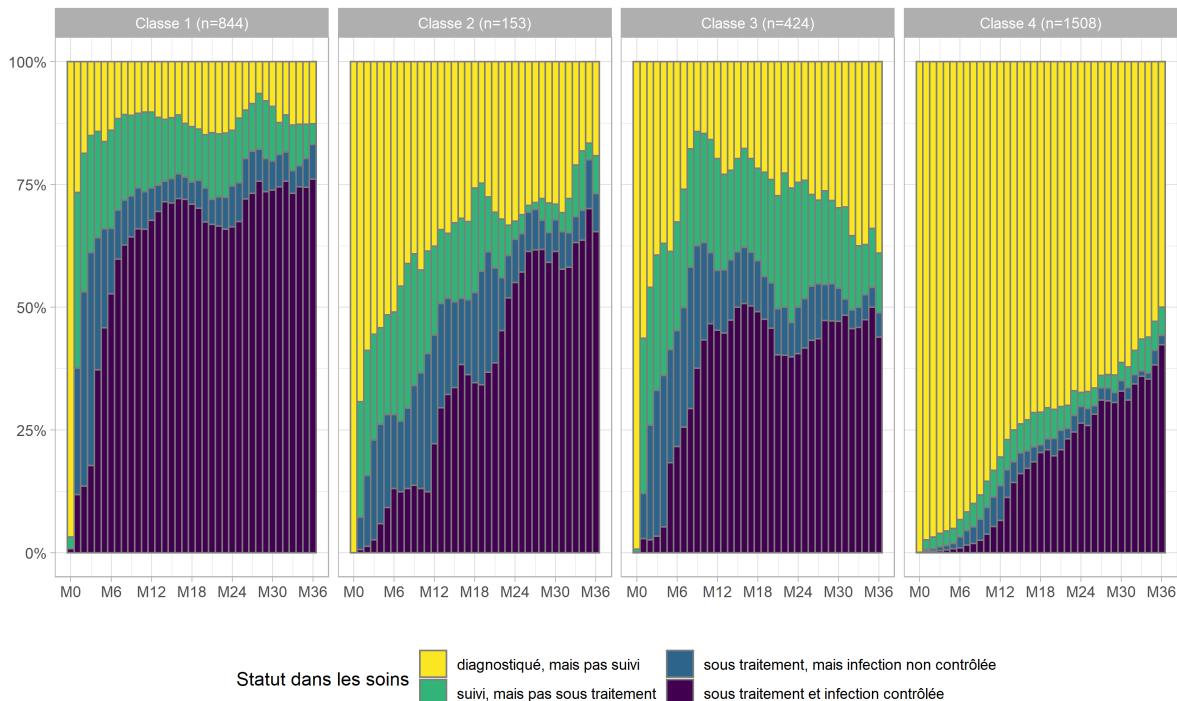
```
R> n_par_classe <- table(mod4$pprob$class)
n_par_classe
```

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>
	844	153	424	1508

```
R> care_trajectories$mod4_class2 <- factor(  
  care_trajectories$mod4_class,  
  levels = 1:4,  
  labels = paste0(  
    "Classe ",  
    1:4,  
    " (n=",  
    n_par_classe,  
    ")")  
)  
)
```

Représentons la cascade observée dans chacune de ces classes.

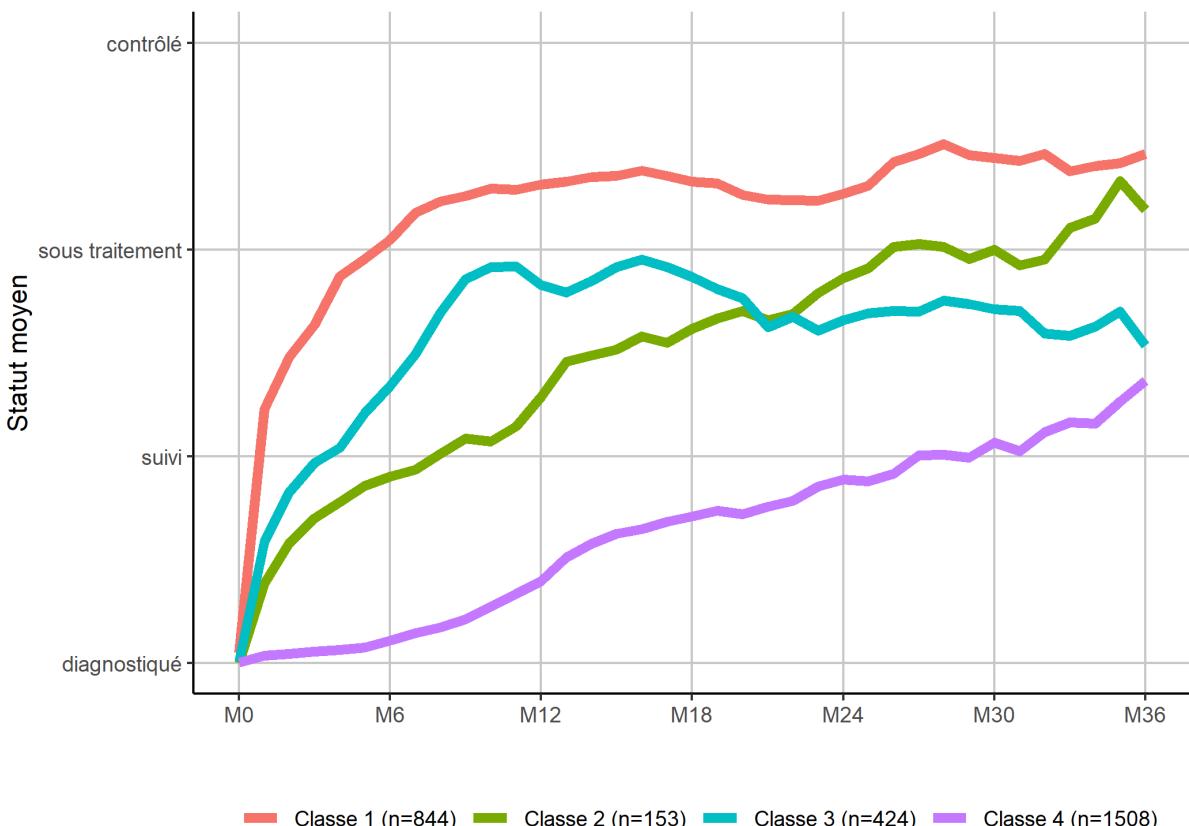
```
R> care_trajectories$care_statusF <- to_factor(care_trajectories$care_status)
ggplot(care_trajectories[month <= 36]) +
  aes(x = month, fill = care_statusF) +
  geom_bar(color = "gray50", width = 1, position = "fill") +
  scale_x_continuous(breaks = 0:6*6, labels = paste0("M", 0:6*6)) +
  scale_y_continuous(labels = scales::percent) +
  xlab("") + ylab("") +
  theme_light() +
  theme(legend.position = "bottom") +
  labs(fill = "Statut dans les soins") +
  scale_fill_viridis(discrete = TRUE, direction = -1) +
  guides(fill = guide_legend(nrow = 2)) +
  facet_grid(~ mod4_class2)
```



Une mani re alternative de pr senter les classes consiste   repr senter chaque mois, non pas la distribution dans chaque  tat, mais un «  tat moyen » en consid rant que le statut dans les soins peut  tre assimil    un score allant de 1   4.

```
R> moyennes_mensuelles <- care_trajectories[
  month <= 36,
  .(status_moyen = mean(num_status)),
  by = .(month, mod4_class2)
]
```

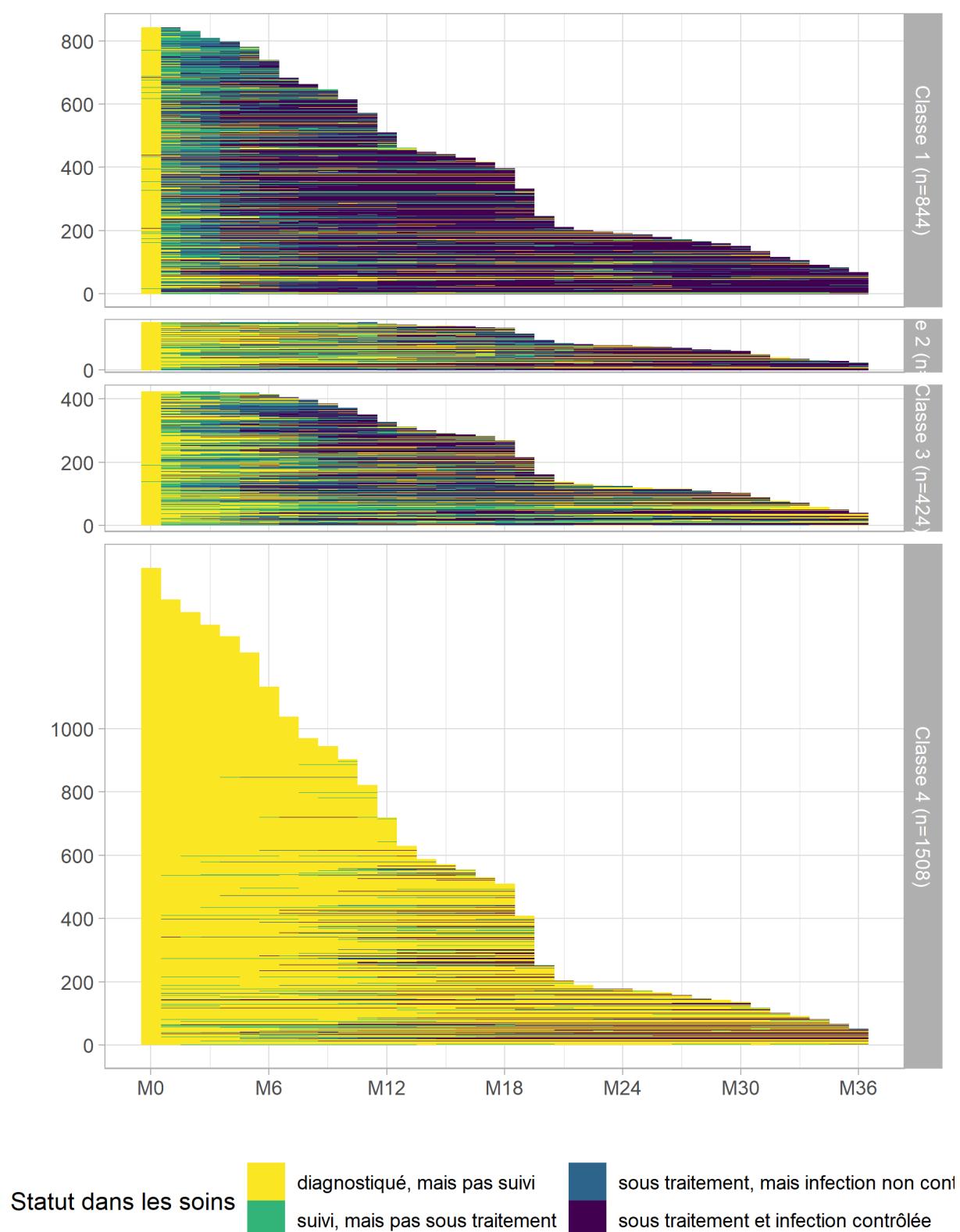
```
R> ggplot(moyennes_mensuelles) +
  aes(x = month, y = status_moyen, color = mod4_class2) +
  geom_line(size = 2) +
  scale_x_continuous(breaks = 0:6*6, labels = paste0("M", 0:6*6)) +
  scale_y_continuous(
    breaks = 1:4, limits = c(1, 4),
    labels = c("diagnostiqué", "suivi", "sous traitement", "contrôlé")
  ) +
  xlab("") + ylab("Statut moyen") + labs(color = "") +
  theme_classic() +
  theme(
    legend.position = "bottom",
    panel.grid.major = element_line(colour = "grey80")
  )
```



Il faut cependant rester vigilant, dans la mesure où ce type de représentation synthétique peut masquer la diversité des trajectoires sous-jacentes, notamment en termes de durée ou d'enchaînement des événements. Même si cela est peut-être plus difficile à lire, il est toujours bon de regarder les tapis de séquences.

```
R> care_trajectories[, tmp_rang := as.integer(fct_infreq(factor(id))), by = mod4_class]
```

```
R> ggplot(care_trajectories[month <= 36]) +
  aes(x = month, y = tmp_rang, fill = care_statusF) +
  geom_raster() +
  facet_grid(mod4_class2 ~ ., space = "free", scales = "free") +
  scale_x_continuous(breaks = 0:6*6, labels = paste0("M", 0:6*6)) +
  scale_y_continuous(breaks = 0:5*200, minor_breaks = NULL) +
  xlab("") + ylab("") +
  theme_light() +
  theme(legend.position = "bottom") +
  labs(fill = "Statut dans les soins") +
  scale_fill_viridis(discrete = TRUE, direction = -1) +
  guides(fill = guide_legend(nrow = 2))
```



## Modèle à observations répétées

Pour prendre en considération l'ensemble des observations présentes (sans se limiter aux individus observés au moins sur une certaine période), nous pouvons avoir recours à un modèle à observations répétées.

Il s'agit de modèles classiques sauf qu'au lieu de considérer une ligne par individu, nous allons intégrer dans le modèle une ligne par individu et par pas de temps. Dans la mesure où nous avons plusieurs observations pour une même personne, cela doit être pris en compte par l'ajout d'un effet aléatoire dans le cadre d'un modèle mixte ou en ayant recours à un (voir le chapitre sur les modèles à effets aléatoires).

Vue la nature de notre variable d'intérêt (plusieurs modalités ordonnées), nous aurons recours à une régression logistique ordinale (voir le chapitre dédié, page 0). Pour un modèle mixte ordinal on peut utiliser la fonction `clmm` de l'extension `ordinal`. Pour un modèle GEE ordinal, citons `ordgee` de l'extension `geepack` ou encore `ordLORgee` de `multgee`. Il importe également que la dimension temporelle soit inclue dans les variables du modèle.

Ici, nous allons utiliser `ordgee`. Il nous faut tout d'abord transformer notre variable d'intérêt en un facteur ordonné.

```
R> care_trajectories[, care_statusF := to_factor(care_status, ordered = TRUE)]
```

Nous allons transformer nos variables explicatives en facteurs. Pour le temps, dans la mesure où l'effet n'est pas forcément linéaire, nous allons l'intégrer en tant que variable catégorielle. Par contre, comme nous n'avons que très peu d'observations individuelles après 3 ans, nous ne prendrons en compte que les observations des 36 premiers mois. Nous allons aussi retirer les observations à M0 puisqu'à ce moment précis tous les individus sont dans la même situation (diagnostiqués mais pas en soins.)

```
R> ct36 <- care_trajectories[month > 0 & month <= 36, ]
  ct36[, sexF := to_factor(sex)]
  ct36[, ageF := to_factor(age)]
  ct36[, educationF := to_factor(education)]
  ct36[, monthF := to_factor(month)]
```

Calculons notre modèle.

```
R> library(geepack)
mod_td <- ordgee(
  care_statusF ~ sexF + ageF + educationF + monthF,
  data = ct36,
  id = ct36$id
)
```

Les coefficients du modèle s'obtiennent avec `summary`. Malheureusement, il n'existe pas de `tieder` pour ce type de modèle. Nous allons donc procéder manuellement.

```
R> res <- summary(mod_td)$mean
res$term <- rownames(res)
head(res)
```

	estimate	san.se	wald	p	term
Inter:diagnostiqué, mais pas suivi	-2.8569848	0.1272376	504.17899	0.0000000	Inter:diagnostiqué, mais pas suivi
Inter:suivi, mais pas sous traitement	-3.4247822	0.1289357	705.53705	0.0000000	Inter:suivi, mais pas sous traitement
Inter:sous traitement, mais infection non contrôlée	-3.9464056	0.1320344	893.36522	0.0000000	Inter:sous traitement, mais infection non contrôlée
sexFfemme	0.7419722	0.0867777	73.10695	0.0000000	sexFfemme
ageF30-59	0.5576609	0.0890031	39.25822	0.0000000	ageF30-59
ageF60+	0.7147037	0.1930243	13.70970	0.0002133	ageF60+

Les intervalles de confiance à 95% ne sont pas déjà calculés. Faisons-le donc nous même.

```
R> mult <- stats::qnorm((1 + .95) / 2)
res$conf.low <- res$estimate - mult * res$san.se
res$conf.high <- res$estimate + mult * res$san.se
```

Enfin, nous souhaitons disposer des *odds ratios* et non des coefficients bruts. Il faut avoir recours à la fonction `exp` (exponentielle).

```
R> res$estimate <- exp(res$estimate)
  res$conf.low <- exp(res$conf.low)
  res$conf.high <- exp(res$conf.high)
```

Préparons un tableau avec les résultats. Pour le rendre plus lisible, nous allons mettre en forme les *odds ratios* avec une seule décimale. Améliorer le rendu des *p-values* et nous allons utiliser la virgule comme séparateur de décimal, comme il se doit. Nous aurons recours aux fonctions `number` et `pvalue` de l'extension `scales` (voir le chapitre sur la mise en forme des nombres, page 1019).

```
R> tab <- res %>%
  mutate (
    estimate = scales::number(estimate, accuracy = .01, decimal.mark = ","),
    p = scales::pvalue(p, decimal.mark = ","),
    conf.low = scales::number(conf.low, accuracy = .1, decimal.mark = ","),
    conf.high = scales::number(conf.high, accuracy = .1, decimal.mark = ",")
  ) %>%
  dplyr::select(
    Facteur = term, OR = estimate, "p-value" = p,
    "IC 95% bas" = conf.low, "IC 95% haut" = conf.high
  )
```

```
R> knitr::kable(tab, row.names = FALSE, align = "lrrrr")
```

Facteur	OR	p-value	IC 95% bas	IC 95% haut
Inter:diagnostiqué, mais pas suivi	0,06	<0,001	0,0	0,1
Inter:suivi, mais pas sous traitement	0,03	<0,001	0,0	0,0
Inter:sous traitement, mais infection non contrôlée	0,02	<0,001	0,0	0,0
sexFemme	2,10	<0,001	1,8	2,5
ageF30-59	1,75	<0,001	1,5	2,1
ageF60+	2,04	<0,001	1,4	3,0
educationFsecondaire	1,17	0,109	1,0	1,4
educationFsupérieur	1,40	0,002	1,1	1,7
monthF2	1,06	0,147	1,0	1,1
monthF3	1,36	<0,001	1,2	1,5
monthF4	2,79	<0,001	2,5	3,1
monthF5	3,82	<0,001	3,4	4,3
monthF6	4,69	<0,001	4,1	5,4
monthF7	5,63	<0,001	4,9	6,4
monthF8	6,40	<0,001	5,6	7,3
monthF9	7,02	<0,001	6,1	8,1
monthF10	7,49	<0,001	6,5	8,6
monthF11	7,96	<0,001	6,9	9,2
monthF12	8,42	<0,001	7,3	9,8
monthF13	9,31	<0,001	8,0	10,8
monthF14	10,11	<0,001	8,6	11,8
monthF15	10,74	<0,001	9,2	12,6
monthF16	11,46	<0,001	9,8	13,4

Facteur	OR	p-value	IC 95% bas	IC 95% haut
monthF17	11,43	<0,001	9,7	13,4
monthF18	11,53	<0,001	9,8	13,6
monthF19	11,50	<0,001	9,7	13,6
monthF20	11,70	<0,001	9,7	14,1
monthF21	11,67	<0,001	9,6	14,2
monthF22	12,60	<0,001	10,3	15,4
monthF23	12,78	<0,001	10,4	15,7
monthF24	12,76	<0,001	10,4	15,7
monthF25	13,11	<0,001	10,6	16,2
monthF26	13,60	<0,001	11,0	16,9
monthF27	14,40	<0,001	11,6	17,9
monthF28	15,65	<0,001	12,5	19,6
monthF29	14,89	<0,001	11,9	18,7
monthF30	14,98	<0,001	11,9	18,9
monthF31	14,02	<0,001	11,0	17,9
monthF32	14,27	<0,001	11,0	18,5
monthF33	14,41	<0,001	11,0	18,9
monthF34	14,73	<0,001	11,0	19,7
monthF35	16,17	<0,001	11,8	22,1
monthF36	16,07	<0,001	11,4	22,6

On peut facilement représenter tout cela graphiquement. On va supprimer les termes «seuils», grâce à `str_detect` de `stringr`. On notera le recours à `fct_inorder` de `forcats` pour conserver l'ordre des termes selon leur ordre d'apparition.

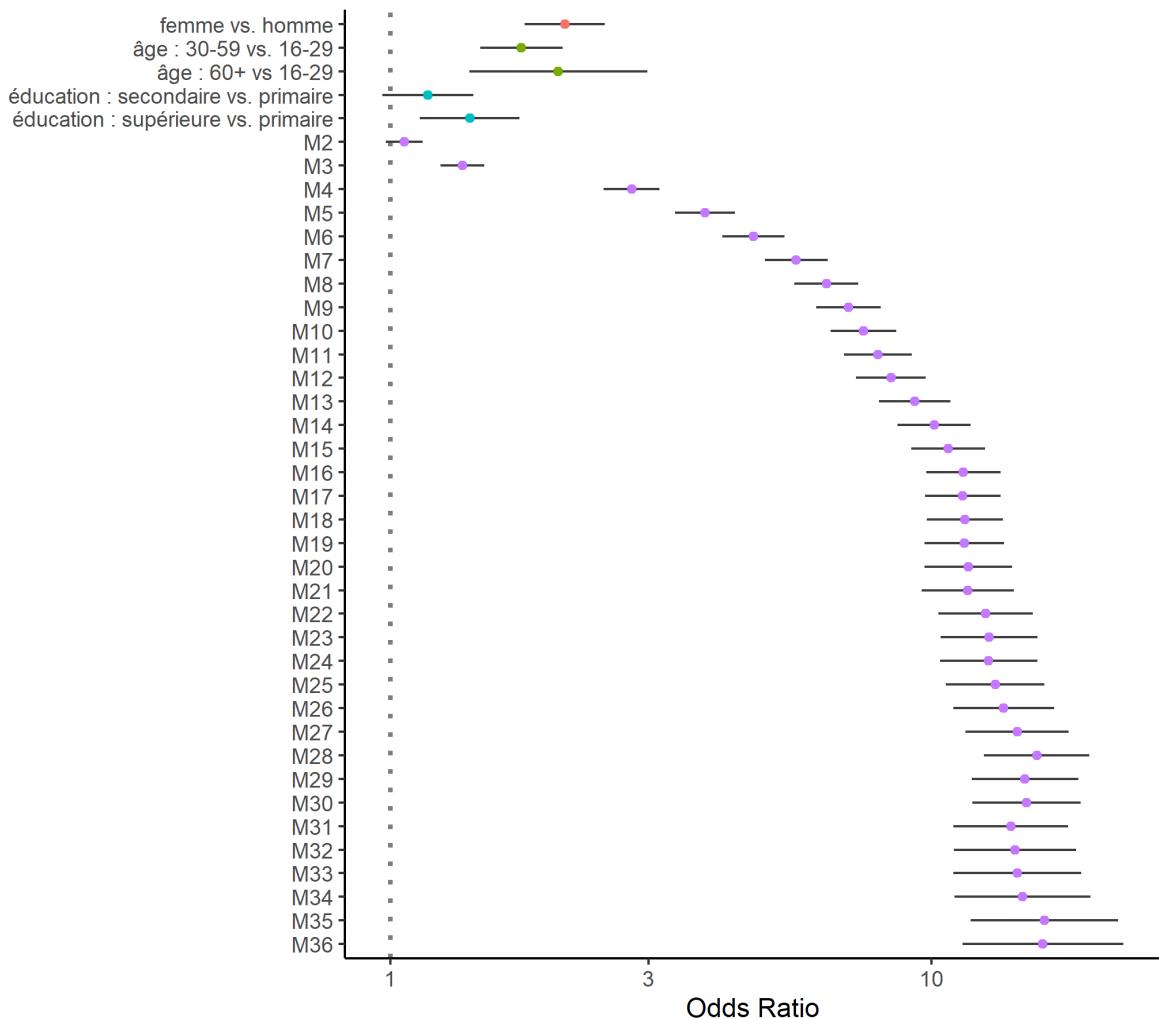
```
R> res <- res[!str_detect(res$term, "Inter"),]
res$term <- fct_inorder(res$term)
res$term <- fct_recode(res$term,
  "femme vs. homme" = "sexFfemme",
  "âge : 30-59 vs. 16-29" = "ageF30-59",
  "âge : 60+ vs 16-29" = "ageF60+",
  "éducation : secondaire vs. primaire" = "educationFsecondaire",
  "éducation : supérieure vs. primaire" = "educationFsupérieur",
  "M2" = "monthF2",
  "M3" = "monthF3",
  "M4" = "monthF4",
  "M5" = "monthF5",
  "M6" = "monthF6",
  "M7" = "monthF7",
  "M8" = "monthF8",
  "M9" = "monthF9",
  "M10" = "monthF10",
  "M11" = "monthF11",
  "M12" = "monthF12",
  "M13" = "monthF13",
  "M14" = "monthF14",
  "M15" = "monthF15",
  "M16" = "monthF16",
  "M17" = "monthF17",
  "M18" = "monthF18",
  "M19" = "monthF19",
  "M20" = "monthF20",
  "M21" = "monthF21",
  "M22" = "monthF22",
  "M23" = "monthF23",
  "M24" = "monthF24",
  "M25" = "monthF25",
  "M26" = "monthF26",
  "M27" = "monthF27",
  "M28" = "monthF28",
  "M29" = "monthF29",
  "M30" = "monthF30",
  "M31" = "monthF31",
  "M32" = "monthF32",
  "M33" = "monthF33",
  "M34" = "monthF34",
  "M35" = "monthF35",
  "M36" = "monthF36")
res$var_group <- c("a", "b", "b", "c", "c", rep("d", 35))

GGally::ggcoef(
  res, exponentiate = TRUE,
```

```

mapping = aes(x = estimate, y = fct_rev(term), color = var_group)
) +
theme_classic() + ylab("") + xlab("Odds Ratio") +
theme(legend.position = "none")

```



Sur ce graphique, on visualise bien l'évolution temporelle traduite par les odds ratios associés à chaque mois, ainsi que les effets globaux de nos covariables : les femmes ont une meilleure progression dans la cascade de soins que les hommes, de même que les plus éduqués et les plus âgés.

## Modèle de survie multi-états

Depuis la fin du XX<sup>e</sup> siècle, de nombreux développements ont réalisés pour étendre les modèles de survie à des processus multi-états. Ces modèles permettent de considérer une grande variété de processus. Plusieurs implémentations existent dans R<sup>3</sup>. Ici, nous allons utiliser l'extension **msm** qui repose sur des

modèles de Markov multi-états et peut prendre en compte des co-variables dans le modèle.

En premier lieu, pour cette extension, il nous faut disposer des données sous une forme longue, c'est-à-dire avec une ligne par individu et point d'observation dans le temps, ce qui est déjà le cas du fichier `care_trajectories`. Les différents status possibles doivent également être codés sous la forme de nombres entiers croissants (ici 1 correspondra à D, 2 à C, 3 à T et 4 à S).

```
R> library(msm)
care_trajectories$status <- as.integer(to_factor(care_trajectories$care_status))
setorder(care_trajectories, id, month)
```

Par ailleurs, nous n'allons concevoir dans l'analyse que les individus avec au moins deux points d'observation, ici ceux observés au moins jusqu'à un mois.

```
R> ct <- care_trajectories[id %in% care_trajectories[month == 1, id]]
```

La fonction `statetable msm` permet de calculer le nombre et le type de transitions observées dans les données.

```
R> statetable msm(status, id, data = ct)
```

from/to	1	2	3	4
1	21916	1168	467	247
2	501	4323	597	210
3	26	141	3489	770
4	33	230	43	12275

Il faut ensuite définir les transitions possibles dans le modèle en faisant une matrice carrée. On indiquera 0 si la transition n'est pas possible, une valeur positive sinon.

---

3. Comme par exemple `msSurv` pour une estimation non-paramétrique.

```
R> tr <- rbind(  
+   c(0, 1, 0, 0), # de 1 : vers 2  
+   c(1, 0, 1, 0), # de 2 : vers 1 ou vers 3  
+   c(0, 1, 0, 1), # de 3 : vers 2 ou vers 4  
+   c(0, 0, 1, 0) # de 4 : vers 3  
)
```

Dans notre matrice de transitions, nous n’avons pas défini de transition directe entre le statut 1 et le statut 3, alors que de telles transitions sont pourtant observées dans notre fichier. En fait, nous pouvons considérer qu’une transition de 1 vers 3 correspond en fait à deux transitions successives, de 1 vers 2 puis de 2 vers 3. La fonction `msm` s’aura identifier d’elle-mêmes ces doubles, voire triples, transitions.

On peut facilement représenter notre matrice de transition sous forme de schéma à l’aide de l’excellente extension **DiagrammeR**.

```
R> library(DiagrammeR)  
mermaid(  
graph TD  
1[diagnostiqué, mais pas suivi]  
2[suivi, mais pas sous traitement]  
3[sous traitement, mais infection non contrôlée]  
4[sous traitement et infection contrôlée]  
  
1--entrée<br />en soins-->2  
2--initiation<br />traitement-->3  
2--sortie<br />de soins-->1  
3--contrôle<br />infection-->4  
3--arrêt<br />traitement-->2  
4--échec<br />virologique-->3  
", height = 300)
```

Il ne nous reste plus qu'à spécifier notre modèle. L'option `obstype = 1` indique à `msm` que nos données correspondent à des *snapshots* à certains moments donnés (ici tous les mois) et donc que les transitions d'un état à un autre ont eu lieu entre nos points d'observation. Les types 2 et 3 correspondent à des dates de transition exactes (voir l'aide la fonction pour plus de détails).

```
R> ms_mod <- msm(
  status ~ month, subject = id, data = ct, qmatrix = tr, obstype = 1
)
```

En exécutant cette commande, vous risquez d'obtenir le message d'erreur suivant :

```
Error in Ccall.msm(params, do.what = "lik", ...) : numerical overflow in calculating likelihood
```

Cela est dû à un problème d'échelle dans l'optimisation du modèle qui génère des nombres plus grands que ce que peut gérer l'ordinateur. Il peut être résolu de la manière suivante. Tout d'abord, on reexécute le modèle avec l'option `control = list(trace = TRUE)`.

```
R> ms_mod <- msm(
  status ~ month, subject = id, data = ct, qmatrix = tr, obstype = 1,
  control = list(trace = TRUE)
)
```

On obtient le message suivant :

```
initial value 74796.800445
Error in Ccall.msm(params, do.what = "lik", ...) : numerical overflow in calculating likelihood
```

Ce qui importe de retenir, c'est la valeur initiale du paramètre d'optimisation avant que l'erreur ne se produise. On va l'utiliser (ou une valeur proche) comme paramètre d'échelle pour l'optimisation avec l'option `fnscale` :

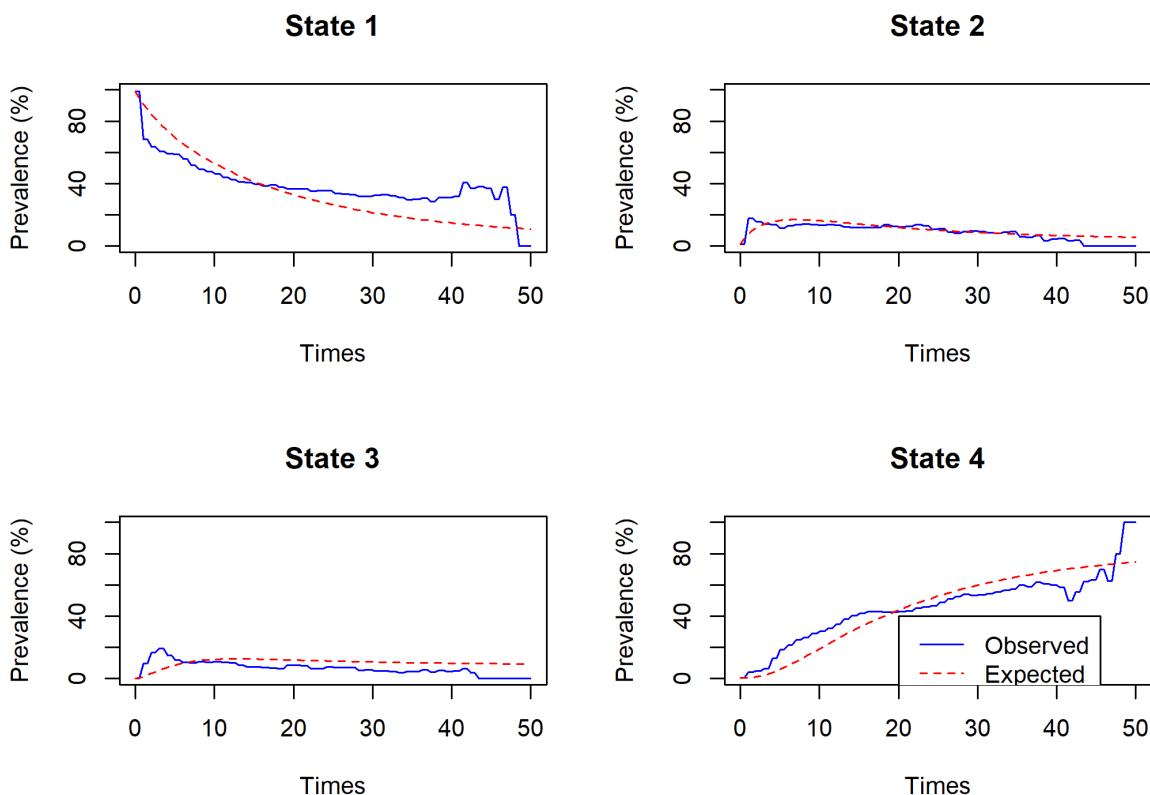
```
R> ms_mod <- msm(
  status ~ month, subject = id, data = ct, qmatrix = tr, obstype = 1,
  control = list(fnscale = 75000, trace = TRUE)
)
```

```
initial value 0.997291
iter 10 value 0.520302
iter 20 value 0.497598
iter 30 value 0.497487
```

```
final value 0.497484
converged
Used 39 function and 37 gradient evaluations
```

On peut comparer la prévalence dans chaque état au cours du temps telle que modélisée par le modèle avec les valeurs observées avec la fonction `plot.prevalence msm`.

```
R> plot.prevalence(ms_mod)
```

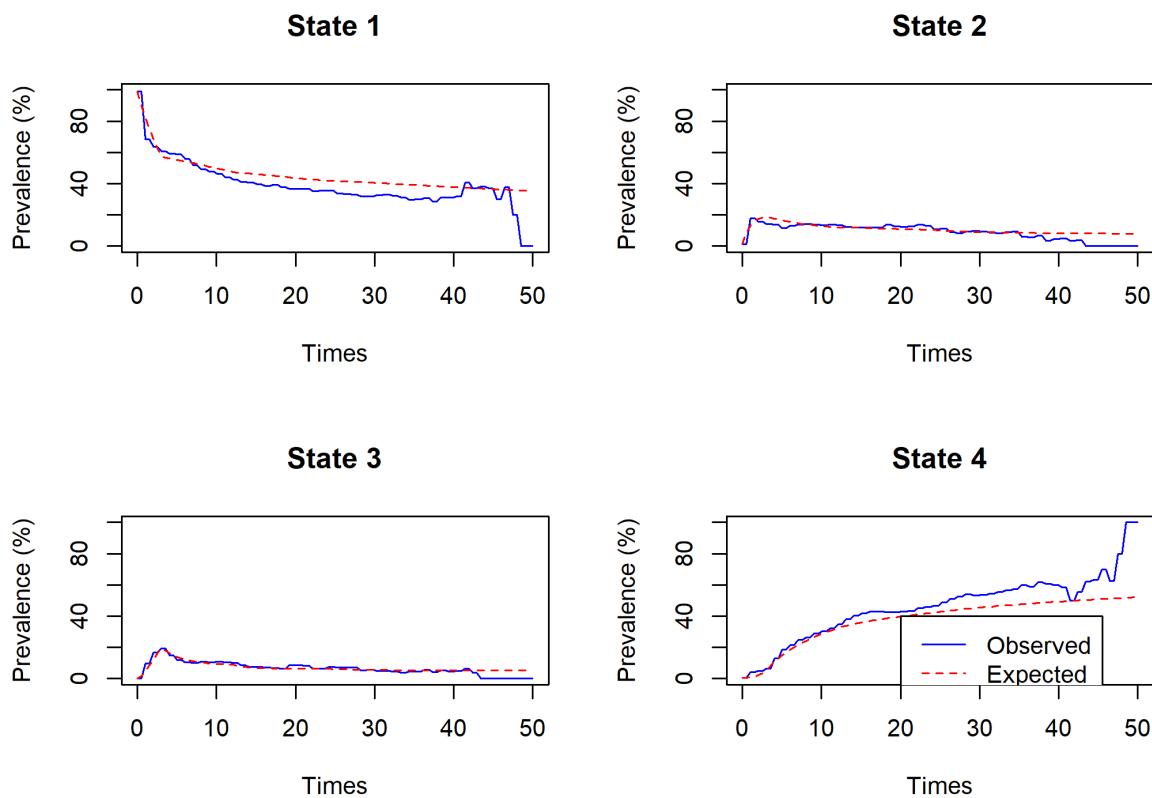


Par défaut, `msm` considère que les intensités de transition d'un état à un autre sont constantes au cours du temps. Or, dans notre exemple, il apparaît que les prévalences observées varient différemment pendant les premiers mois après le diagnostic. Nous allons donc recalculer le modèle en spécifiant avec le paramètre `pci` que nous souhaitons considérer des intensités de transition différentes pour les trois premiers mois, la première année, la seconde année et après la seconde année. Comme il faudra plus d'itérations pour faire converger notre modèle, nous avons également augmenté la valeur du paramètre `maxit` (100 par défaut).

```
R> ms_mod <- msm(
  status ~ month, subject = id, data = ct, qmatrix = tr, obstype = 1,
  pci = c(3, 12, 24),
  control = list(fnscale = 75000, trace = TRUE, maxit = 500)
)
```

Comparons à nouveau les prévalences estimées avec les prévalences observées.

```
R> plot.prevalence.msm(ms_mod)
```



Comme on peut le voir, l'ajustement entre les deux a été amélioré. Les prévalences elles-mêmes peuvent s'obtenir avec `prevalence.msm`.

```
R> prevalence.msm(ms_mod)
```

```
$Observed
  State 1 State 2 State 3 State 4 Total
0      2799      24       0       7   2830
5      1551     292     297     460   2600
```

10	958	257	210	623	2048
15	578	168	104	603	1453
20	283	87	65	327	762
25	192	63	39	267	561
30	147	41	23	245	456
35	69	17	12	136	234
40	14	2	2	27	45
45	7	0	0	12	19
50	0	0	0	1	1

\$Expected

	State 1	State 2	State 3	State 4	Total
0	2799.0000000	24.00000000	0.00000000	7.0000000	2830
5	1436.9517812	424.94481382	358.54921455	379.5541904	2600
10	1017.4844830	258.18412962	191.08425368	581.2471337	2048
15	667.5131754	168.10101158	98.36234055	519.0234724	1453
20	331.9787922	81.73411456	46.82691929	301.4601739	762
25	233.1379082	55.54030396	32.92027747	239.4015104	561
30	184.4057933	40.32826279	24.41822743	206.8477165	456
35	91.3576268	19.57230370	12.12106094	110.9490085	234
40	16.9423183	3.63904521	2.30424907	22.1143875	45
45	6.9062654	1.49869230	0.96896925	9.6260731	19
50	0.3515898	0.07725341	0.05092087	0.5202359	1

\$`Observed percentages`

	State 1	State 2	State 3	State 4
0	98.90459	0.8480565	0.000000	0.2473498
5	59.65385	11.2307692	11.423077	17.6923077
10	46.77734	12.5488281	10.253906	30.4199219
15	39.77977	11.5622849	7.157605	41.5003441
20	37.13911	11.4173228	8.530184	42.9133858
25	34.22460	11.2299465	6.951872	47.5935829
30	32.23684	8.9912281	5.043860	53.7280702
35	29.48718	7.2649573	5.128205	58.1196581
40	31.11111	4.4444444	4.444444	60.0000000
45	36.84211	0.0000000	0.000000	63.1578947
50	0.00000	0.0000000	0.000000	100.0000000

\$`Expected percentages`

	State 1	State 2	State 3	State 4
0	98.90459	0.8480565	0.000000	0.2473498
5	55.26738	16.3440313	13.790354	14.5982381
10	49.68186	12.6066470	9.330286	28.3812077
15	45.94034	11.5692369	6.769604	35.7208171
20	43.56677	10.7262618	6.145265	39.5617026
25	41.55756	9.9002324	5.868142	42.6740660
30	40.43987	8.8439173	5.354874	45.3613413

```

35 39.04172 8.3642323 5.179941 47.4141062
40 37.64960 8.0867671 5.120553 49.1430833
45 36.34877 7.8878542 5.099838 50.6635426
50 35.15898 7.7253409 5.092087 52.0235901

```

Ceci dit, le format dans lequel sont renvoyées les prévalences n'est que peu pratique pour les exploiter ensuite, par exemple avec `ggplot2`. L'extension `JLutils` fournit une fonction expérimentale `tidy.prevalence msm`<sup>4</sup> permettant de transformer ce résultat dans un format `tidy`.

```
R> library(JLutils)
prev <- tidy.prevalence.msm(prevalence.msm(ms_mod, times = 0:36))
head(prev)
```

time	status	observed	expected	observed.percentage	expected.percentage
0	State 1	2799	2799.000	98.90459	98.90459
1	State 1	1942	2329.479	68.62191	82.31374
2	State 1	1766	1905.419	63.54804	68.56490
3	State 1	1651	1552.664	60.76555	57.14627
4	State 1	1589	1500.043	59.58005	56.24458
5	State 1	1551	1436.952	59.65385	55.26738

```
R> prev$status <- to_factor(prev$status)
casc_status <- c(
  "diagnostiqué, mais pas suivi",
  "suivi, mais pas sous traitement",
  "sous traitement, mais infection non contrôlée",
  "sous traitement et infection contrôlée"
)
levels(prev$status) <- casc_status
```

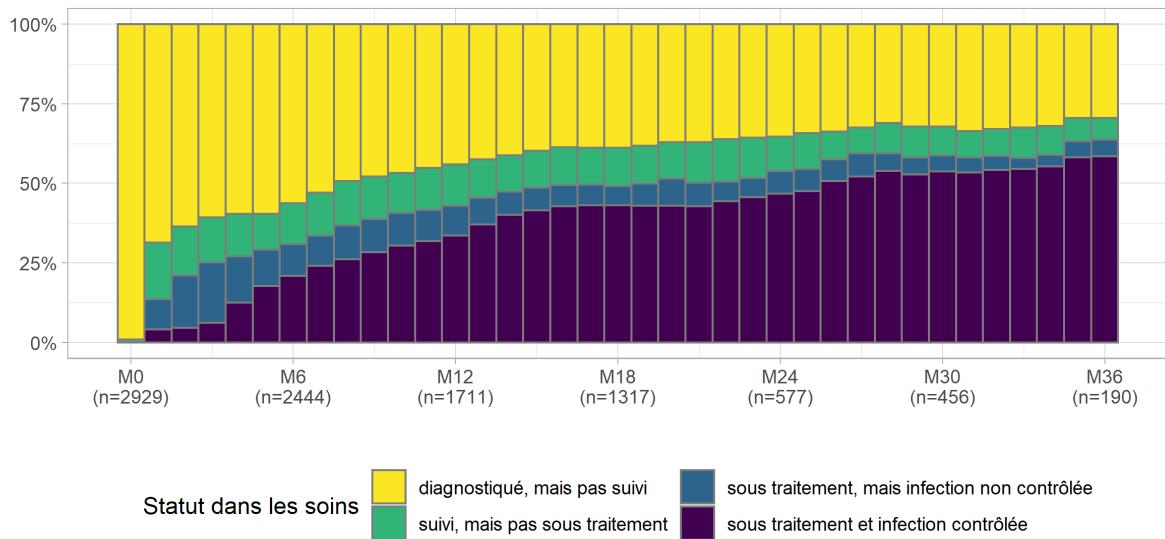
Il est alors ensuite facile de produire le graphique de la cascade de soins, estimée par le modèle, que l'on pourra mettre en comparaison de la cascade observée que nous avions calculé tout à l'heure.

---

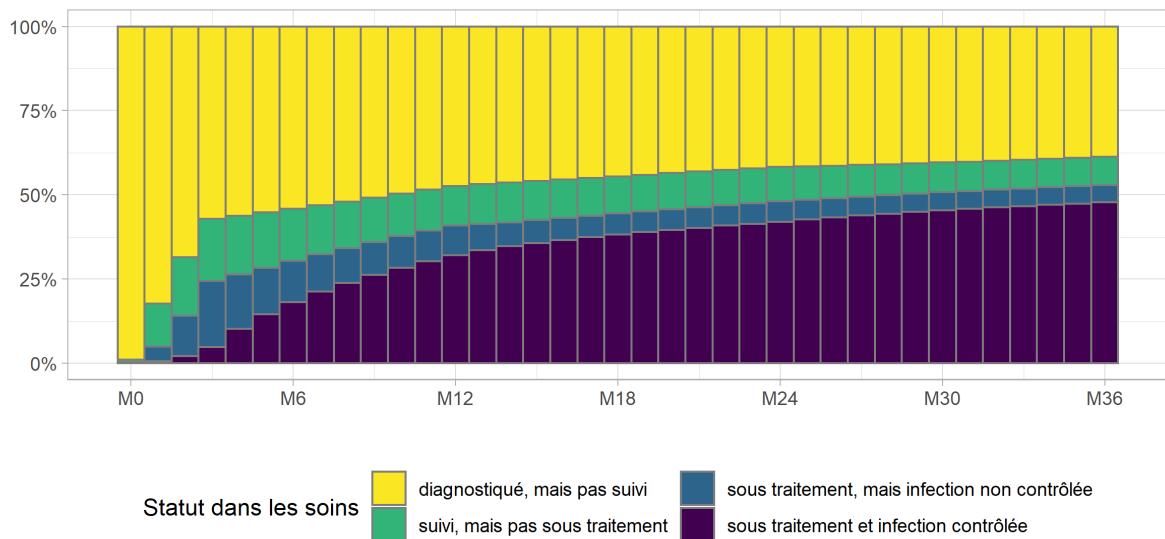
4. avant l'éventuelle intégration directement dans `msm` d'un `tidier` officiel.

```
R> casc_est <- ggplot(prev) +
  aes(x = time, fill = status, weight = expected) +
  geom_bar(color = "gray50", width = 1, position = "fill") +
  scale_x_continuous(breaks = 0:6*6, labels = paste0("M", 0:6*6)) +
  scale_y_continuous(labels = scales::percent) +
  ggtitle("Cascade des soins estimée, selon le temps depuis le diagnostic") +
  xlab("") + ylab("") +
  theme_light() +
  theme(legend.position = "bottom") +
  labs(fill = "Statut dans les soins") +
  scale_fill_viridis(discrete = TRUE, direction = -1) +
  guides(fill = guide_legend(nrow = 2))
multiplot(casc_obs, casc_est)
```

Cascade des soins observée, selon le temps depuis le diagnostic



Cascade des soins estimée, selon le temps depuis le diagnostic



Comme on peut le voir, dans le modèle, l'évolution est plus «lissée» comparativement aux données brutes observées.

Un des intérêts de `msm` est la possibilité d'y intégrer des covariables, permettant ainsi de calculer un modèle multivarié. Les variables peuvent être dépendantes du temps, puisqu'il suffit de renseigner leur valeur à chaque point d'observation.

```
R> ct$sex <- to_factor(ct$sex)
  ct$age <- to_factor(ct$age)
  ct$education <- to_factor(ct$education)
```

```
R> ms_mod_mult <- msm(
  status ~ month, subject = id, data = ct, qmatrix = tr, obstype = 1,
  pci = c(3, 12, 24),
  control = list(fnscale = 75000, trace = TRUE, maxit = 500),
  covariates = ~ sex + age + education
)
```

Les *risques relatifs*, ou *hazard ratios* en anglais, associés à chaque covariable et à chaque transition s'obtiennent avec `hazard msm`. Une fois encore, le format de sortie n'est pas le plus adapté pour un traitement graphique, mais on pourra avoir recours à `tidy.hazard msm` de **JLutils**.

```
R> hr <- tidy.hazard(msm(hazard.ms(m(m(m(ms_mod_mult))
head(hr)
```

term	transition	from	to	estimate	conf.low	conf.high
sexfemme	State 1 - State 2	State 1	State 2	1.8855232	1.7059612	2.083985
sexfemme	State 2 - State 1	State 2	State 1	0.9245660	0.7655997	1.116540
sexfemme	State 2 - State 3	State 2	State 3	0.9015913	0.8044458	1.010468
sexfemme	State 3 - State 2	State 3	State 2	1.0959810	0.8854966	1.356498
sexfemme	State 3 - State 4	State 3	State 4	1.3985443	1.2300980	1.590057
sexfemme	State 4 - State 3	State 4	State 3	0.8610144	0.6678918	1.109979

On va recoder certaines étiquettes en vue de faire un graphique des résultats.

```
R> hr$type <- "Transition ascendante"
  hr[hr$transition == c("State 2 - State 1"),]$type <- "Transition descendante"
  hr[hr$transition == c("State 3 - State 2"),]$type <- "Transition descendante"
  hr[hr$transition == c("State 4 - State 3"),]$type <- "Transition descendante"

  hr$term <- fct_recode(
    fct_inorder(hr$term),
    "femme vs. homme" = "sexfemme",
    "30-59 vs. 16-29" = "age30-59",
    "60+ vs. 16-29" = "age60+",
    "éduc. secondaire vs. primaire" = "educationsecondaire",
    "éduc. supérieure vs. primaire" = "educationsupérieur"
  )

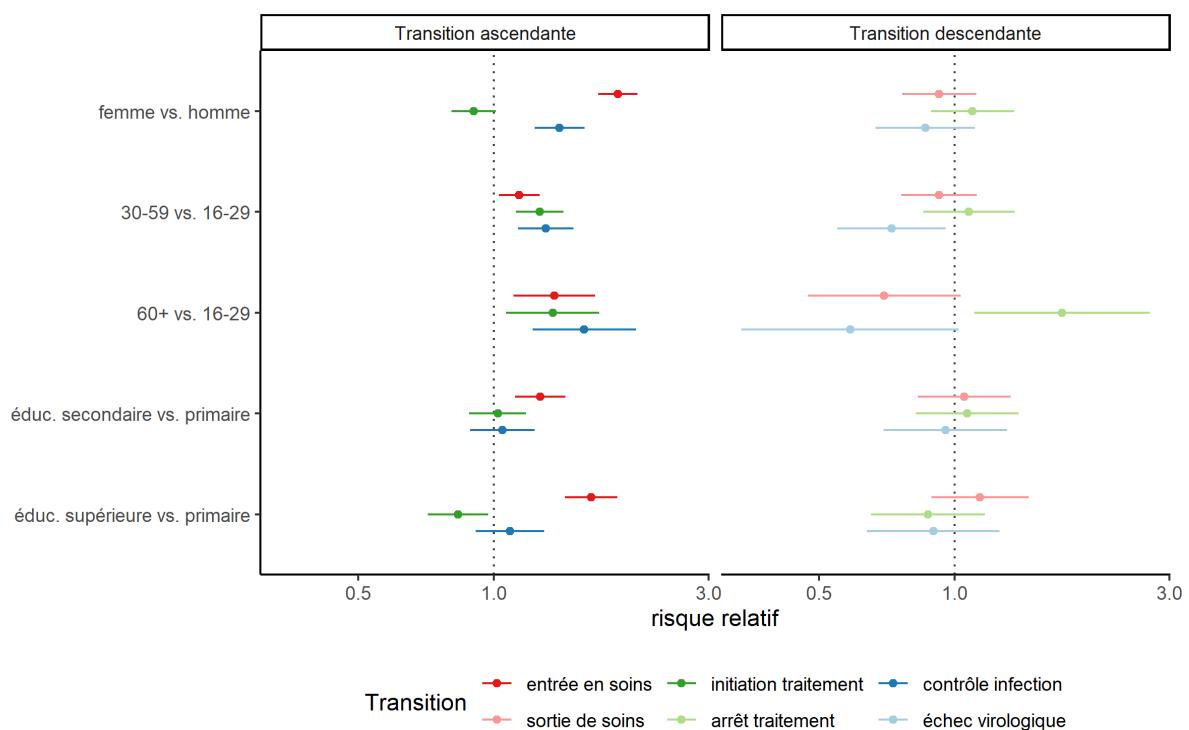
  hr$transition <- fct_recode(
    fct_inorder(hr$transition),
    "entrée en soins" = "State 1 - State 2",
    "sortie de soins" = "State 2 - State 1",
    "initiation traitement" = "State 2 - State 3",
    "arrêt traitement" = "State 3 - State 2",
    "contrôle infection" = "State 3 - State 4",
    "échec virologique" = "State 4 - State 3"
  )

head(hr)
```

term	transition	from	to	estimate	conf.low	conf.high	type
femme vs. homme	entrée en soins	State 1	State 2	1.8855232	1.7059612	2.083985	Transition ascendante
femme vs. homme	sortie de soins	State 2	State 1	0.9245660	0.7655997	1.116540	Transition descendante
femme vs. homme	initiation traitement	State 2	State 3	0.9015913	0.8044458	1.010468	Transition ascendante
femme vs. homme	arrêt traitement	State 3	State 2	1.0959810	0.8854966	1.356498	Transition descendante
femme vs. homme	contrôle infection	State 3	State 4	1.3985443	1.2300980	1.590057	Transition ascendante
femme vs. homme	échec virologique	State 4	State 3	0.8610144	0.6678918	1.109979	Transition descendante

Vu le nombre de coefficients (un risque relatif par covariable et par transition), on va organiser le graphique en distinguant les transitions ascendantes et les transitions descendantes, d'une part, et en regroupant les risques relatifs d'une même covariable, d'autre part. Pour alléger le graphique, nous allons également retirer la covariable `timeperiod` créé par l'argument `pci`, en ayant recours à `str_detect` de l'extension `stringr` pour repérer les lignes en questions (voir le chapitre sur la manipulation de texte, page 294).

```
R> ggplot(data = hr[!str_detect(hr$term, "time")], ) +
  aes(
    x = fct_rev(term), y = estimate, color = fct_rev(transition),
    ymin = conf.low, ymax = conf.high
  ) +
  geom_hline(yintercept = 1, color = "gray25", linetype = "dotted") +
  geom_errorbar(position = position_dodge(0.5), width = 0) +
  geom_point(position = position_dodge(0.5)) +
  scale_y_log10() +
  facet_grid(~ type) +
  coord_flip() +
  theme_classic() +
  theme(legend.position = "bottom") +
  ylab("risque relatif") + xlab("") +
  labs(color = "Transition") +
  scale_color_brewer(palette = "Paired") +
  guides(color = guide_legend(reverse = TRUE))
```



Ce modèle de survie multi-états permet de mettre en évidence des effets différenciés selon la transition considérée. Par exemple, les femmes sont plus rapides en matière d'entrée en soins et de contrôle de l'infection (une fois le traitement initié) mais plus lentes à démarrer le traitement (une fois entrées en soins). Par contre, le sexe ne semble pas jouer sur les transitions descendantes (échec virologique, arrête du traitement ou sortie de soins).



# Analyse de réseaux

Un bon tutoriel pour s'initier à la visualisation des réseaux avec **R**, *Network visualization with R* de Katherine Ognyanova, est disponible en ligne : <http://kateto.net/network-visualization>.

On pourra poursuivre avec différents billets de blog publiés par François Briatte sur <https://politbistro.hypotheses.org/>. Enfin, François Briatte a également recensé de nombreuses ressources en ligne sur l'analyse de réseau dans son *An awesome list of network analysis resources* (<http://f.briatte.org/r/awesome-network-analysis-list>).



# Analyse spatiale

Il est tout à fait possible de réaliser des analyses spatiales sous R. Historiquement, l'extension principale pour la gestion des objets spatiaux sous R est l'extension **sp**. Depuis quelques années, une nouvelle extension **sf** s'est développée. Alors, faut-il plutôt apprendre **sp** ou **sf**? Chris Brown tente de répondre à cette question dans son billet [Should I learn sf or sp for spatial R programming?](#). Ces deux extensions ont leurs avantages et inconvénients. Du fait que **sp** est plus ancienne, elle est compatible avec plus d'autres extensions. De l'autre côté, **sf** peut s'avérer plus simple pour le néophyte. Dans tous les cas, l'extension **raster** sera un bon complément pour gérer les données de type **raster**.

Pour une présentation détaillée (en anglais) de l'analyse spatiale sous R, on pourra se référer à l'ouvrage *Geocomputation with R* de Robin Lovelace, Jakub Nowosad et Jannes Muenchow, consultable en ligne (<https://geocompr.robinlovelace.net/>). Cette ouvrage priviliege plutôt l'extension **sf**.

On pourra également se référer aux différentes vignettes incluses par leurs extensions et consultables en ligne sur :

- <https://cran.r-project.org/package=sp> pour **sp**
- <https://cran.r-project.org/package=sf> pour **sf**
- <https://cran.r-project.org/package=raster> pour **raster**

Pour la réalisation de cartes avec R, on pourra se référer au chapitre dédié, page 895.

Enfin, le site [Awesome R](#) fournit une sélection d'extensions dédiées à l'analyse spatiale.



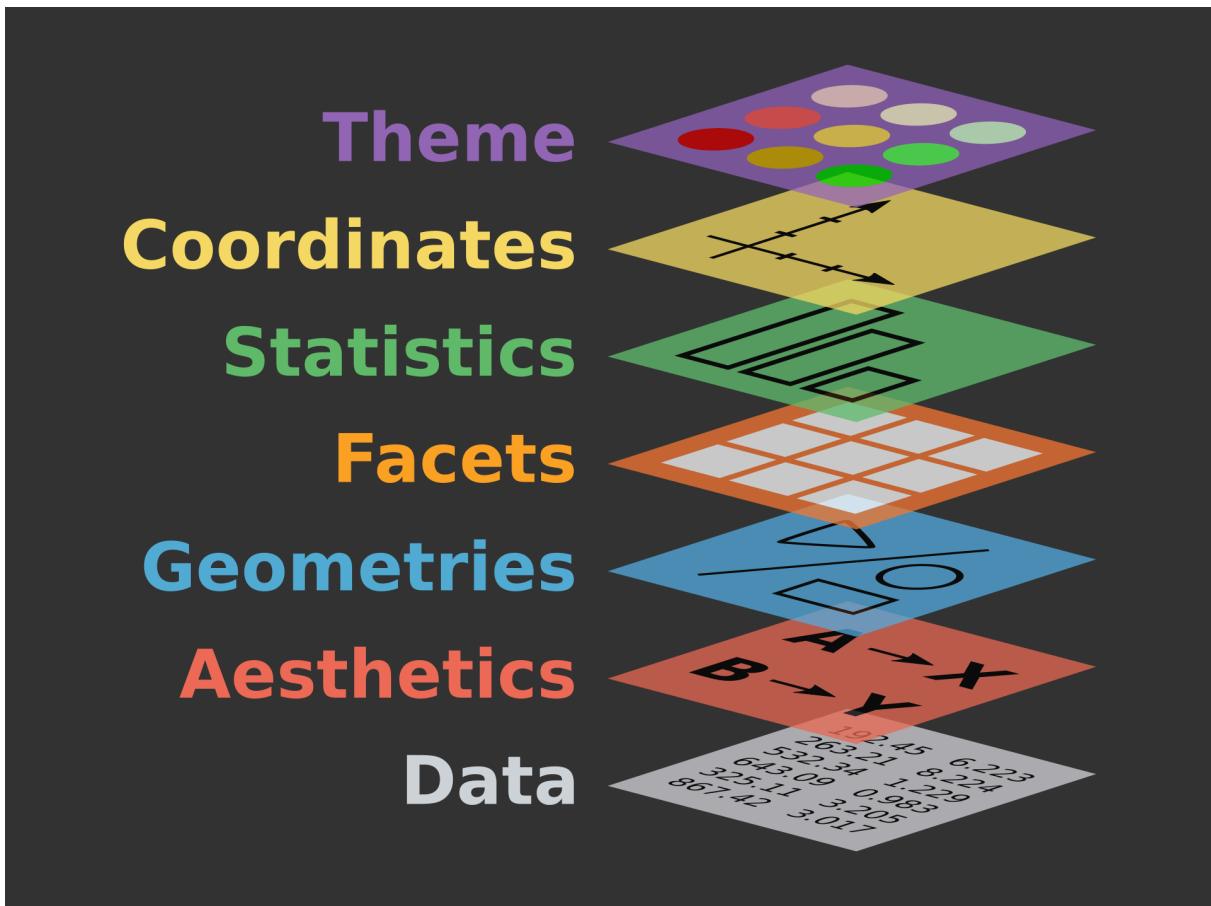
# ggplot2 et la grammaire des graphiques

Ouvrages .....	790
Assistants pour ggplot2 .....	790

## IMPORTANT

Ce chapitre est en cours d'écriture.

On pourra se référer au chapitre «5 Graphiques» du support de cours d'Ewen Galic intitulé *Logiciel R et programmation* ([http://egallic.fr/Enseignement/R/m1\\_stat\\_eco\\_logiciel\\_R.pdf](http://egallic.fr/Enseignement/R/m1_stat_eco_logiciel_R.pdf)), en complément des deux chapitres introductifs d'analyse-R : introduction à ggplot2, page 395 et graphiques bivariés avec ggplot2, page 417.



Grammaire des graphiques de ggplot2

## Ouvrages

- *ggplot2: Elegant Graphics for Data Analysis (Use R!)* d’Hadley Wickham<sup>1</sup>
- *R Graphics Cookbook* de Winston Chang

## Assistants pour ggplot2

Plusieurs extensions proposent une assistance visuelle pour l’utilisation de **ggplot2** via des *add-ins* dans RStudio :

- **esquisse** pour explorer visuellement des données et produire des graphiques de base avec **ggplot2** (plus d’informations sur <https://github.com/dreamRs/esquisse>) ;
- **ggplotAssist** pour générer des graphiques **ggplot2** étape par étape (plus d’informations sur

1. Une version PDF est disponible sur <http://moderngraphics11.pbworks.com/f/ggplot2-Book09hWickham.pdf>.

- <https://github.com/cardiomoon/ggplotAssist/> ;  
• **ggThemeAssist** pour éditer le thème d'un graphique déjà réalisé.



# Étendre ggplot2

Nouvelles géométries . . . . .	794
Étiquettes non superposées . . . . .	794
Graphiques en sucettes ( <i>lollipop</i> ) . . . . .	796
Graphique d'haltères ( <i>dumbbell</i> ) . . . . .	798
Accolades de comparaison ( <i>bracket</i> ) . . . . .	800
Diagramme en crêtes ( <i>ridges</i> ) . . . . .	801
Graphique en gaufres ( <i>waffle</i> ) . . . . .	802
Graphique en mosaïque ( <i>mosaic plot</i> ) . . . . .	804
Graphique de pirates : alternative aux boîtes à moustache ( <i>pirat plot</i> ) . . . . .	805
Axes, légende et facettes . . . . .	806
Axes «limités» . . . . .	806
Répéter les étiquettes des axes sur des facettes . . . . .	807
Cartes . . . . .	809
Graphiques complexes . . . . .	809
Graphiques divergents . . . . .	809
Graphiques interactifs . . . . .	812
Graphiques animés . . . . .	812
Surligner certaines données . . . . .	817
Thèmes et couleurs . . . . .	820
Palettes de couleurs . . . . .	820
hrbrthemes . . . . .	820
ggthemes . . . . .	821
Combiner plusieurs graphiques . . . . .	823

De nombreuses extensions permettent d'étendre les possibilités graphiques de **ggplot2**. Certaines ont déjà été abordées dans les différents chapitres d'**analyse-R**. Le présent chapitre ne se veut pas exhaustif et ne présente qu'une sélection choisie d'extensions.

Le site **ggplot2 extensions** (<https://exts.ggplot2.tidyverse.org/gallery/>) recense diverses extensions pour **ggplot2**.

Pour une présentation des fonctions de base et des concepts de **ggplot2**, on pourra se référer au chapitre dédié, page 789 ainsi qu'au deux chapitres introductifs : introduction à **ggplot2**, page 395 et graphiques bivariés avec **ggplot2**, page 417.

Pour trouver l’inspiration et des exemples de code, rien ne vaut l’excellent site <https://www.r-graph-gallery.com/>.

## Nouvelles géométries

### Étiquettes non superposées

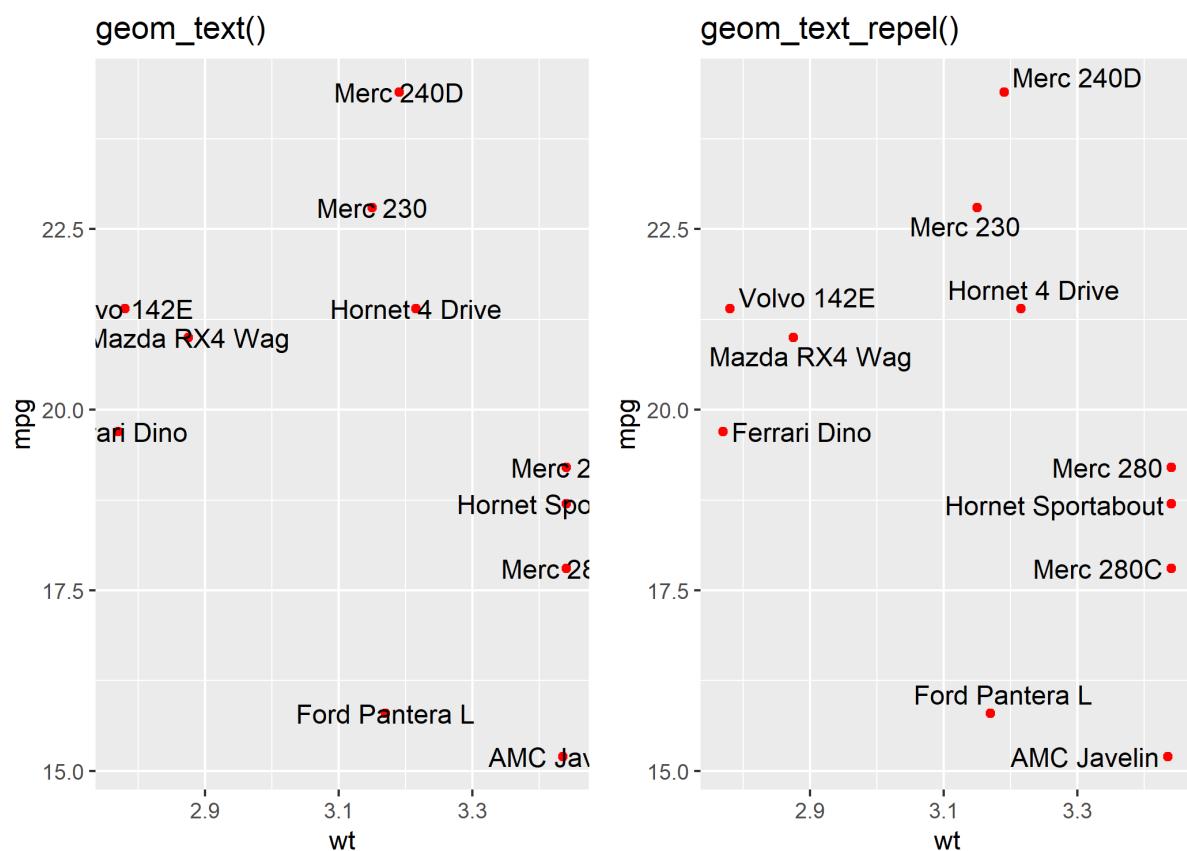
Lorsque l’on affiche des étiquettes de texte, ces dernières peuvent se superposer lorsqu’elles sont proches. Les géométries `geom_text_repel` et `geom_label_repel` de l’extension `ggrepel` prennent en compte la position des différentes étiquettes pour éviter qu’elles ne se chevauchent.

```
R> library(ggplot2)
library(ggrepel)
library(ggrepel)

dat <- subset(mtcars, wt > 2.75 & wt < 3.45)
dat$car <- rownames(dat)
p <- ggplot(dat) +
  aes(wt, mpg, label = car) +
  geom_point(color = "red")

p1 <- p + geom_text() +
  labs(title = "geom_text()")
p2 <- p + geom_text_repel() +
  labs(title = "geom_text_repel()")

cowplot:::plot_grid(p1, p2, nrow = 1)
```



Pour plus d'informations : <https://ggrepel.slowkow.com/>

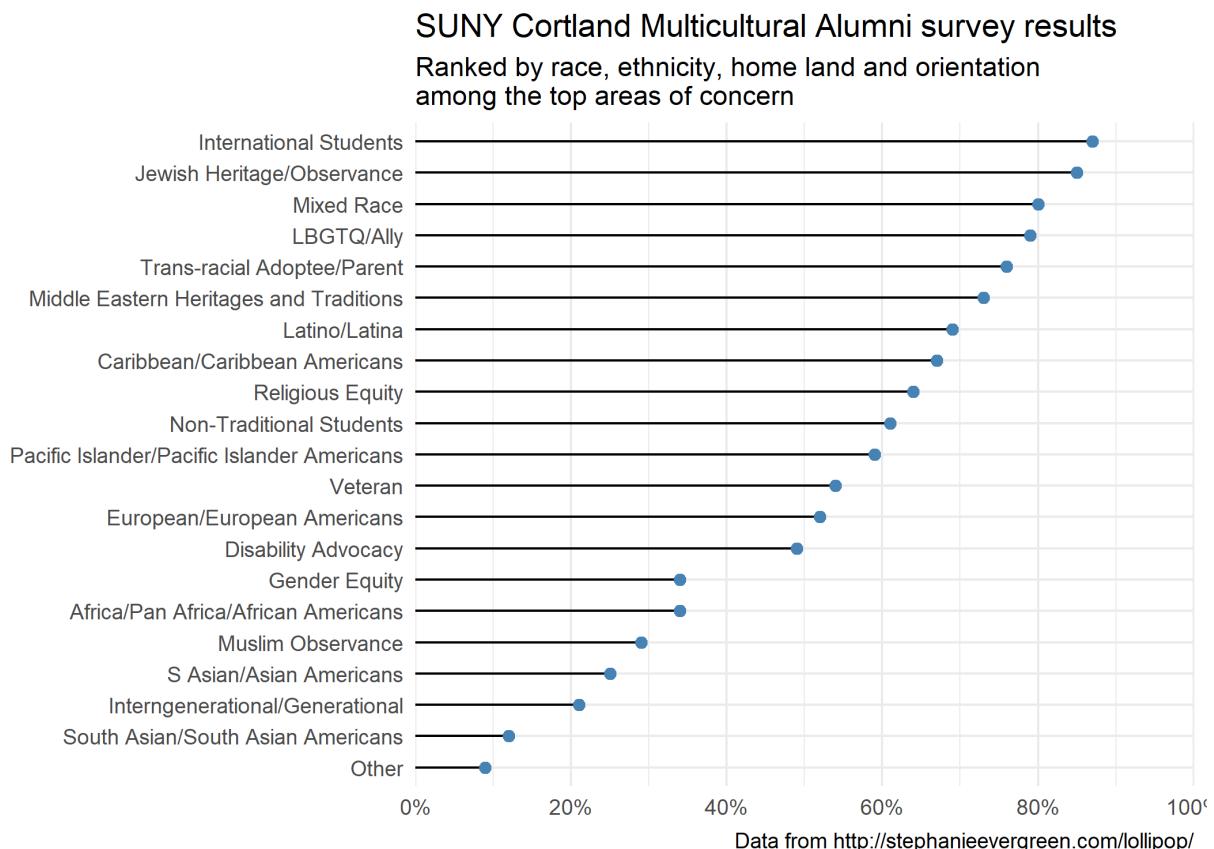
## Graphiques en sucettes (*lollipop*)

L’extension `ggalt` propose une géométrie `geom_lollipop` permettant de réaliser des graphiques dit en «sucettes».

```
R> df <- read.csv(text = "category,pct
  Other,0.09
  South Asian/South Asian Americans,0.12
  Intergenerational/Generational,0.21
  S Asian/Asian Americans,0.25
  Muslim Observance,0.29
  Africa/Pan Africa/African Americans,0.34
  Gender Equity,0.34
  Disability Advocacy,0.49
  European/European Americans,0.52
  Veteran,0.54
  Pacific Islander/Pacific Islander Americans,0.59
  Non-Traditional Students,0.61
  Religious Equity,0.64
  Caribbean/Caribbean Americans,0.67
  Latino/Latina,0.69
  Middle Eastern Heritages and Traditions,0.73
  Trans-racial Adoptee/Parent,0.76
  LBGTQ/Ally,0.79
  Mixed Race,0.80
  Jewish Heritage/Observance,0.85
  International Students,0.87", stringsAsFactors = FALSE, sep = ",",
  header = TRUE)

library(ggplot2)
library(ggalt)
library(scales)

ggplot(df) +
  aes(y = reorder(category, pct), x = pct) +
  geom_lollipop(point.colour = "steelblue", point.size = 2, horizontal = TRUE) +
  scale_x_continuous(expand = c(0, 0), labels = percent, breaks = seq(0, 1, by = 0.2),
                     limits = c(0, 1)) +
  labs(
    x = NULL, y = NULL,
    title = "SUNY Cortland Multicultural Alumni survey results",
    subtitle = "Ranked by race, ethnicity, home land and orientation\namong the top areas of concern",
    caption = "Data from http://stephanieevergreen.com/lollipop/")
  ) +
  theme_minimal()
```

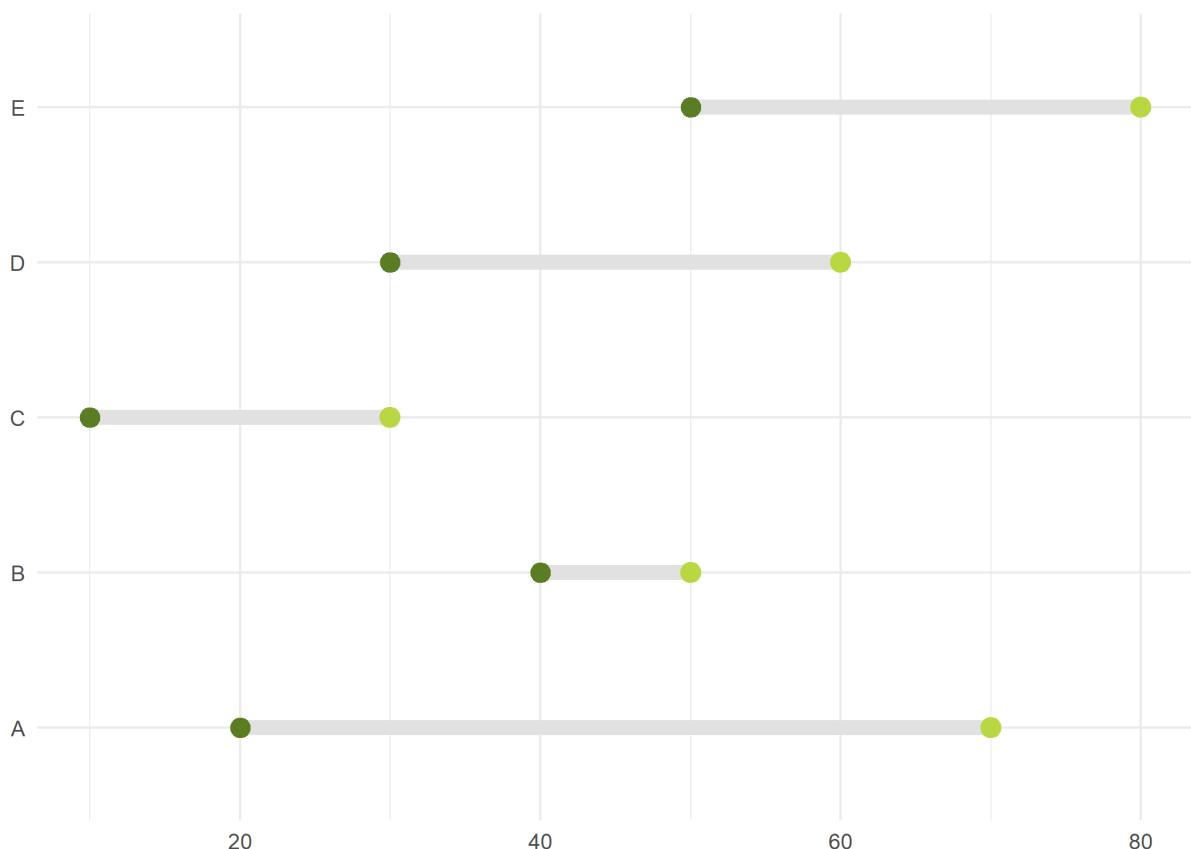


## Graphique d'haltères (dumbbell)

L'extension `ggalt` propose une géométrie `geom_dumbbell` permettant de réaliser des graphiques dit en «haltères».

```
R> library(ggalt)
df <- data.frame(
  trt = LETTERS[1:5],
  l = c(20, 40, 10, 30, 50),
  r = c(70, 50, 30, 60, 80)
)

ggplot(df) +
  aes(y = trt, x = l, xend = r) +
  geom_dumbbell(
    size = 3,
    color = "#e3e2e1",
    colour_x = "#5b8124",
    colour_xend = "#bad744"
  ) +
  labs(x = NULL, y = NULL) +
  theme_minimal()
```

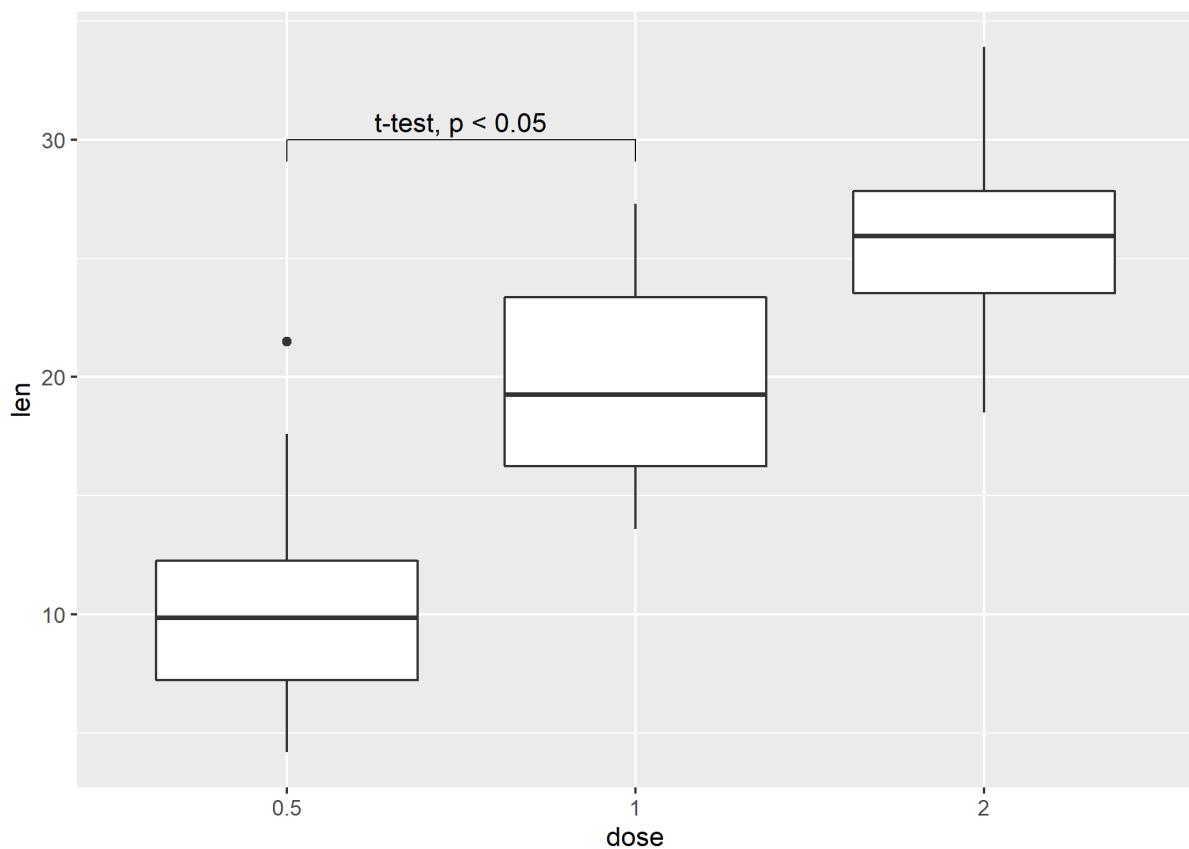


## Accolades de comparaison (bracket)

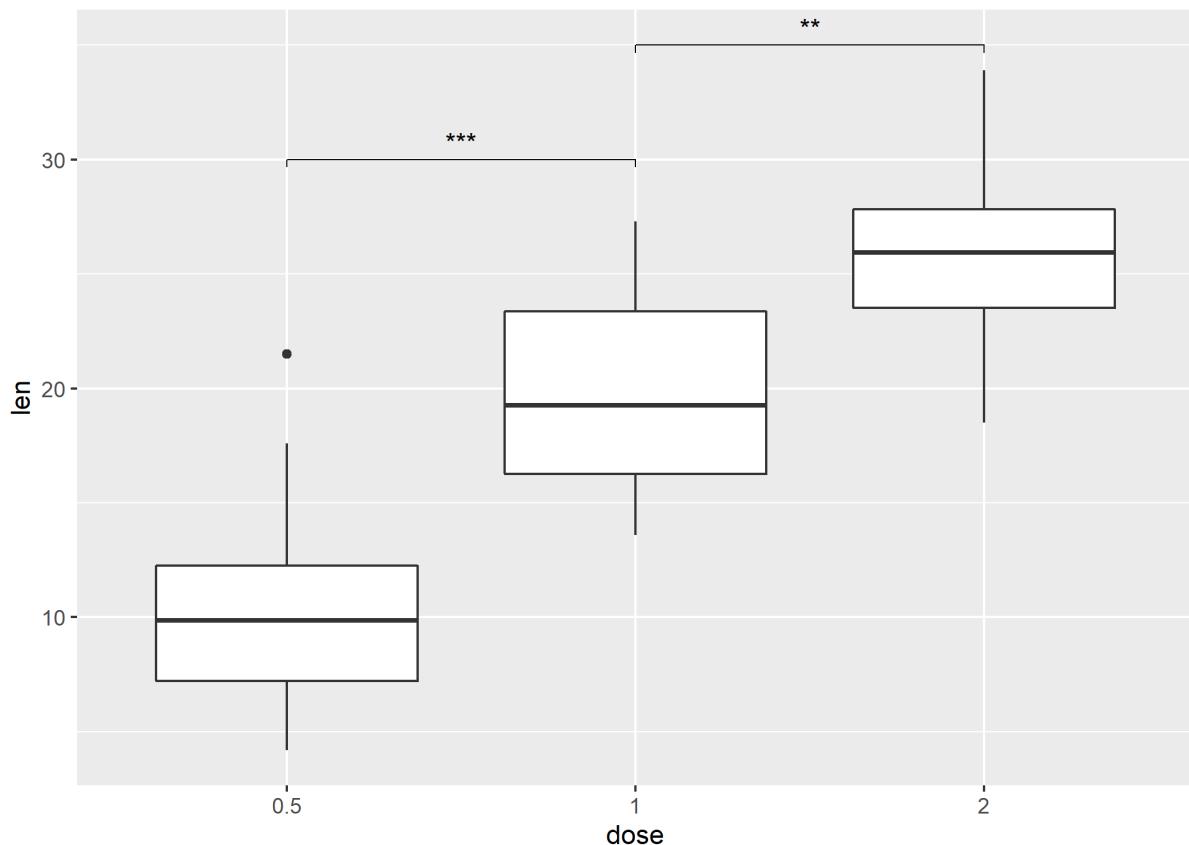
La géométrie `geom_braket` de l'extension `ggpubr` permet d'ajouter sur un graphique des accolades de comparaison entre groupes.

```
R> library(ggpubr)
df <- ToothGrowth
df$dose <- factor(df$dose)

ggplot(df) +
  aes(x = dose, y = len) +
  geom_boxplot() +
  geom_braket(
    xmin = "0.5", xmax = "1", y.position = 30,
    label = "t-test, p < 0.05"
  )
```



```
R> ggplot(df) +
  aes(x = dose, y = len) +
  geom_boxplot() +
  geom_pvalue_label(
    xmin = c("0.5", "1"),
    xmax = c("1", "2"),
    y.position = c(30, 35),
    label = c("***", "**"),
    tip.length = 0.01
  )
```



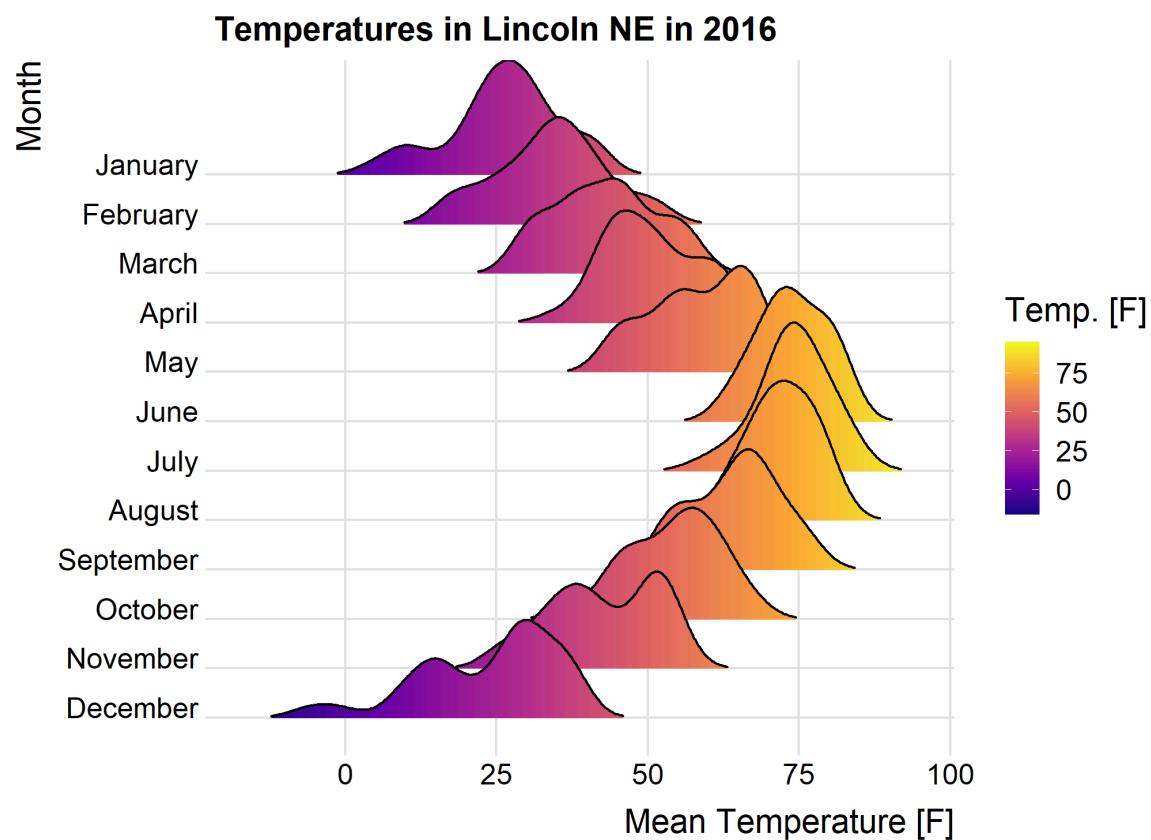
Plus d'informations : <https://rpkgs.datanovia.com/ggpubr/>

## Diagramme en crêtes (ridges)

L'extension **ggridges** fournit une géométrie `geom_density_ridges_gradient` pour la création de diagramme en crêtes.

```
R> library(ggrridges)
ggplot(lincoln_weather, aes(x = `Mean Temperature [F]`, y = Month, fill = stat(x))) +
  geom_density_ridges_gradient(scale = 3, rel_min_height = 0.01) +
  scale_fill_viridis_c(name = "Temp. [F]", option = "C") +
  labs(title = "Temperatures in Lincoln NE in 2016") +
  theme_ridges()
```

Picking joint bandwidth of 3.37



Plus d'informations : <https://wilkelab.org/ggrridges/>

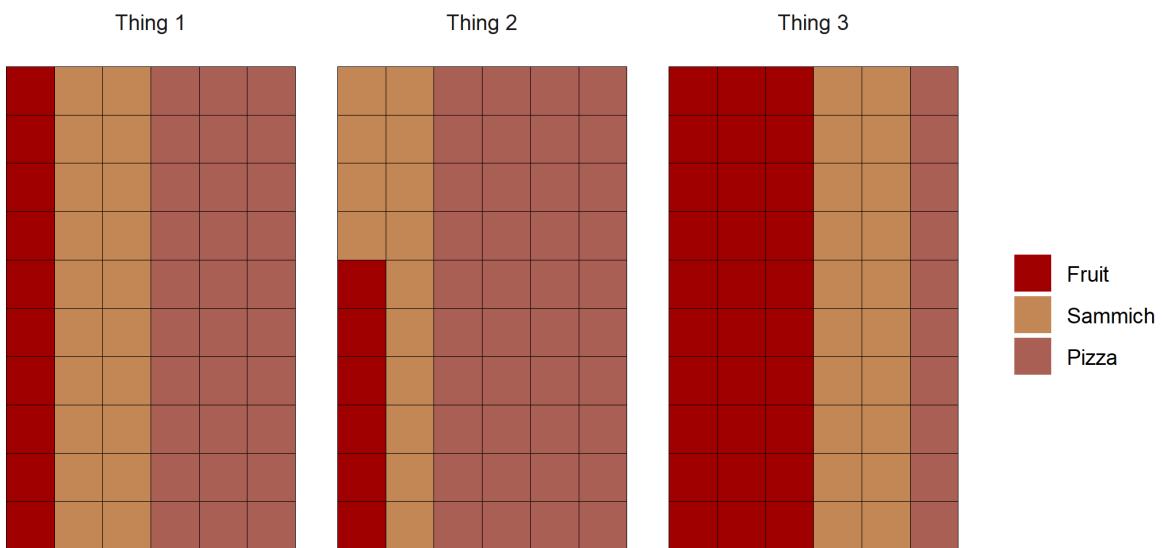
## Graphique en gaufres (waffle)

L'extension **waffle** propose `geom_waffle` pour des graphiques dits en «gaufres».

ATTENTION : elle s'installe avec la commande  
`install.packages("waffle", repos = "https://cinc.rud.is")`.

```
R> library(waffle)
xdf <- data.frame(
  parts = factor(rep(month.abb[1:3], 3), levels = month.abb[1:3]),
  vals = c(10, 20, 30, 6, 14, 40, 30, 20, 10),
  fct = c(rep("Thing 1", 3), rep("Thing 2", 3), rep("Thing 3", 3))
)

ggplot(xdf) +
  aes(fill = parts, values = vals) +
  geom_waffle() +
  facet_wrap(~fct) +
  scale_fill_manual(
    name = NULL,
    values = c("#a40000", "#c68958", "#ae6056"),
    labels = c("Fruit", "Sammich", "Pizza")
  ) +
  coord_equal() +
  theme_minimal() +
  theme_enhance_waffle()
```

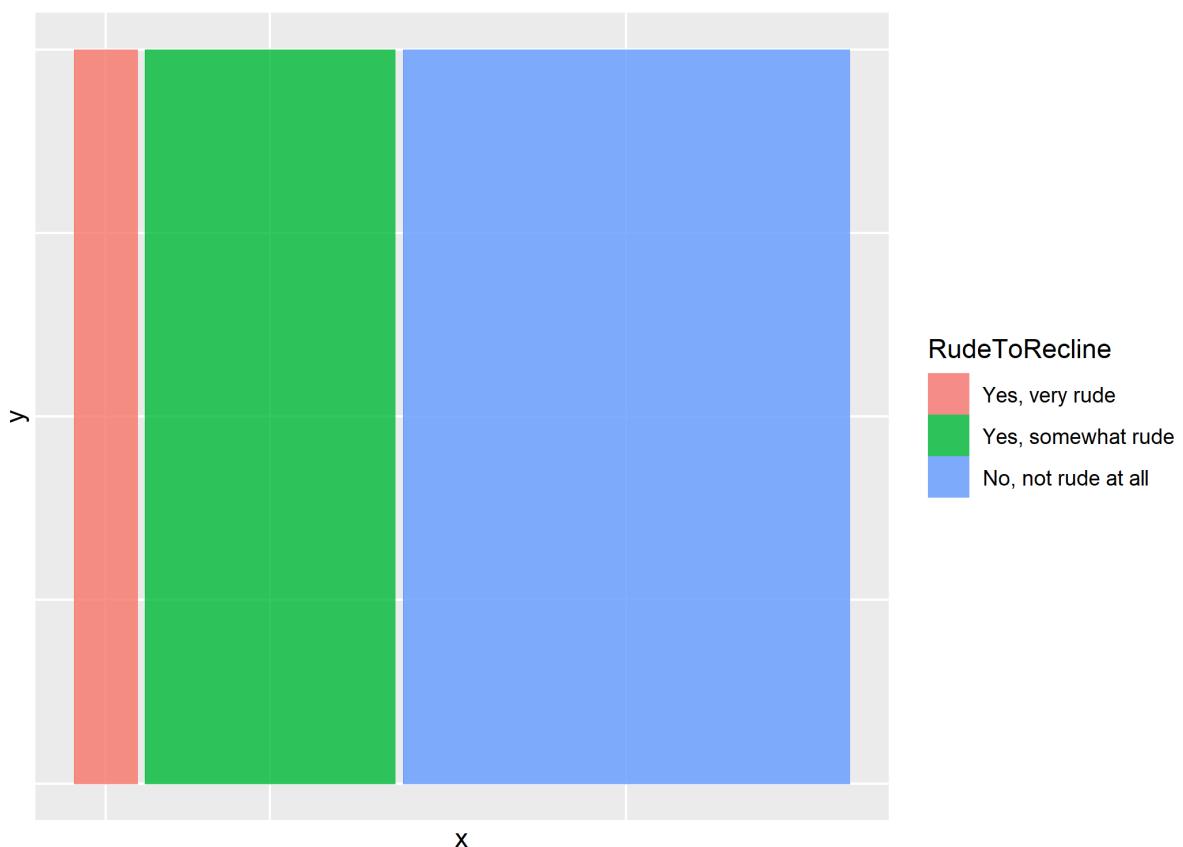


Plus d'informations : <https://github.com/hrbrmstr/waffle>

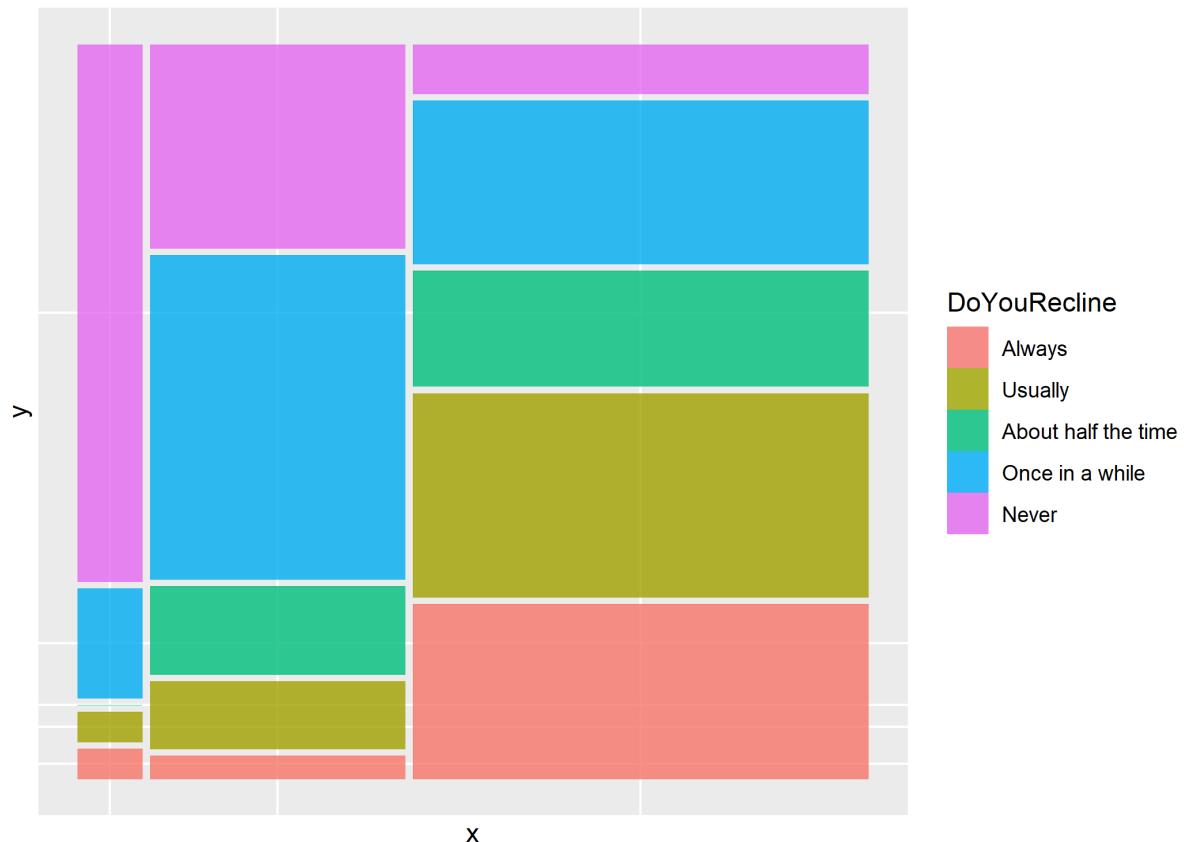
## Graphique en mosaïque (*mosaic plot*)

L'extension `ggmosaic` permet de réaliser des graphiques en mosaïque avec `geom_mosaic`.

```
R> library(ggmosaic)
  ggplot(data = fly) +
    geom_mosaic(
      aes(x = product(RudeToRecline), fill = RudeToRecline),
      na.rm = TRUE
    )
```



```
R> ggplot(data = fly) +
  geom_mosaic(
    aes(x = product(DoYouRecline, RudeToRecline), fill = DoYouRecline),
    na.rm = TRUE
  )
```



Plus d'informations : <https://cran.r-project.org/web/packages/ggmosaic/vignettes/ggmosaic.html>

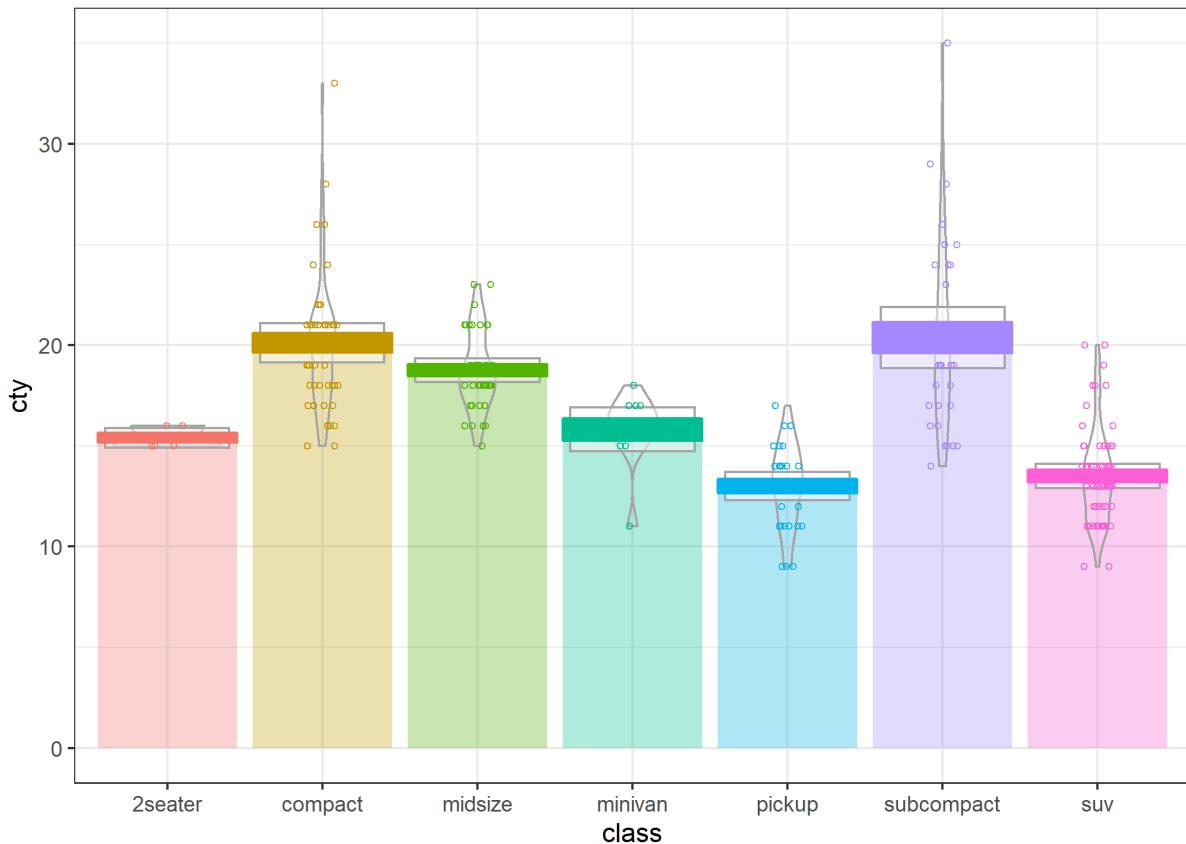
## Graphique de pirates : alternative aux boîtes à moustache (*pirate plot*)

Cette représentation alternative aux boîtes à moustache s'obtient avec la géométrie `geom_pirate` de l'extension `ggpirate`<sup>1</sup>.

---

1. Cette extension n'étant pas sur CRAN, on l'installera avec la commande `devtools::install_github("mikabr/ggpirate")`.

```
R> library(ggplot2)
library(ggpirate)
ggplot(mpg, aes(x = class, y = cty)) +
  geom_pirate(aes(colour = class, fill = class)) +
  theme_bw()
```



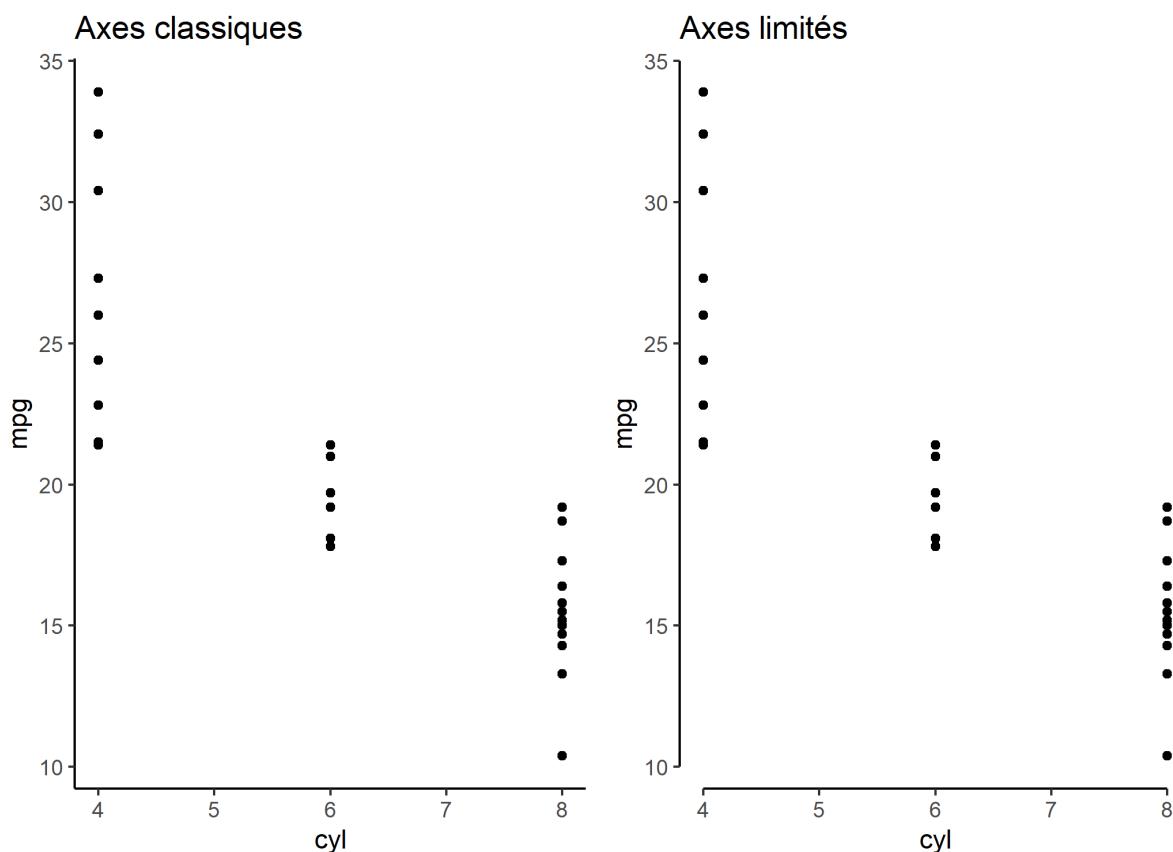
Pour plus d'informations : <https://github.com/mikab/r/ggpirate>

## Axes, légende et facettes

### Axes «limités»

`coord_capped_cart` et `coord_capped_flip` de l'extension `lemon` permet de limiter le dessin des axes au minimum et au maximum. Voir l'exemple ci-dessous.

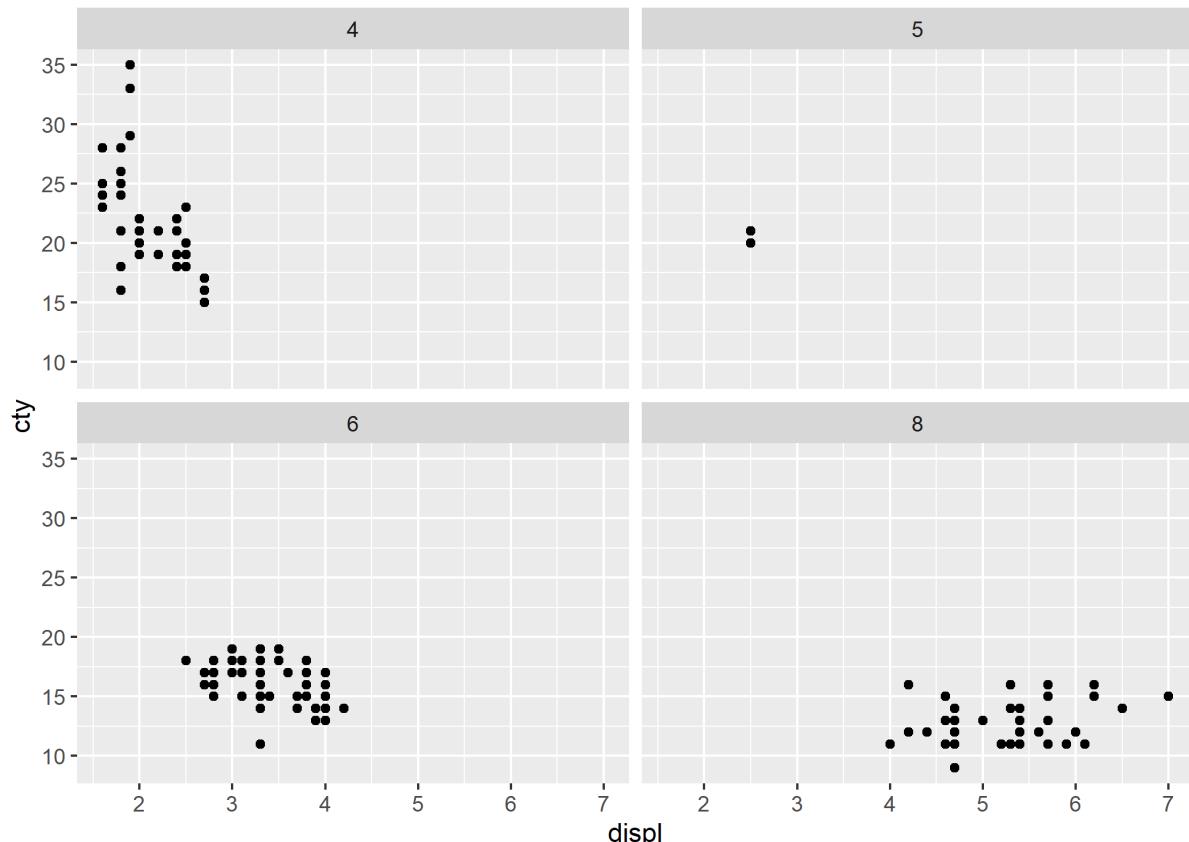
```
R> library(ggplot2)
library(lemon)
p <- ggplot(mtcars) +
  aes(x = cyl, y = mpg) +
  geom_point() +
  theme_classic() +
  ggtitle("Axes classiques")
pcapped <- p +
  coord_capped_cart(bottom = "both", left = "both") +
  ggtitle("Axes limités")
cowplot::plot_grid(p, pcapped, nrow = 1)
```



## Répéter les étiquettes des axes sur des facettes

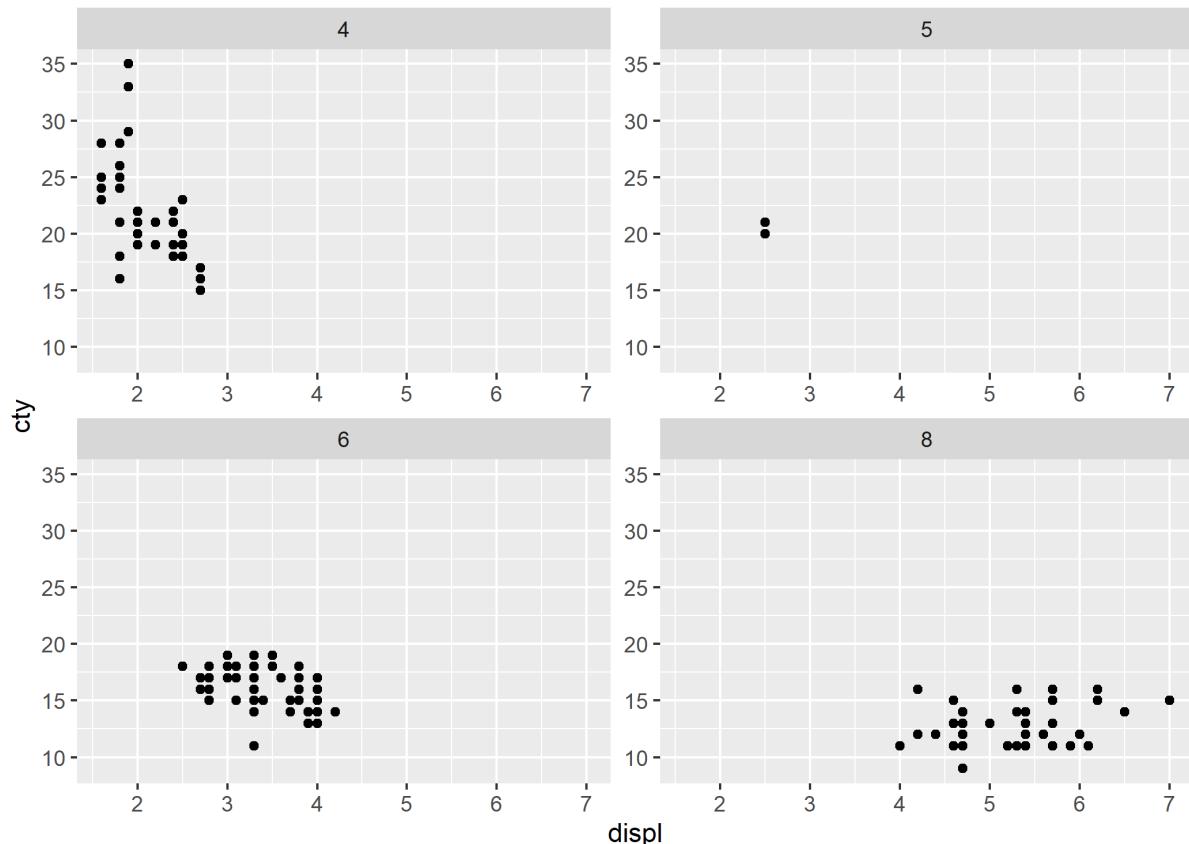
Lorsque l'on réalise des facettes, les étiquettes des axes ne sont pas répétées.

```
R> library(ggplot2)
R> ggplot(mpg) +
  aes(displ, cty) +
  geom_point() +
  facet_wrap(~cyl)
```



L’extension `lemon` propose `facet_rep_grid` et `facet_rep_wrap` qui répètent les axes sur chaque facette.

```
R> library(lemon)
ggplot(mpg) +
  aes(displ, cty) +
  geom_point() +
  facet_wrap(~cyl, repeat.tick.labels = TRUE)
```



## Cartes

Voir le chapitre dédié, page 895.

## Graphiques complexes

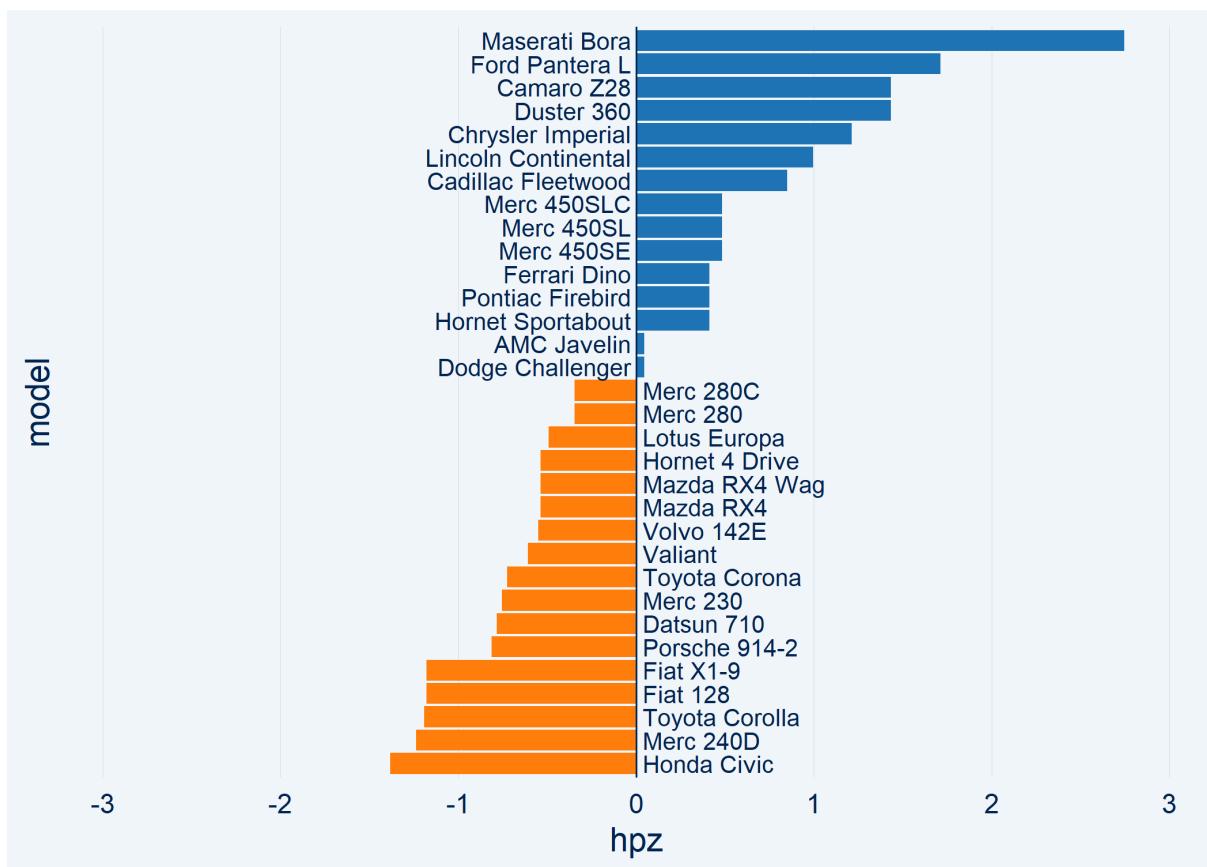
### Graphiques divergents

L'extension `ggcharts` fournit plusieurs fonctions de haut niveau pour faciliter la réalisation de

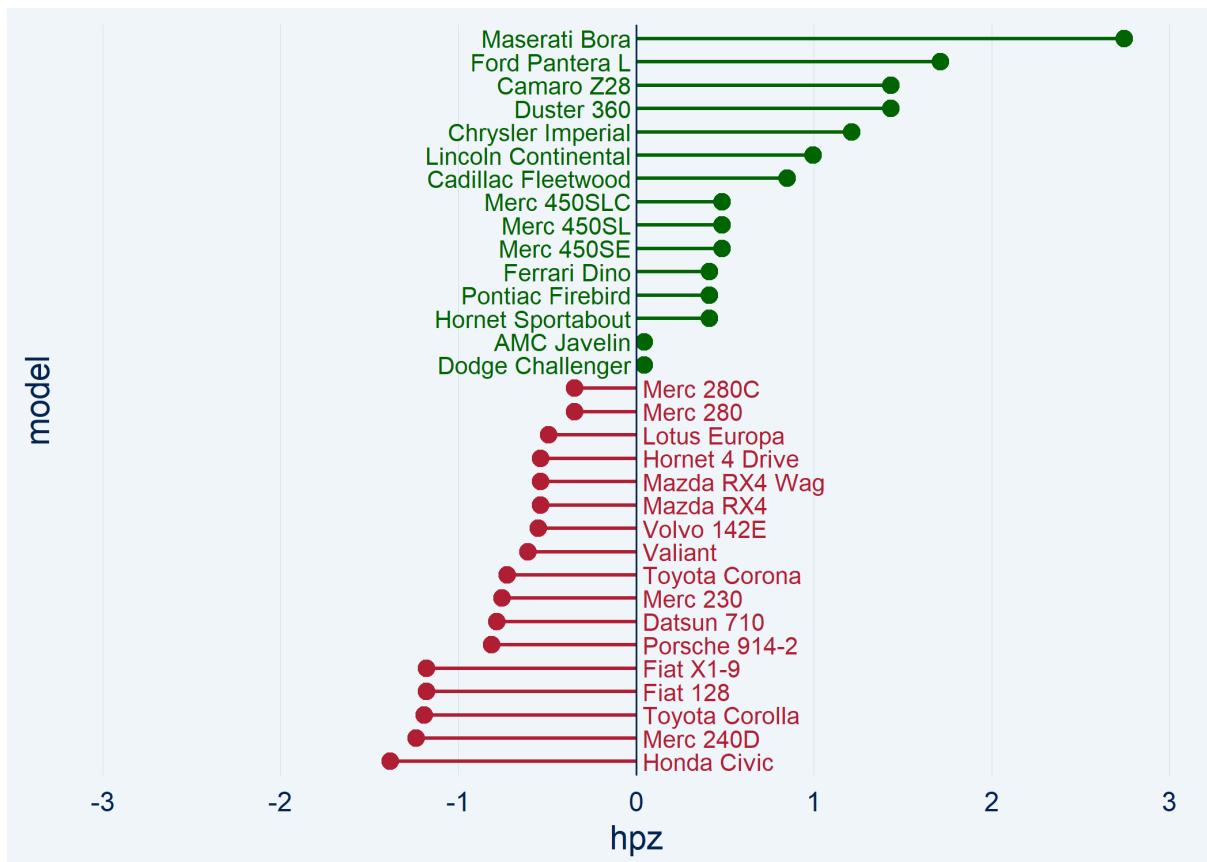
graphiques divergents en barres ([diverging\\_bar\\_chart](#)), en sucettes ([diverging\\_lollipop\\_chart](#)) voire même une pyramide des âges ([pyramid\\_chart](#)).

```
R> library(ggcharts)
data(mtcars)
mtcars_z <- dplyr::transmute(
  .data = mtcars,
  model = row.names(mtcars),
  hpz = scale(hp)
)

diverging_bar_chart(data = mtcars_z, x = model, y = hpz)
```



```
R> diverging_lollipop_chart(
  data = mtcars_z,
  x = model,
  y = hpz,
  lollipop_colors = c("#006400", "#b32134"),
  text_color = c("#006400", "#b32134")
)
```



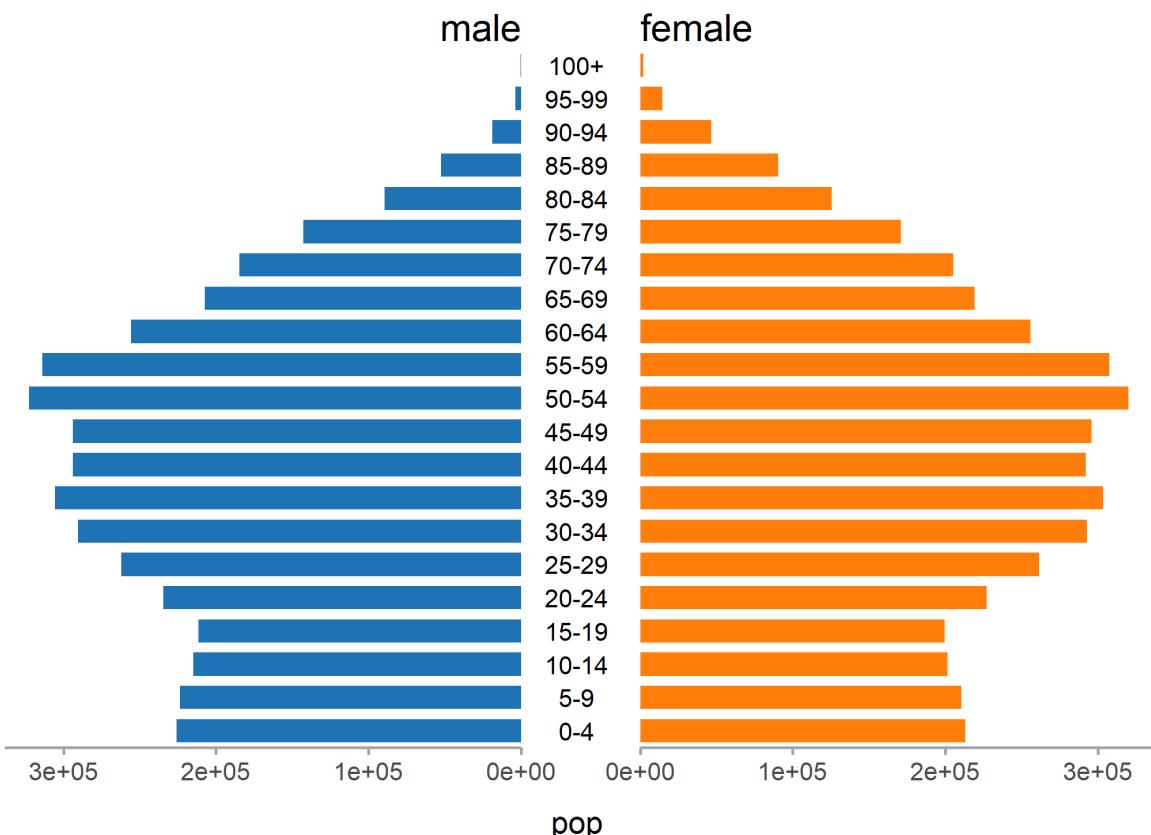
```
R> data("popch")
pyramid_chart(data = popch, x = age, y = pop, group = sex)
```

Warning: `expand\_scale()` is deprecated; use `expansion()` instead.

Warning: `expand\_scale()` is deprecated; use `expansion()` instead.

Warning: `expand\_scale()` is deprecated; use `expansion()` instead.

```
Warning: `expand_scale()` is deprecated; use `expansion()` instead.
```



## Graphiques interactifs

Voir le chapitre dédié, page 875.

## Graphiques animés

L’extension **ganimate** permet de réaliser des graphiques animés.

Voici un exemple :

```
R> library(ggplot2)
library(gganimate)
library(gapminder)

ggplot(gapminder, aes(gdpPercap, lifeExp, size = pop, colour = country)) +
  geom_point(alpha = 0.7, show.legend = FALSE) +
  scale_colour_manual(values = country_colors) +
  scale_size(range = c(2, 12)) +
  scale_x_log10() +
  facet_wrap(~continent) +
  # Here comes the gganimate specific bits
  labs(title = "Year: {frame_time}", x = "GDP per capita", y = "life expectancy") +
  transition_time(year) +
  ease_aes("linear")
```

```
Rendering [-----] at 2.1 fps ~ eta: 47s
Rendering [-----] at 1.8 fps ~ eta: 1m
Rendering [>-----] at 1.7 fps ~ eta: 1m
Rendering [>-----] at 1.8 fps ~ eta: 1m
Rendering [>-----] at 1.7 fps ~ eta: 1m
Rendering [=>-----] at 1.7 fps ~ eta: 1m
Rendering [==>-----] at 1.6 fps ~ eta: 1m
Rendering [====>-----] at 1.6 fps ~ eta: 1m
Rendering [====>-----] at 1.6 fps ~ eta: 1m
Rendering [====>-----] at 1.5 fps ~ eta: 1m
Rendering [=====>-----] at 1.5 fps ~ eta: 1m
Rendering [=====>-----] at 1.5 fps ~ eta: 49s
Rendering [=====>-----] at 1.5 fps ~ eta: 48s
Rendering [=====>-----] at 1.6 fps ~ eta: 47s
Rendering [=====>-----] at 1.6 fps ~ eta: 46s
Rendering [=====>-----] at 1.6 fps ~ eta: 45s
Rendering [=====>-----] at 1.6 fps ~ eta: 44s
Rendering [=====>-----] at 1.6 fps ~ eta: 43s
Rendering [=====>-----] at 1.6 fps ~ eta: 42s
Rendering [=====>-----] at 1.6 fps ~ eta: 41s
Rendering [=====>-----] at 1.6 fps ~ eta: 40s
Rendering [=====>-----] at 1.6 fps ~ eta: 39s
Rendering [=====>-----] at 1.7 fps ~ eta: 39s
Rendering [=====>-----] at 1.7 fps ~ eta: 38s
Rendering [=====>-----] at 1.6 fps ~ eta: 37s
Rendering [=====>-----] at 1.6 fps ~ eta: 37s
Rendering [=====>-----] at 1.6 fps ~ eta: 36s
Rendering [=====>-----] at 1.6 fps ~ eta: 36s
Rendering [=====>-----] at 1.6 fps ~ eta: 35s
Rendering [=====>-----] at 1.6 fps ~ eta: 34s
Rendering [=====>-----] at 1.5 fps ~ eta: 34s
```

```
Rendering [=====>-----] at 1.5 fps ~ eta: 33s
Rendering [=====>-----] at 1.5 fps ~ eta: 32s
Rendering [=====>-----] at 1.5 fps ~ eta: 31s
Rendering [=====>-----] at 1.5 fps ~ eta: 30s
Rendering [=====>-----] at 1.6 fps ~ eta: 30s
Rendering [=====>-----] at 1.6 fps ~ eta: 29s
Rendering [=====>-----] at 1.6 fps ~ eta: 28s
Rendering [=====>-----] at 1.6 fps ~ eta: 28s
Rendering [=====>-----] at 1.6 fps ~ eta: 27s
Rendering [=====>-----] at 1.6 fps ~ eta: 26s
Rendering [=====>-----] at 1.6 fps ~ eta: 25s
Rendering [=====>-----] at 1.6 fps ~ eta: 24s
Rendering [=====>-----] at 1.6 fps ~ eta: 23s
Rendering [=====>-----] at 1.6 fps ~ eta: 22s
Rendering [=====>-----] at 1.6 fps ~ eta: 21s
Rendering [=====>-----] at 1.6 fps ~ eta: 20s
Rendering [=====>-----] at 1.6 fps ~ eta: 19s
Rendering [=====>-----] at 1.6 fps ~ eta: 18s
Rendering [=====>-----] at 1.6 fps ~ eta: 17s
Rendering [=====>-----] at 1.6 fps ~ eta: 16s
Rendering [=====>-----] at 1.7 fps ~ eta: 16s
Rendering [=====>-----] at 1.7 fps ~ eta: 15s
Rendering [=====>-----] at 1.7 fps ~ eta: 14s
Rendering [=====>-----] at 1.7 fps ~ eta: 13s
Rendering [=====>-----] at 1.7 fps ~ eta: 12s
Rendering [=====>-----] at 1.7 fps ~ eta: 11s
Rendering [=====>-----] at 1.7 fps ~ eta: 11s
Rendering [=====>-----] at 1.7 fps ~ eta: 10s
Rendering [=====>-----] at 1.7 fps ~ eta: 9s
Rendering [=====>-----] at 1.7 fps ~ eta: 8s
Rendering [=====>-----] at 1.7 fps ~ eta: 7s
Rendering [=====>-----] at 1.7 fps ~ eta: 6s
Rendering [=====>-----] at 1.7 fps ~ eta: 6s
Rendering [=====>-----] at 1.7 fps ~ eta: 5s
Rendering [=====>-----] at 1.7 fps ~ eta: 4s
Rendering [=====>-----] at 1.7 fps ~ eta: 3s
Rendering [=====>-----] at 1.7 fps ~ eta: 2s
Rendering [=====>-----] at 1.8 fps ~ eta: 1s
Rendering [=====>-----] at 1.8 fps ~ eta: 0s
```

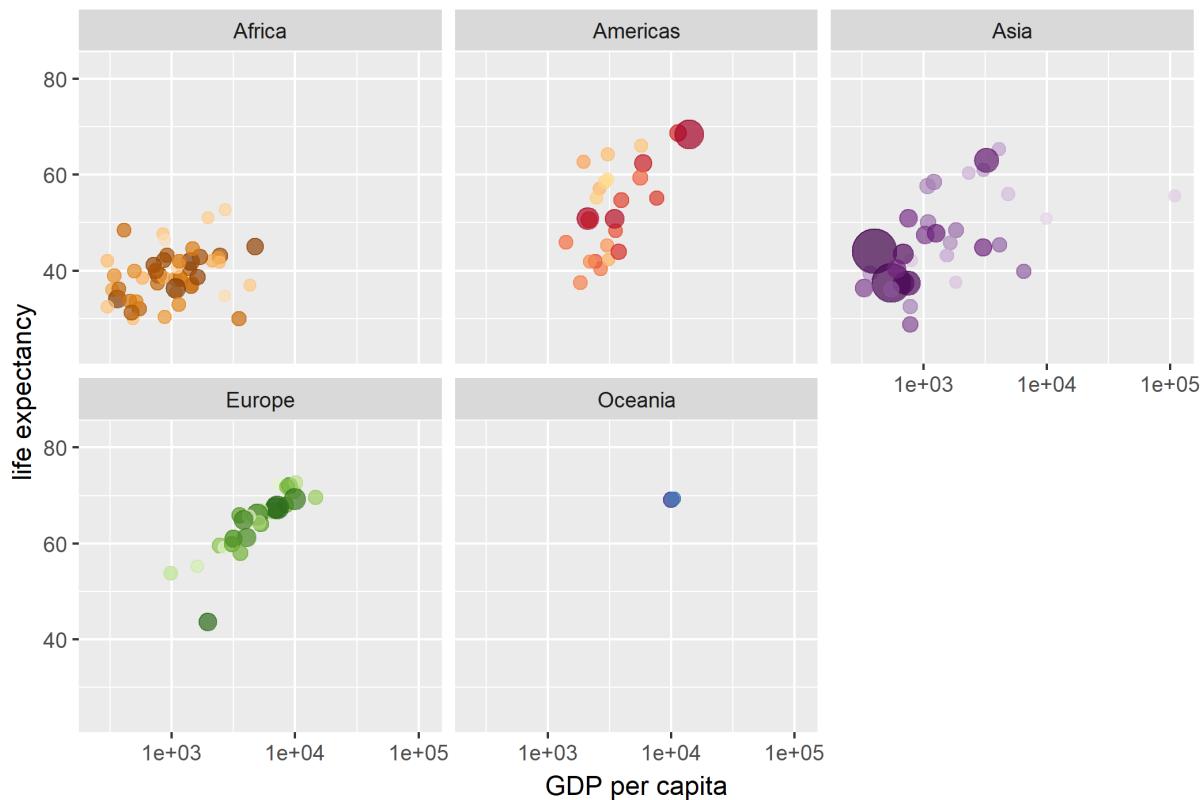
```
Frame 1 (1%)
Frame 2 (2%)
Frame 3 (3%)
Frame 4 (4%)
Frame 5 (5%)
Frame 6 (6%)
```

Frame 7 (7%)  
Frame 8 (8%)  
Frame 9 (9%)  
Frame 10 (10%)  
Frame 11 (11%)  
Frame 12 (12%)  
Frame 13 (13%)  
Frame 14 (14%)  
Frame 15 (15%)  
Frame 16 (16%)  
Frame 17 (17%)  
Frame 18 (18%)  
Frame 19 (19%)  
Frame 20 (20%)  
Frame 21 (21%)  
Frame 22 (22%)  
Frame 23 (23%)  
Frame 24 (24%)  
Frame 25 (25%)  
Frame 26 (26%)  
Frame 27 (27%)  
Frame 28 (28%)  
Frame 29 (29%)  
Frame 30 (30%)  
Frame 31 (31%)  
Frame 32 (32%)  
Frame 33 (33%)  
Frame 34 (34%)  
Frame 35 (35%)  
Frame 36 (36%)  
Frame 37 (37%)  
Frame 38 (38%)  
Frame 39 (39%)  
Frame 40 (40%)  
Frame 41 (41%)  
Frame 42 (42%)  
Frame 43 (43%)  
Frame 44 (44%)  
Frame 45 (45%)  
Frame 46 (46%)  
Frame 47 (47%)  
Frame 48 (48%)  
Frame 49 (49%)  
Frame 50 (50%)  
Frame 51 (51%)  
Frame 52 (52%)  
Frame 53 (53%)

Frame 54 (54%)  
Frame 55 (55%)  
Frame 56 (56%)  
Frame 57 (57%)  
Frame 58 (58%)  
Frame 59 (59%)  
Frame 60 (60%)  
Frame 61 (61%)  
Frame 62 (62%)  
Frame 63 (63%)  
Frame 64 (64%)  
Frame 65 (65%)  
Frame 66 (66%)  
Frame 67 (67%)  
Frame 68 (68%)  
Frame 69 (69%)  
Frame 70 (70%)  
Frame 71 (71%)  
Frame 72 (72%)  
Frame 73 (73%)  
Frame 74 (74%)  
Frame 75 (75%)  
Frame 76 (76%)  
Frame 77 (77%)  
Frame 78 (78%)  
Frame 79 (79%)  
Frame 80 (80%)  
Frame 81 (81%)  
Frame 82 (82%)  
Frame 83 (83%)  
Frame 84 (84%)  
Frame 85 (85%)  
Frame 86 (86%)  
Frame 87 (87%)  
Frame 88 (88%)  
Frame 89 (89%)  
Frame 90 (90%)  
Frame 91 (91%)  
Frame 92 (92%)  
Frame 93 (93%)  
Frame 94 (94%)  
Frame 95 (95%)  
Frame 96 (96%)  
Frame 97 (97%)  
Frame 98 (98%)  
Frame 99 (99%)  
Frame 100 (100%)

Finalizing encoding... done!

Year: 1952

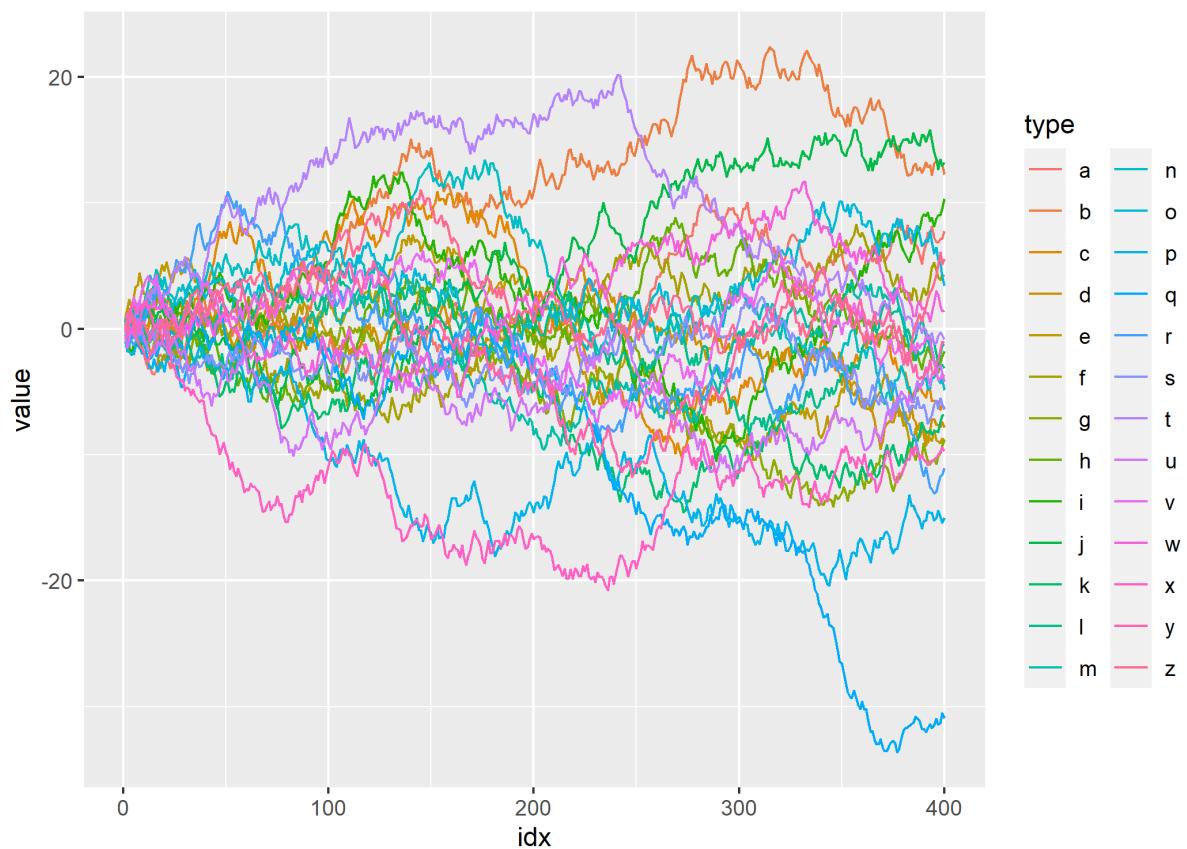


Voir le site de l'extension (<https://g ganimate.com/>) pour la documentation et des tutoriels. Il est conseillé d'installer également l'extension `gifsiki` avec `g ganimate`.

## Surligner certaines données

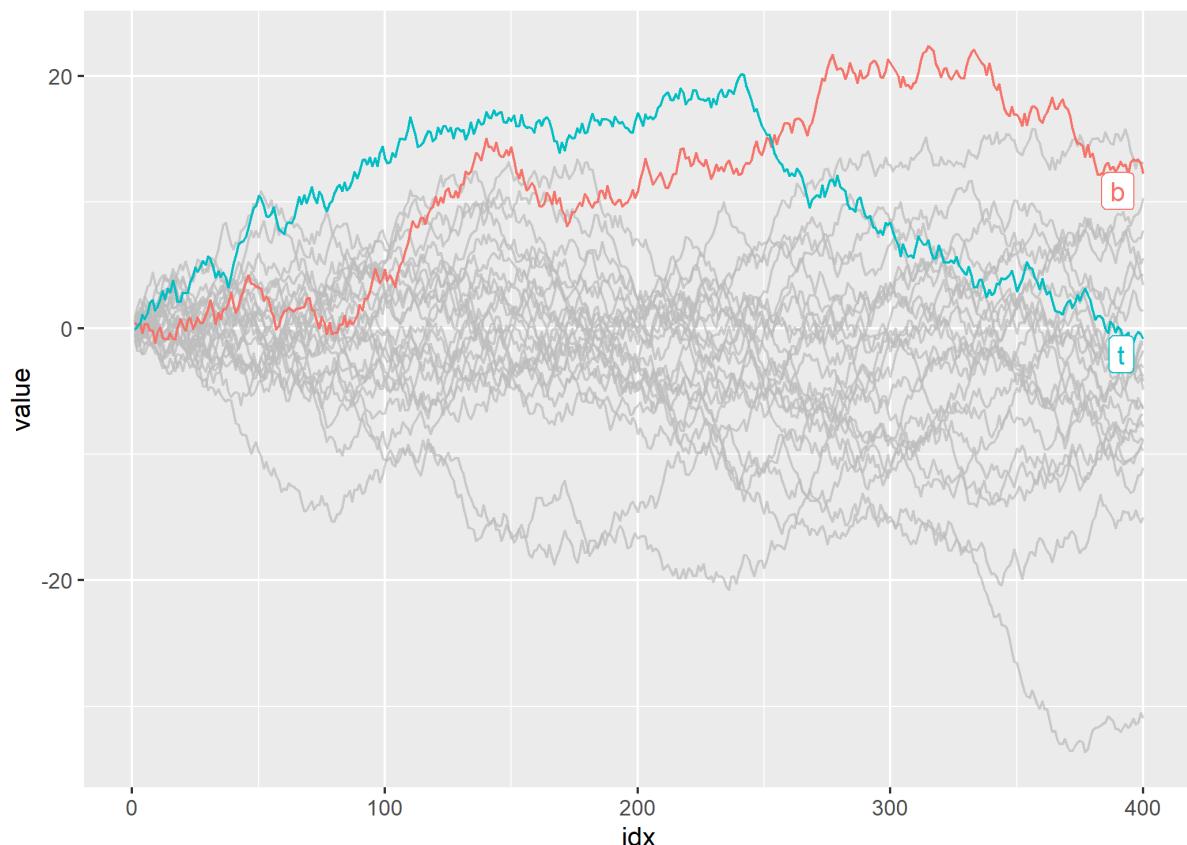
L'extension `gghighlight` fournit une fonction `gghighlight` qui permet de «surligner» les données qui remplissent des conditions spécifiées.

```
R> d <- purrr::map_dfr(  
+   letters,  
+   ~ data.frame(  
+     idx = 1:400,  
+     value = cumsum(runif(400, -1, 1)),  
+     type = .,  
+     flag = sample(c(TRUE, FALSE), size = 400, replace = TRUE),  
+     stringsAsFactors = FALSE  
+   )  
)  
  
ggplot(d) +  
  aes(x = idx, y = value, colour = type) +  
  geom_line()
```

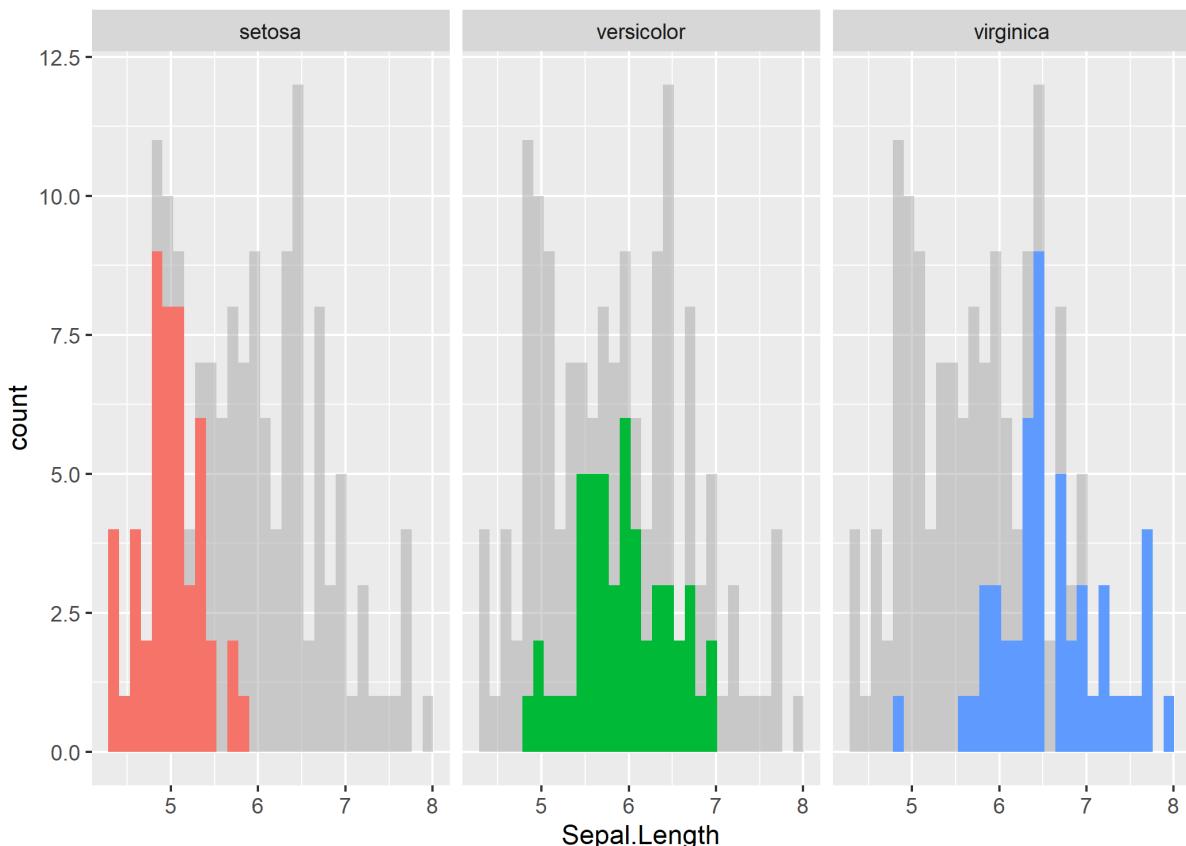


```
R> library(gghighlight)
ggplot(d) +
  aes(x = idx, y = value, colour = type) +
  geom_line() +
  gghighlight(max(value) > 20)
```

label\_key: type



```
R> ggplot(iris, aes(Sepal.Length, fill = Species)) +  
  geom_histogram() +  
  gghighlight() +  
  facet_wrap(~Species)
```



## Thèmes et couleurs

### Palettes de couleurs

Voir le chapitre Couleurs et palettes, page 1021 pour une sélection d’extensions proposant des palettes de couleurs additionnelles.

### hrbrthemes

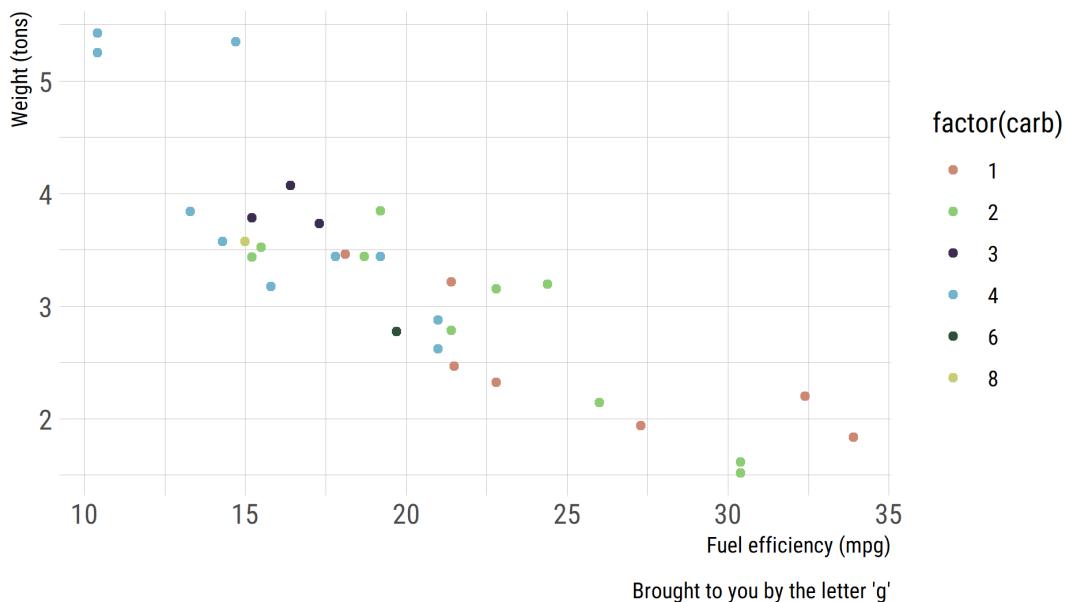
L’extension [hrbrthemes](#) fournit plusieurs thèmes graphiques pour [ggplot2](#). Un exemple ci-dessous. Pour

plus d'informations, voir <https://github.com/hrbrmstr/hrbrthemes>.

```
R> library(ggplot2)
library(hrbrthemes)
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(color = factor(carb))) +
  labs(
    x = "Fuel efficiency (mpg)", y = "Weight (tons)",
    title = "Seminal ggplot2 scatterplot example",
    subtitle = "A plot that is only useful for demonstration purposes",
    caption = "Brought to you by the letter 'g'"
  ) +
  scale_color_ipsum() +
  theme_ipsum_rc()
```

## Seminal ggplot2 scatterplot example

A plot that is only useful for demonstration purposes



## ggthemes

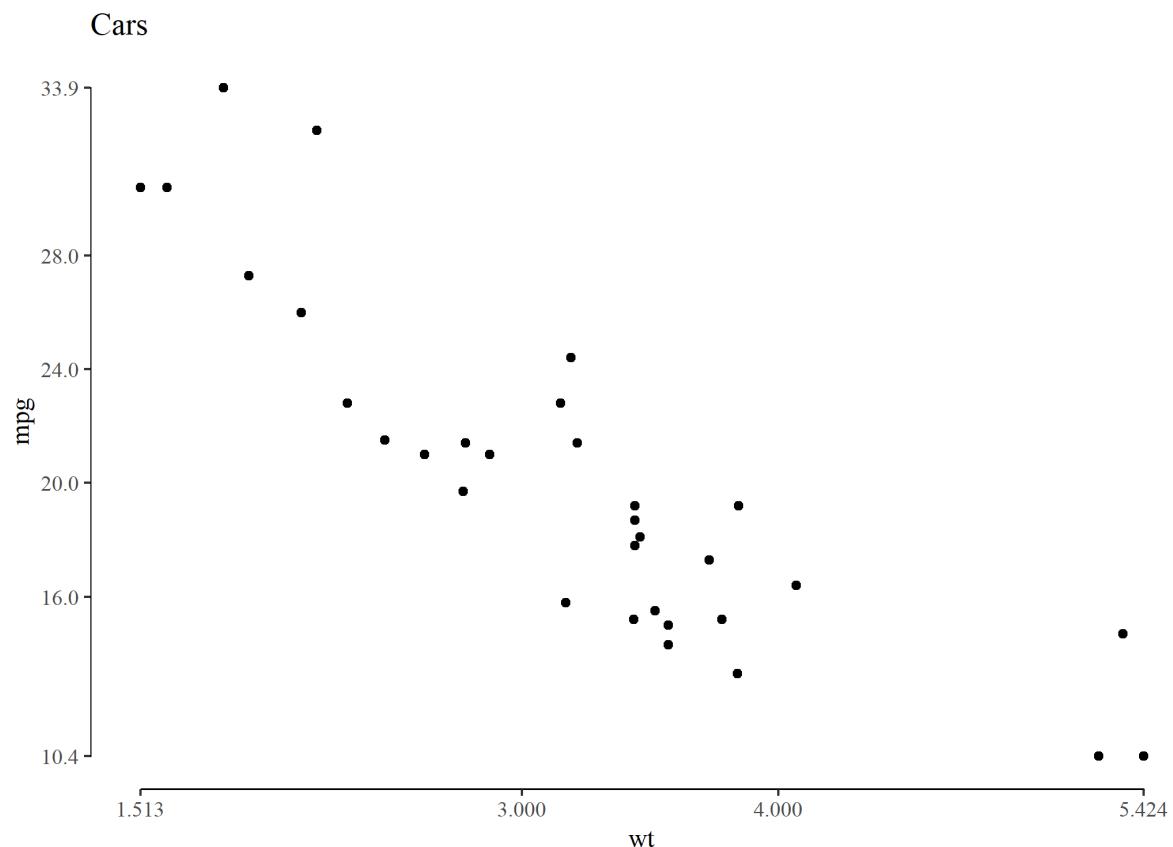
**ggthemes** propose une vingtaine de thèmes différentes présentés sur le site de l'extension : <https://jrnold.github.io/ggthemes/>.

Voir ci-dessous un exemple du thème `theme_tufte` inspiré d'Edward Tufte.

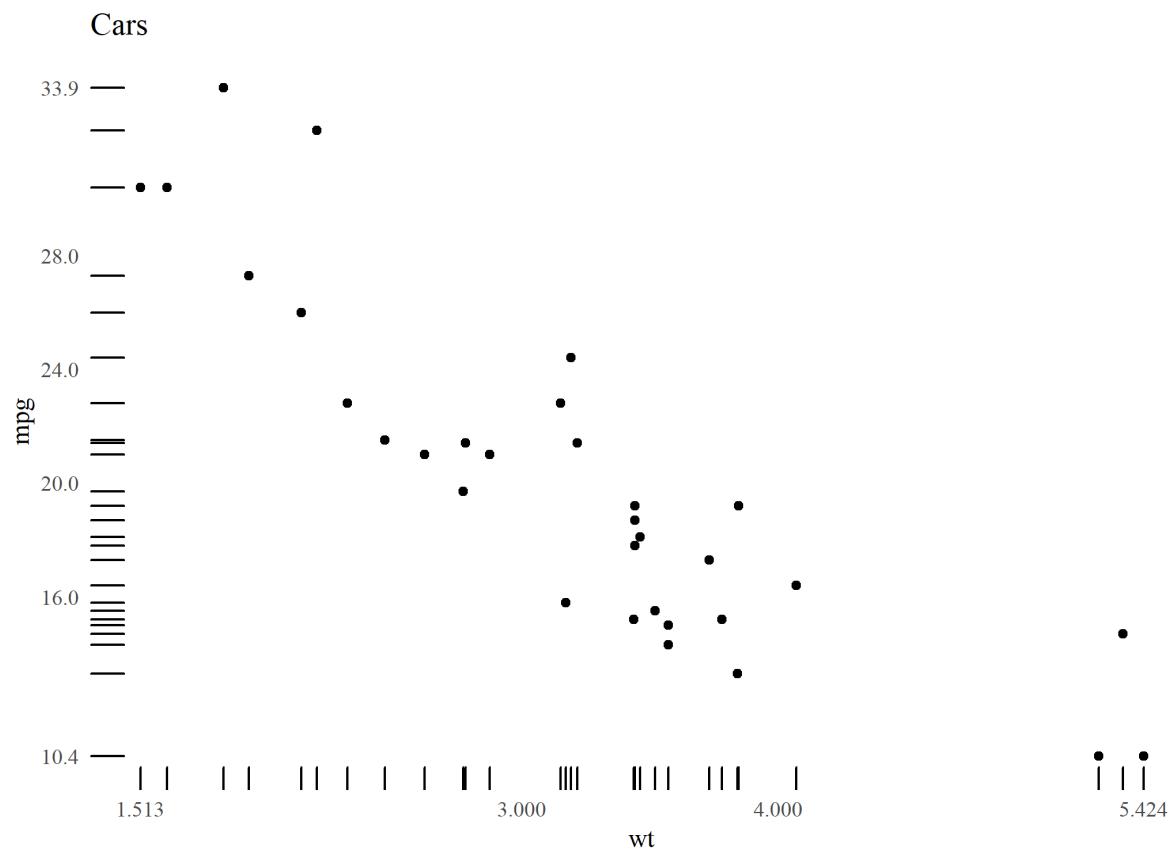
```
R> library(ggplot2)
  library(ggthemes)

p <- ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  scale_x_continuous(breaks = extended_range_breaks()(mtcars$wt)) +
  scale_y_continuous(breaks = extended_range_breaks()(mtcars$mpg)) +
  ggtitle("Cars")

p + geom_rangeframe() +
  theme_tufte()
```



```
R> p + geom_rug() +  
  theme_tufte(ticks = FALSE)
```



## Combiner plusieurs graphiques

Voir le chapitre dédié, page 825.



# Combiner plusieurs graphiques

plot_grid (cowplot) . . . . .	825
patchwork . . . . .	829
multiplot (JLutils) . . . . .	830
Légende partagée entre plusieurs graphiques . . . . .	835

Vous savez réaliser des graphiques avec **ggplot2**, page 395 ? Il est très facile de combiner plusieurs graphiques en un seul.

Commençons par créer quelques graphiques avec **ggplot2**.

```
R> library(ggplot2)
p1 <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point()
p2 <- ggplot(mtcars, aes(factor(cyl))) +
  geom_bar()
p3 <- ggplot(mtcars, aes(factor(cyl), mpg)) +
  geom_violin()
p4 <- ggplot(mtcars, aes(factor(cyl), mpg)) +
  geom_boxplot()
```

## plot\_grid (cowplot)

L'extension **cowplot** propose une fonction `plot_grid`. Son usage est expliqué en détail dans les vignettes dédiées incluses avec l'extension : <https://wilkelab.org/cowplot/>.

```
R> library(cowplot)
```

```
Attaching package: 'cowplot'
```

```
The following object is masked from 'package:ggpubr':
```

```
get_legend
```

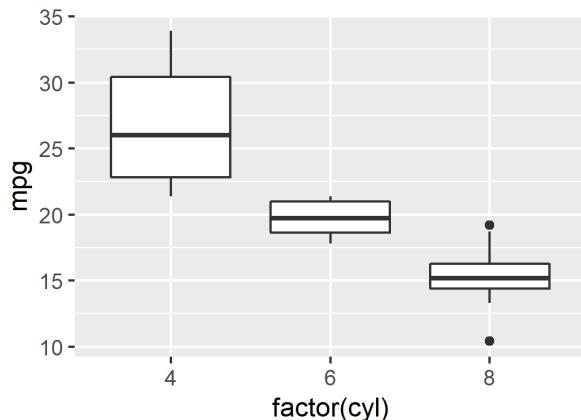
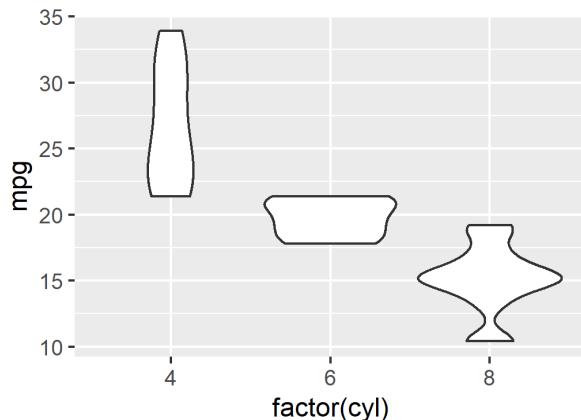
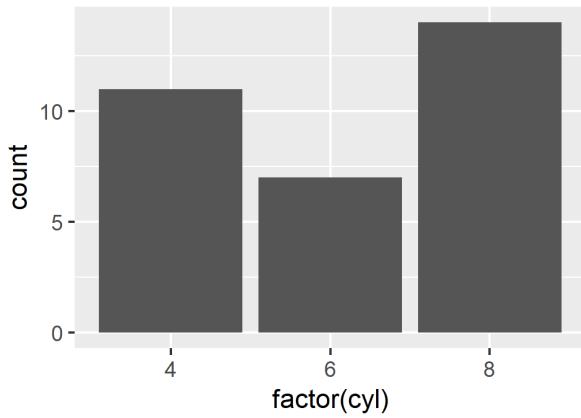
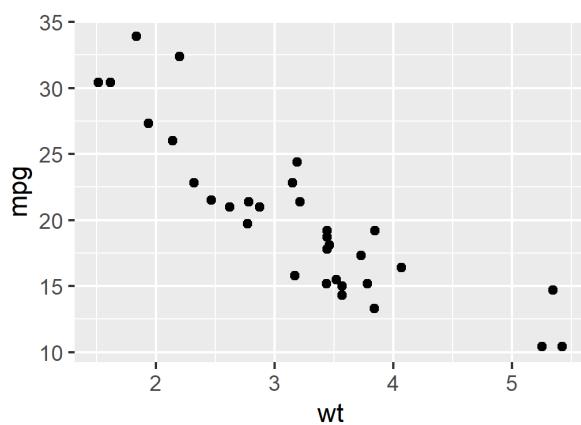
```
The following object is masked from 'package:lubridate':
```

```
stamp
```

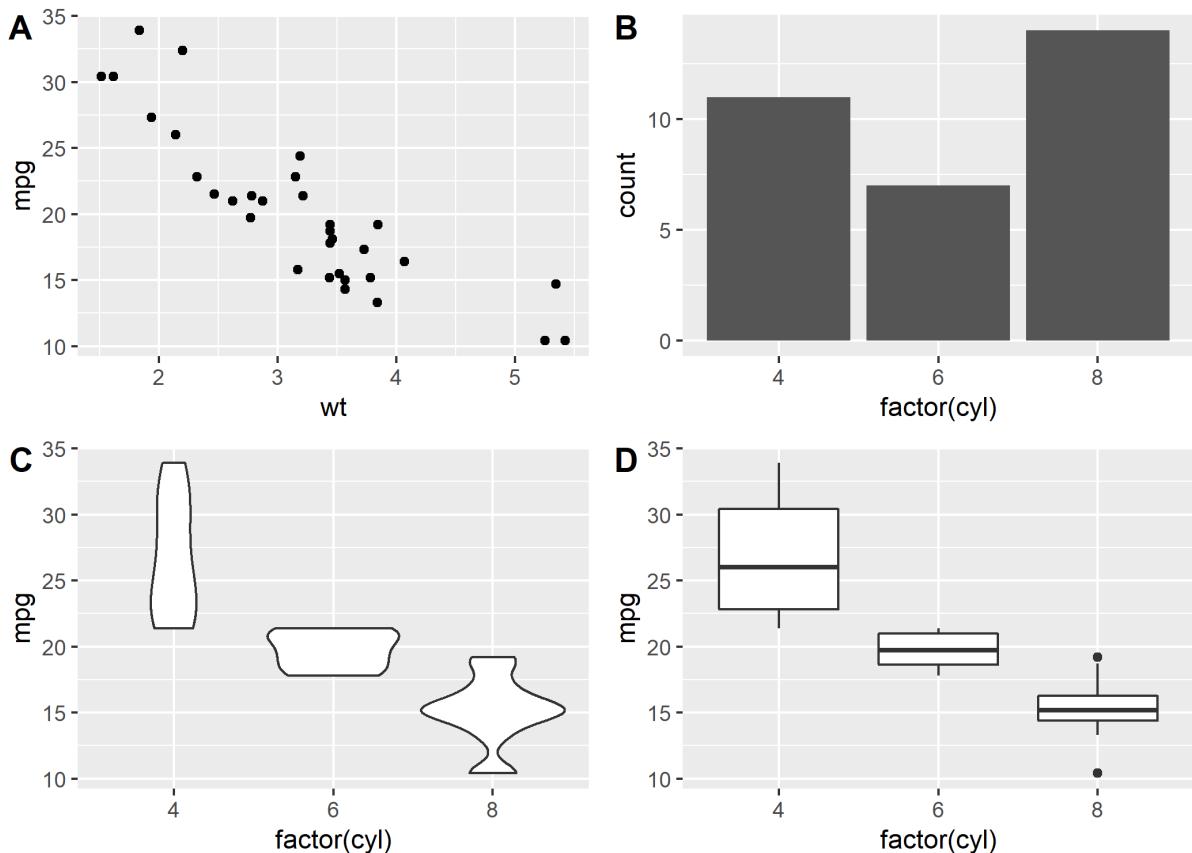
```
The following object is masked from 'package:JLutils':
```

```
get_legend
```

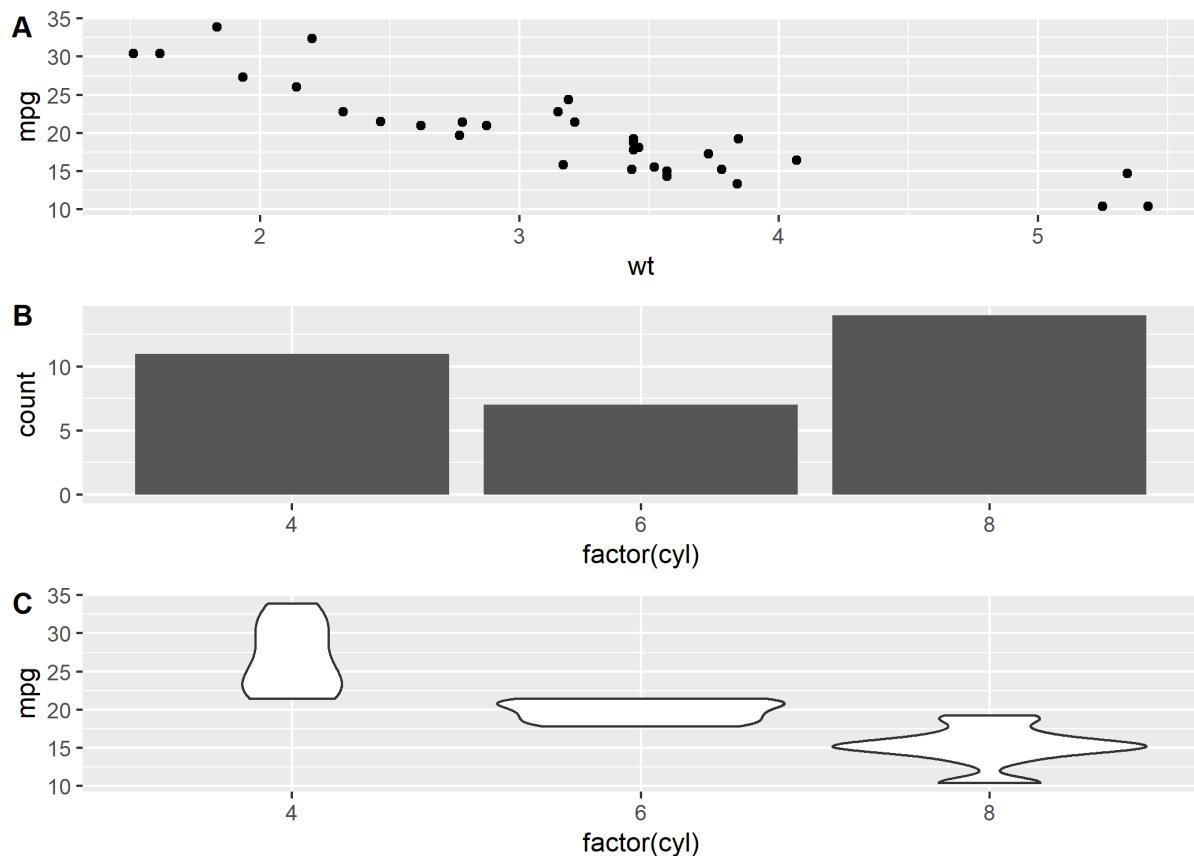
```
R> # simple grid  
plot_grid(p1, p2, p3, p4)
```



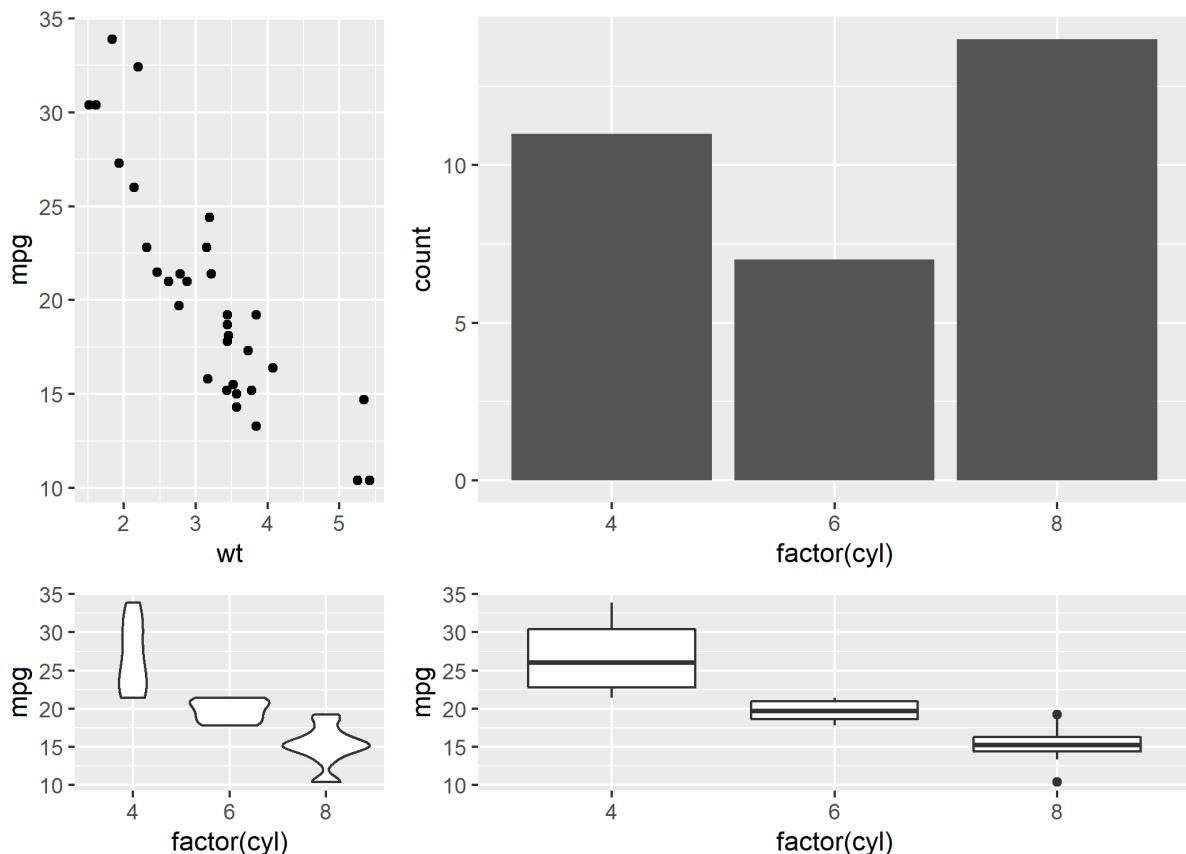
```
R> # simple grid with labels and aligned plots  
plot_grid(p1, p2, p3, p4, labels = c("A", "B", "C", "D"), align = "hv")
```



```
R> # manually setting the number of rows, auto-generate upper-case labels  
plot_grid(p1, p2, p3, nrow = 3, labels = "AUTO", label_size = 12, align = "v")
```



```
R> # making rows and columns of different widths/heights
plot_grid(p1, p2, p3, p4, align = "hv", rel_heights = c(2, 1), rel_widths =
c(1, 2))
```



On notera en passant que le chargement de `cowplot` modifie le style par défaut des graphiques `ggplot2`. Voir <https://cran.r-project.org/web/packages/cowplot/vignettes/introduction.html>.

## patchwork

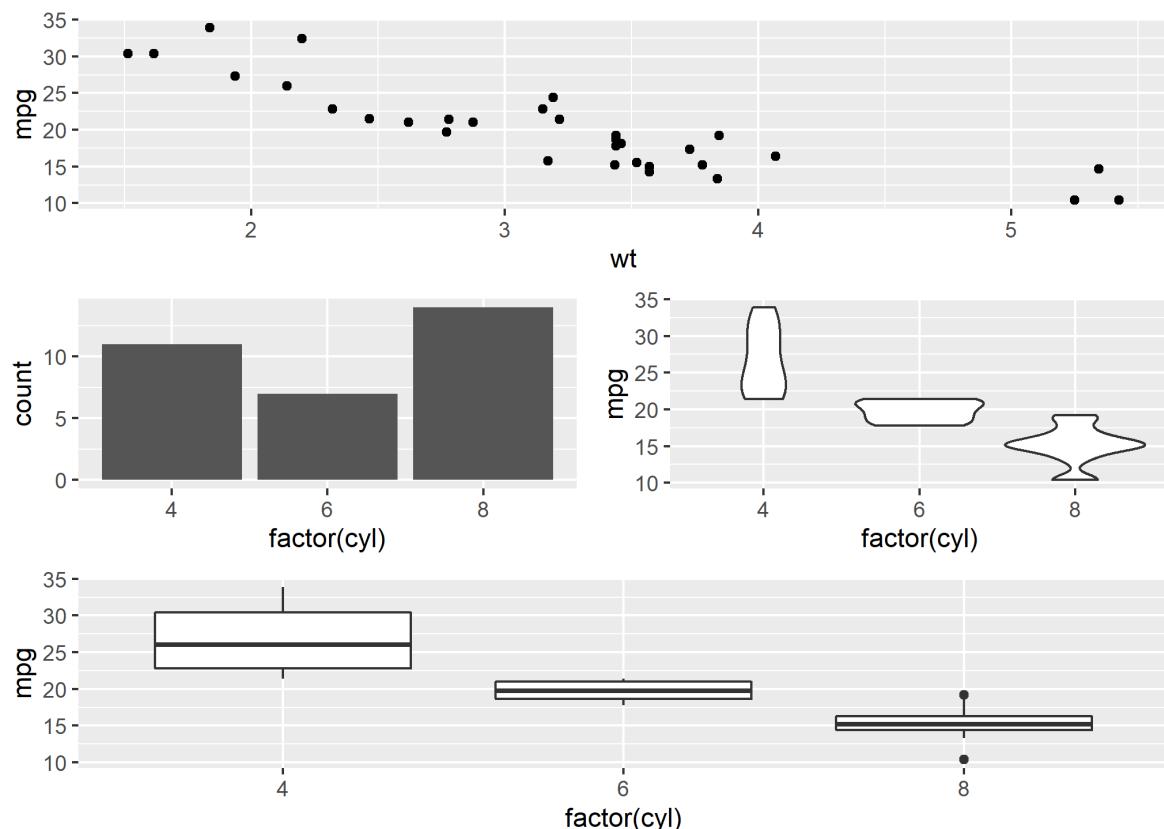
Citons également l'extension `patchwork`, disponible sur GitHub (<https://github.com/thomasp85/patchwork>) qui propose une syntaxe un petit peu différente, par «addition» de graphiques.

```
R> library(patchwork)
```

```
Attaching package: 'patchwork'
```

```
The following object is masked from 'package:cowplot':
```

```
R> p1 + (p2 + p3) + p4 + plot_layout(ncol = 1)
```



## multiplot (JLutils)

Dans son ouvrage *Cookbook for R*, Winston Chang propose une fonction `multiplot` pour combiner plusieurs graphiques<sup>1</sup>

L’extension `JLutils` disponible sur [GitHub](#) propose une version améliorée de cette fonction.

Pour installer `JLutils` si non disponible sur votre PC, copier/coller le code ci-dessous.

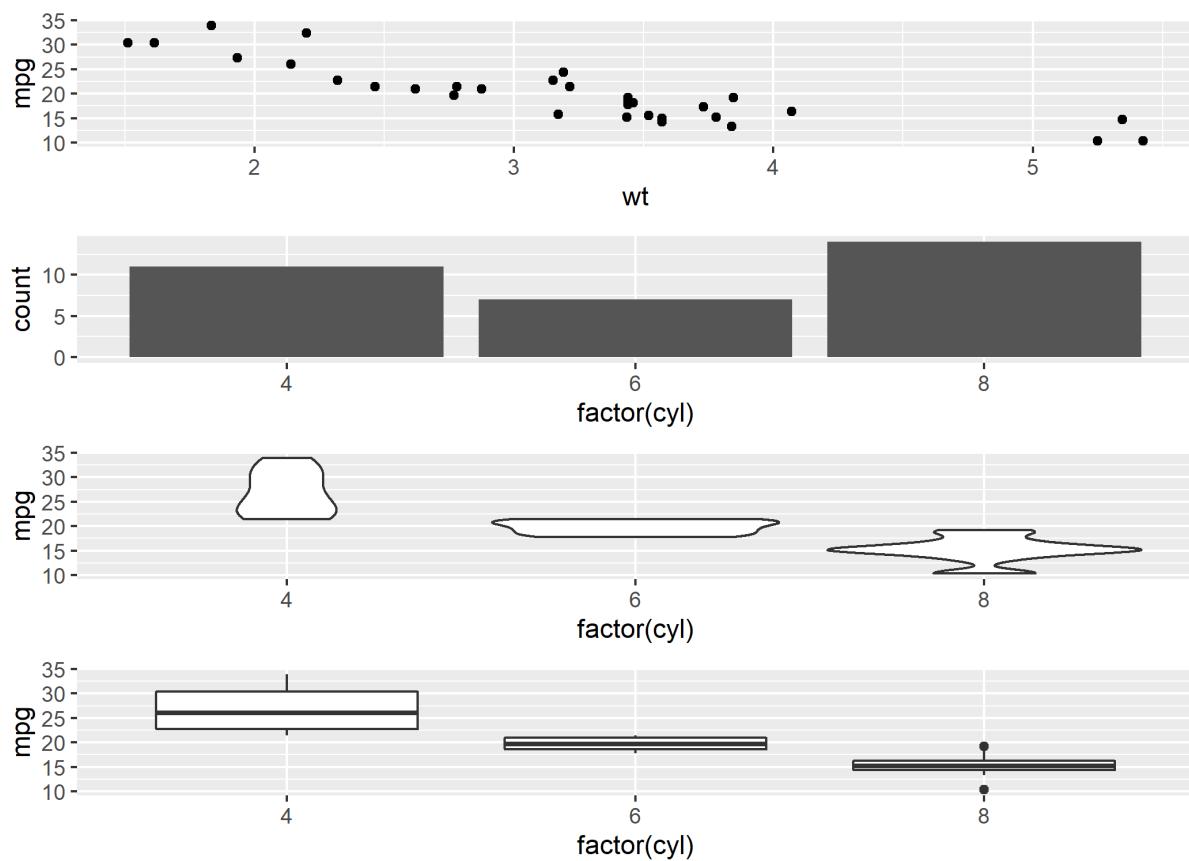
---

1. Voir [http://www.cookbook-r.com/Graphs/Multiple\\_graphs\\_on\\_one\\_page\\_\(ggplot2\)/](http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_(ggplot2)/).

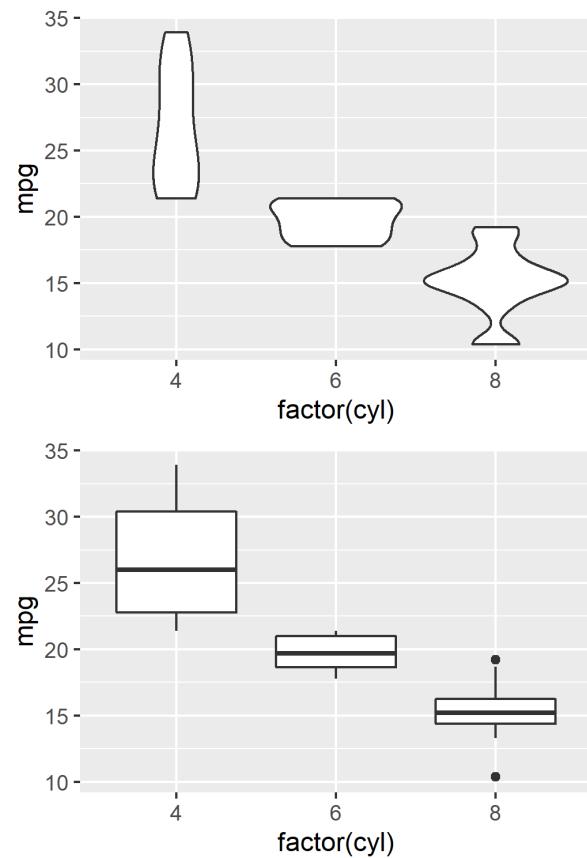
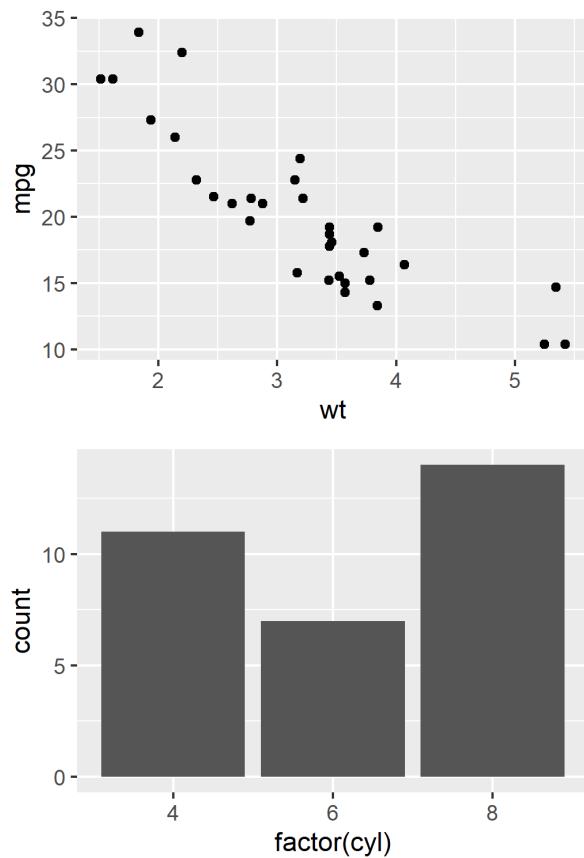
```
R> if (!require(devtools)) {
  install.packages("devtools")
  library(devtools)
}
install_github("lamarange/JLutils")
```

Parce que quelques exemples valent mieux qu'un long discours.

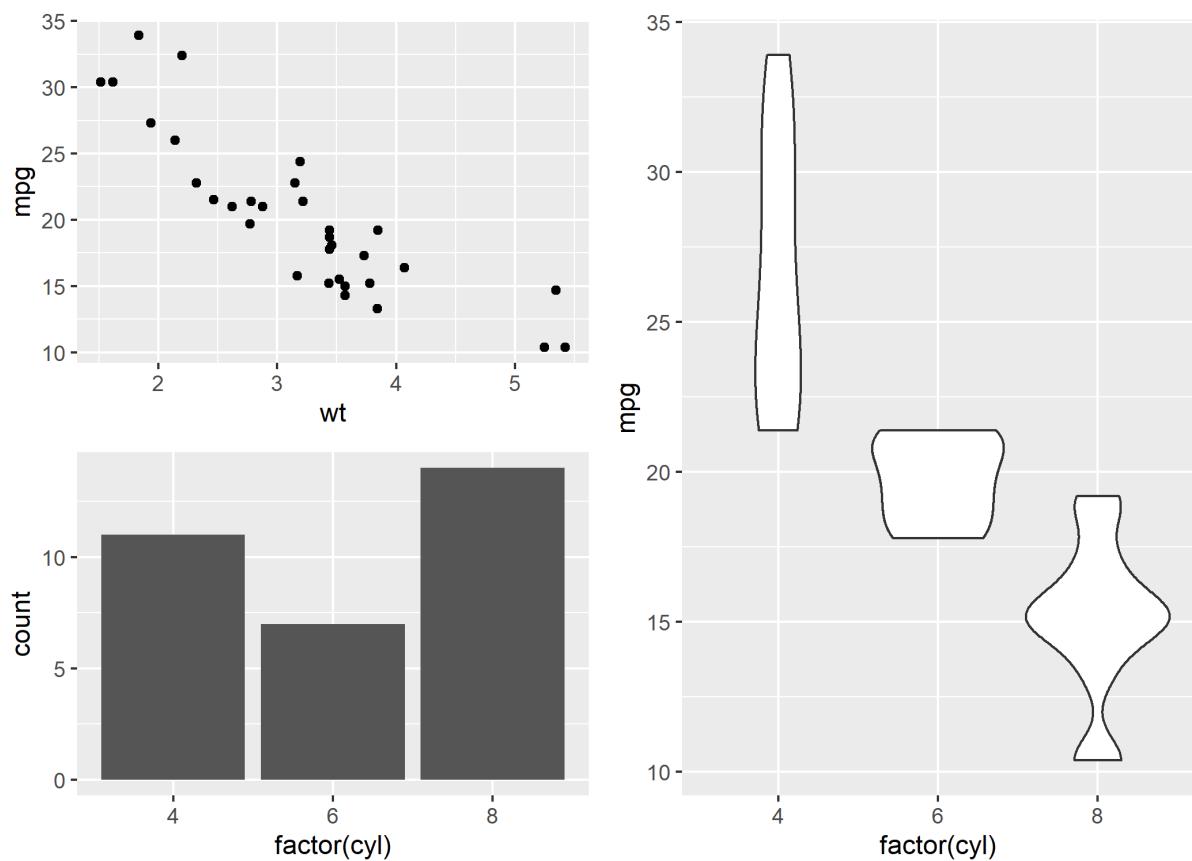
```
R> library(JLutils)
multiplot(p1, p2, p3, p4)
```



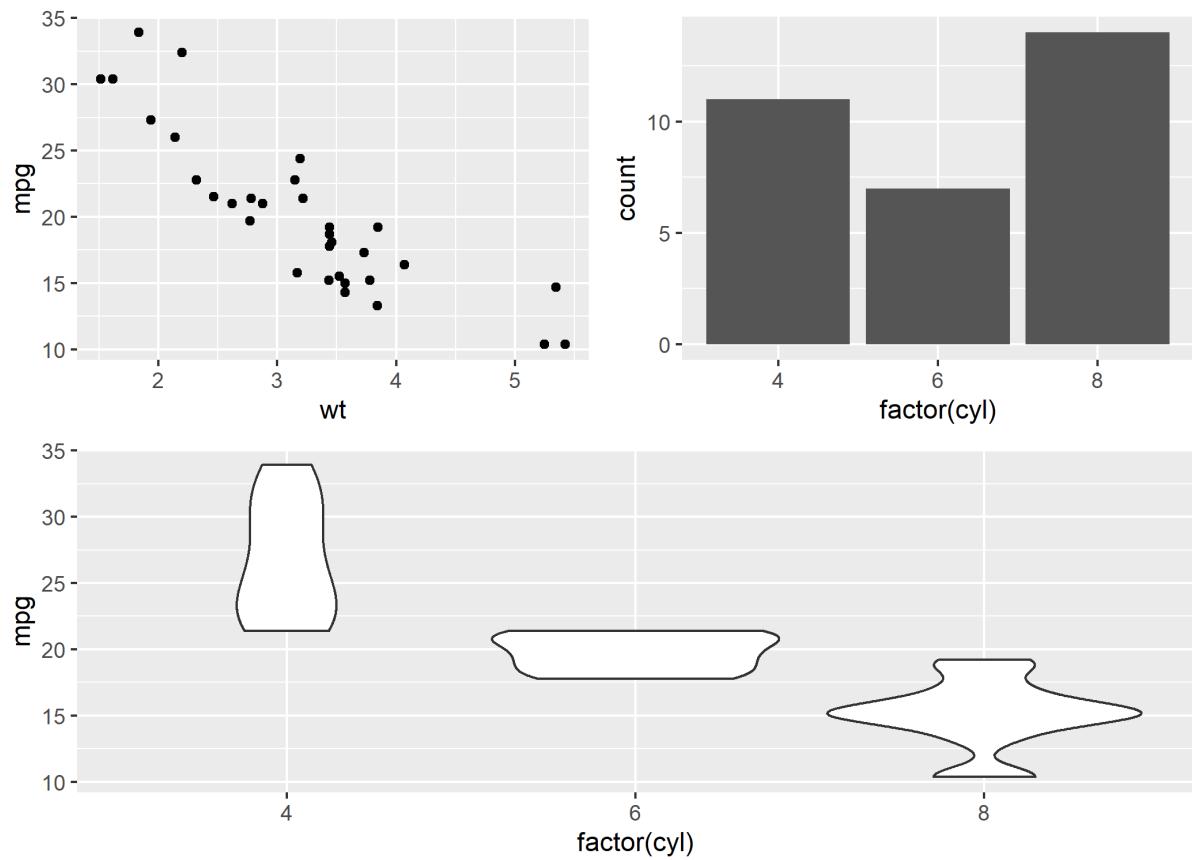
```
R> multiplot(p1, p2, p3, p4, cols = 2)
```



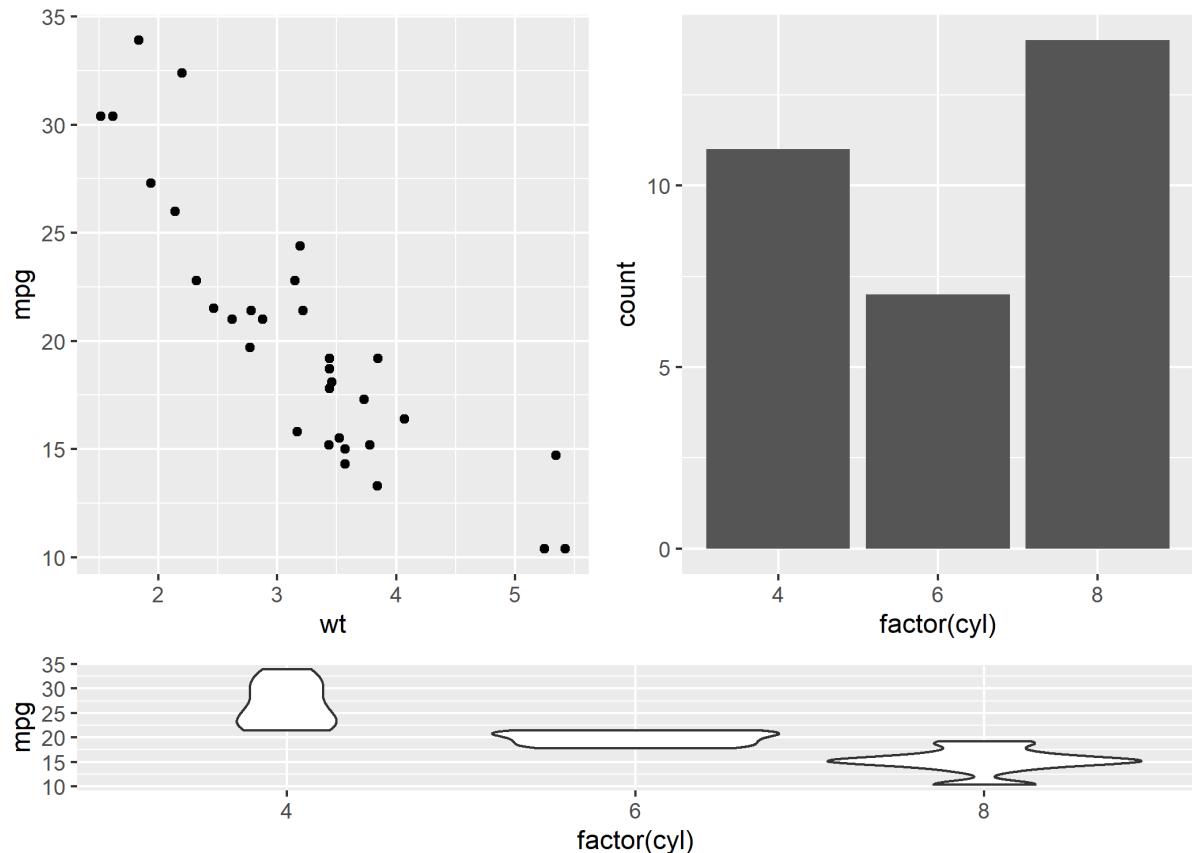
```
R> multiplot(p1, p2, p3, layout = matrix(c(1, 2, 3, 3), nrow = 2))
```



```
R> multiplot(p1, p2, p3, layout = matrix(c(1, 2, 3, 3), nrow = 2, byrow = TRUE))
```



```
R> multiplot(p1, p2, p3, layout = matrix(c(1, 2, 3, 3), nrow = 2, byrow = TRUE),
heights = c(3, 1))
```



## Légende partagée entre plusieurs graphiques

JLutils et cowplot fournissent tous deux une fonction `get_legend` permettant d'extraire la légende d'un graphique puis de l'utiliser avec `multiplot` ou `plot_grid`.

Créons quelques graphiques.

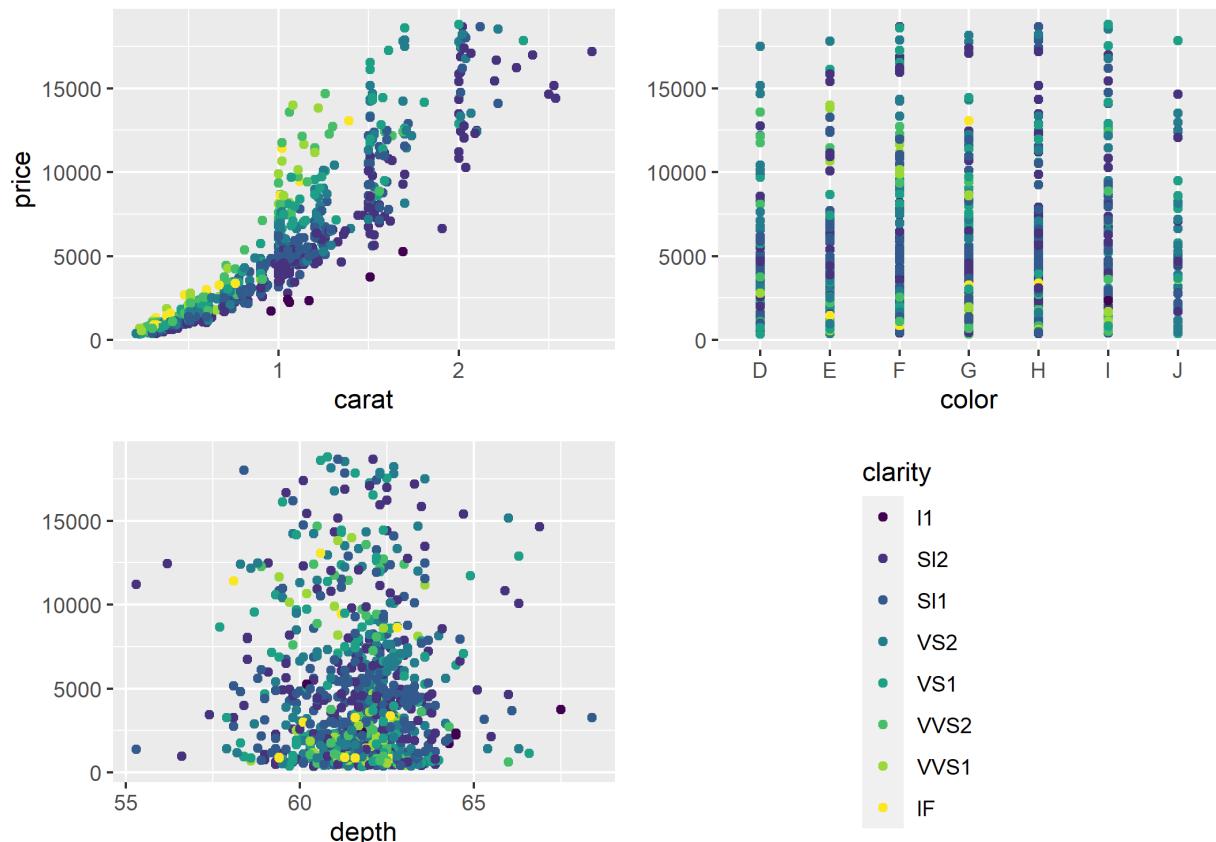
```
R> dsamp <- diamonds[sample(nrow(diamonds), 1000), ]
p1 <- qplot(carat, price, data = dsamp, colour = clarity) +
  theme(plot.margin = unit(c(6, 0, 6, 0), "pt"))
p2 <- qplot(depth, price, data = dsamp, colour = clarity) +
  theme(plot.margin = unit(c(6, 0, 6, 0), "pt")) + ylab("")
p3 <- qplot(color, price, data = dsamp, colour = clarity) +
  theme(plot.margin = unit(c(6, 0, 6, 0), "pt")) + ylab("")
```

Récupérons la légende du premier graphique puis supprimons là dans les trois graphiques.

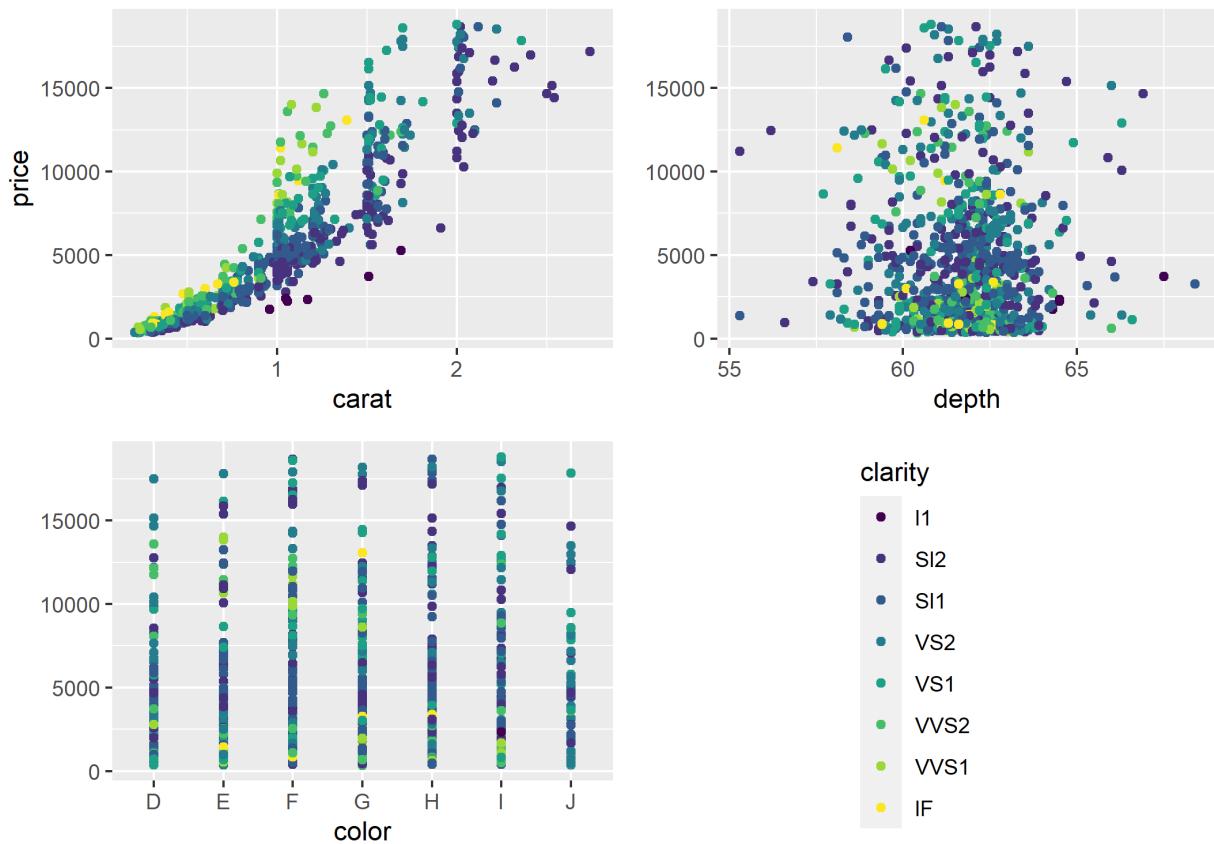
```
R> leg <- get_legend(p1)
p1 <- p1 + theme(legend.position = "none")
p2 <- p2 + theme(legend.position = "none")
p3 <- p3 + theme(legend.position = "none")
```

Combinons le tout.

```
R> multiplot(p1, p2, p3, leg, cols = 2)
```



```
R> plot_grid(p1, p2, p3, leg, ncol = 2)
```



Enfin, citons également la fonction `grid_arrange_shared_legend` de l'extension `lemon`<sup>2</sup>.

---

2. `lemon` fournit également diverses fonctions pour manipuler des graphiques `ggplot2`, comme par exemple la possibilité de répéter les axes quand on utilise des facettes.



# Exemples de graphiques avancés

Questions de connaissance . . . . .	839
Code final du graphique . . . . .	870

## NOTE

Ce chapitre est en cours de rédaction.

Dans ce chapitre, nous présentons plusieurs graphiques avec une mise en forme avancée et détaillons pas à pas leur création.

Chargeons quelques extensions de base.

```
R> library(tidyverse)
```

## Questions de connaissance

Pour ce premier exemple, supposons que nous avons réalisé une petite enquête auprès de 500 étudiants pour mesurer leur connaissance du logiciel R. Commençons par charger les données ici purement fictionnelles).

```
R> load(url("https://larmarange.github.io/analyse-R/data/connaissances.RData"))
```

Nous avons maintenant un objet `quest` en mémoire. Regardons rapidement la forme des données.

```
R> glimpse(quest)
```

```
Rows: 500
Columns: 15
$ id      <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ...
$ conn_a   <fct> non, non, non, non, NA, non, non, non, ...
$ conn_b   <fct> oui, oui, oui, oui, oui, oui, oui, ...
$ conn_c   <fct> oui, oui, oui, oui, oui, oui, oui, ...
$ conn_d   <fct> non, non, non, NA, NA, NA, NA, NA, non...
$ conn_e   <fct> oui, oui, oui, oui, NA, oui, oui, oui, ...
$ conn_f   <fct> oui, non, oui, NA, NA, oui, oui, oui, ...
$ conn_g   <fct> non, oui, oui, oui, oui, oui, oui, ...
$ source_a <fct> oui, oui, oui, oui, oui, oui, oui, ...
$ source_b <fct> oui, non, non, non, non, oui, non, oui...
$ source_c <fct> oui, non, oui, oui, non, non, non, non...
$ source_d <fct> oui, non, non, oui, non, oui, oui, ...
$ source_e <fct> oui, non, non, oui, oui, non, oui, non...
$ source_f <fct> non, non, non, non, non, non, non, ...
$ source_g <fct> non, non, non, non, non, non, non, ...
```

```
R> summary(quest$conn_a)
```

oui	non	NSP	NA's
36	442	1	21

Sept affirmations ont été soumises aux étudiants (variables *conn\_a* à *conn\_g*) et il leur a été demandé, pour chacune, s’il pensait qu’elle était juste. Les réponses possibles étaient “oui”, “non” et “NSP” (ne sait pas).

```
R> library(questionr)
freq.na(quest)
```

	missing	%
conn_d	104	21
conn_a	21	4
conn_e	17	3
conn_g	12	2
conn_f	10	2
id	0	0
conn_b	0	0
conn_c	0	0
source_a	0	0
source_b	0	0

```

source_c      0  0
source_d      0  0
source_e      0  0
source_f      0  0
source_g      0  0

```

On peut également noter que pour certaines questions il y a plusieurs valeurs manquantes (jusqu'à 104).

Nous souhaiterions représenter les réponses sous la forme d'un graphique en barres cumulées. Cependant, un tel graphique ne peut pour le moment être réalisé car les réponses sont stockées dans 7 variables différentes. Pour réaliser le graphique, il nous faut un tableau de données avec une colonne qui contiendrait le nom de la question et une colonne avec les réponses. Cela peut se faire facilement avec la fonction `pivot_longer` de l'extension `tidyverse` (voir le chapitre dédié, page 299).

```

R> conn <- quest %>%
  select(starts_with("conn_")) %>%
  pivot_longer(
    cols = starts_with("conn_"),
    names_to = "question",
    values_to = "reponse"
  )
glimpse(conn)

```

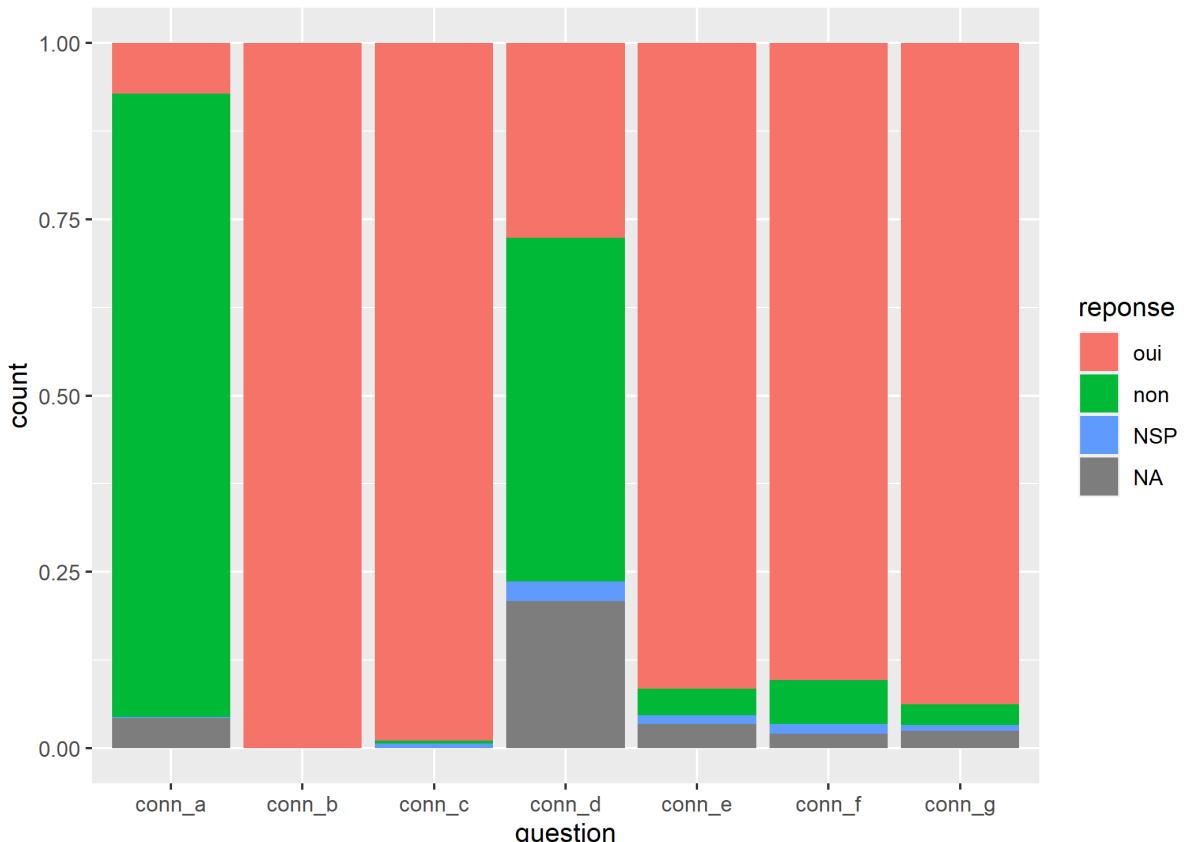
```

Rows: 3,500
Columns: 2
$ question <chr> "conn_a", "conn_b", "conn_c", "conn_d"...
$ reponse   <fct> non, oui, oui, non, oui, oui, non, non...

```

Nous pouvons maintenant réaliser une première ébauche avec `geom_bar` et `position = "fill"`.

```
R> ggplot(conn) +
  aes(x = question, fill = reponse) +
  geom_bar(position = "fill")
```

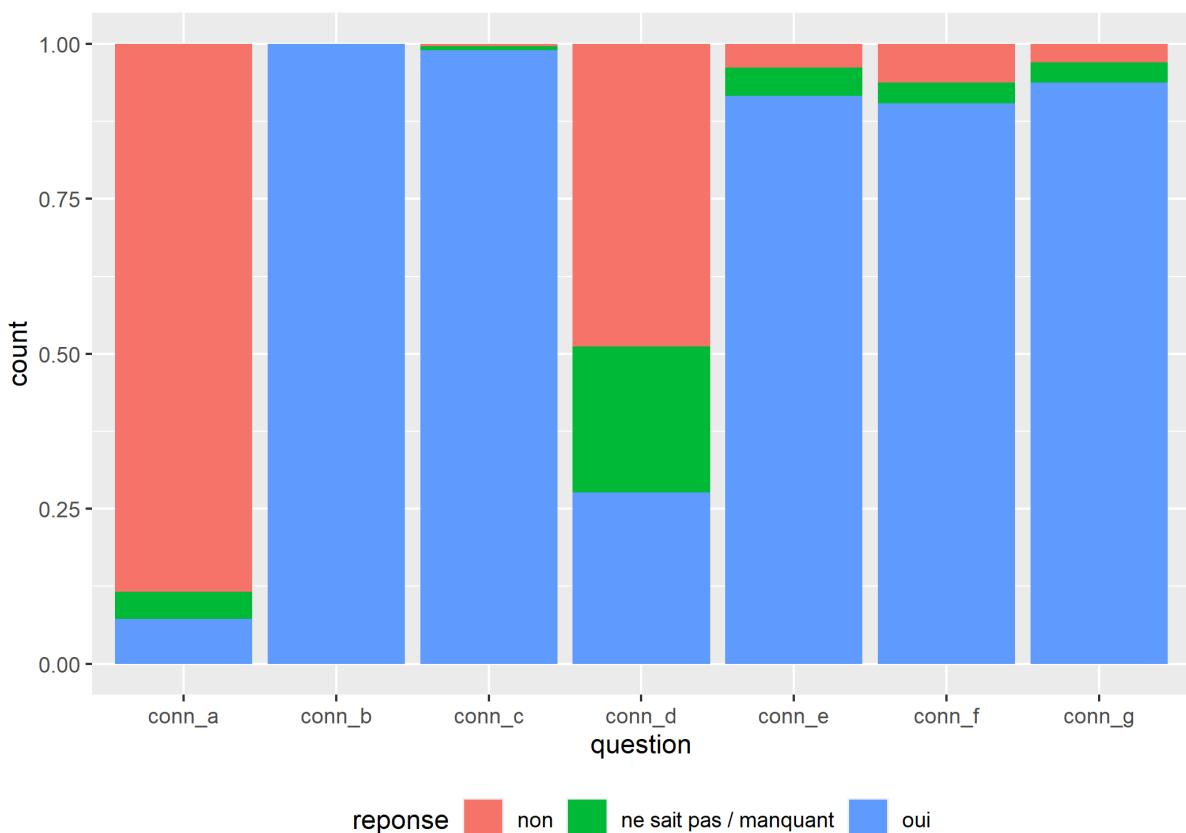


Pour simplifier le graphique, nous allons regrouper les manquants avec les NSP avec `fct_explicit_na`, changer l’ordre des modalités en mettant les NSP entre les oui et les non avec `fct_relevel` et remplacer “NSP” par sa version longue avec `fct_recode`. En effet, il est toujours préférable, pour la lisibilité du graphique, d’éviter un acronyme lorsque ce n’est pas nécessaire.

Nous allons également en profiter pour déplacer la légende sous le graphique avec l’option `legend.position = "bottom"` passée à `theme`.

```
R> conn$reponse <- conn$reponse %>%
  fct_explicit_na("NSP") %>%
  fct_relevel("non", "NSP", "oui") %>%
  fct_recode("ne sait pas / manquant" = "NSP")

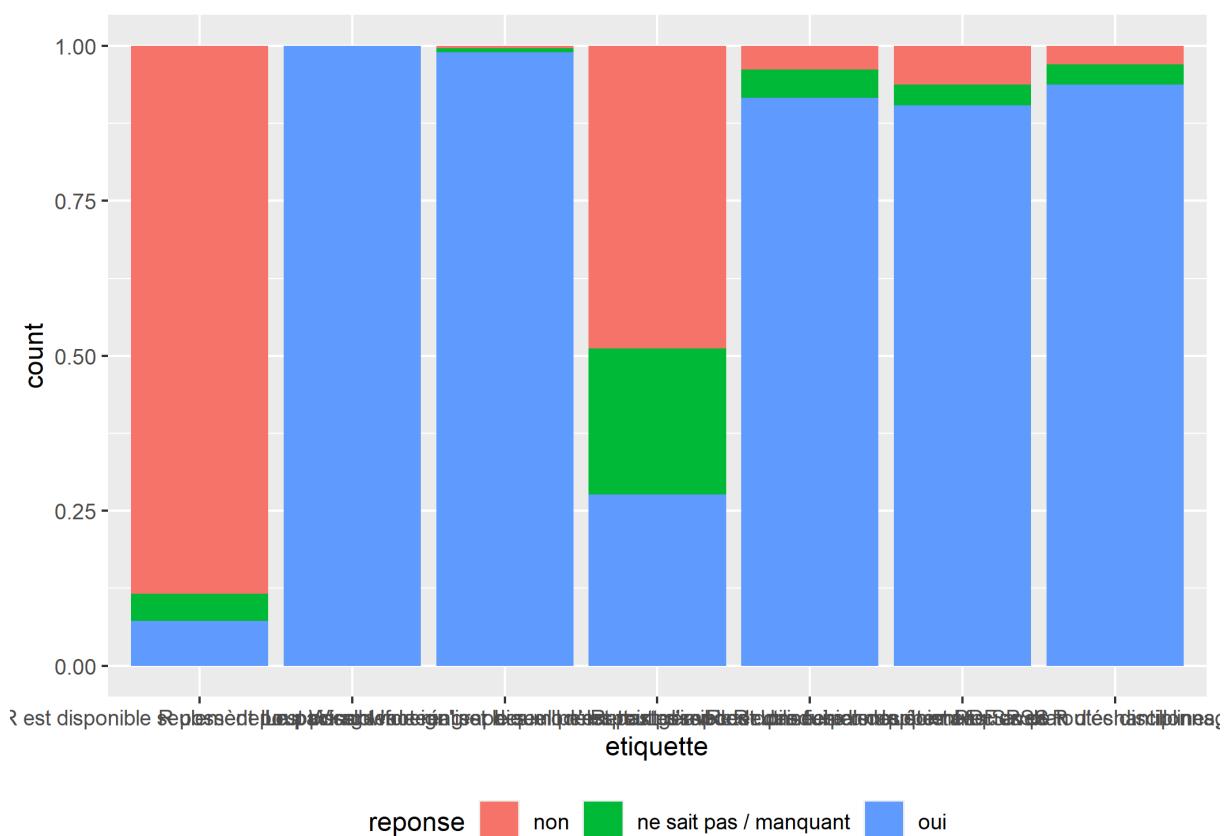
ggplot(conn) +
  aes(x = question, fill = reponse) +
  geom_bar(position = "fill") +
  theme(legend.position = "bottom")
```



Votre lectorat ne sait probablement pas à quoi correspondent les variables `conn_a` à `conn_g`. Il est donc préférable de les remplacer par des étiquettes plus explicites. Souvent, on a tendance à vouloir mettre des étiquettes courtes, quitte à reformuler le questionnaire d'origine. Ceci dit, il est pourtant préférable d'utiliser, quand cela est possible et pertinent, la formulation exacte du questionnaire. Ici nous allons créer une nouvelle variable `etiquette` et nous allons mettre à jour la définition de l'axe des x dans `aes`.

```
R> conn$etiquette <- conn$question %>%
  fct_recode(
    "R est disponible seulement pour Windows" = "conn_a",
    "R possède un puissant moteur graphique" = "conn_b",
    "Il est possible de réaliser des modèles mixtes avec R" = "conn_c",
    "Le package 'foreign' est le seul permettant d'importer des fichiers de données SPSS" = "conn_d",
    "Il n'est pas possible de produire un rapport PDF avec R" = "conn_e",
    "R peut gérer des données d'enquêtes avec un plan d'échantillonnage complexe" = "conn_f",
    "R est utilisée par des scientifiques de toutes disciplines, y compris des sciences sociales" = "conn_g"
  )

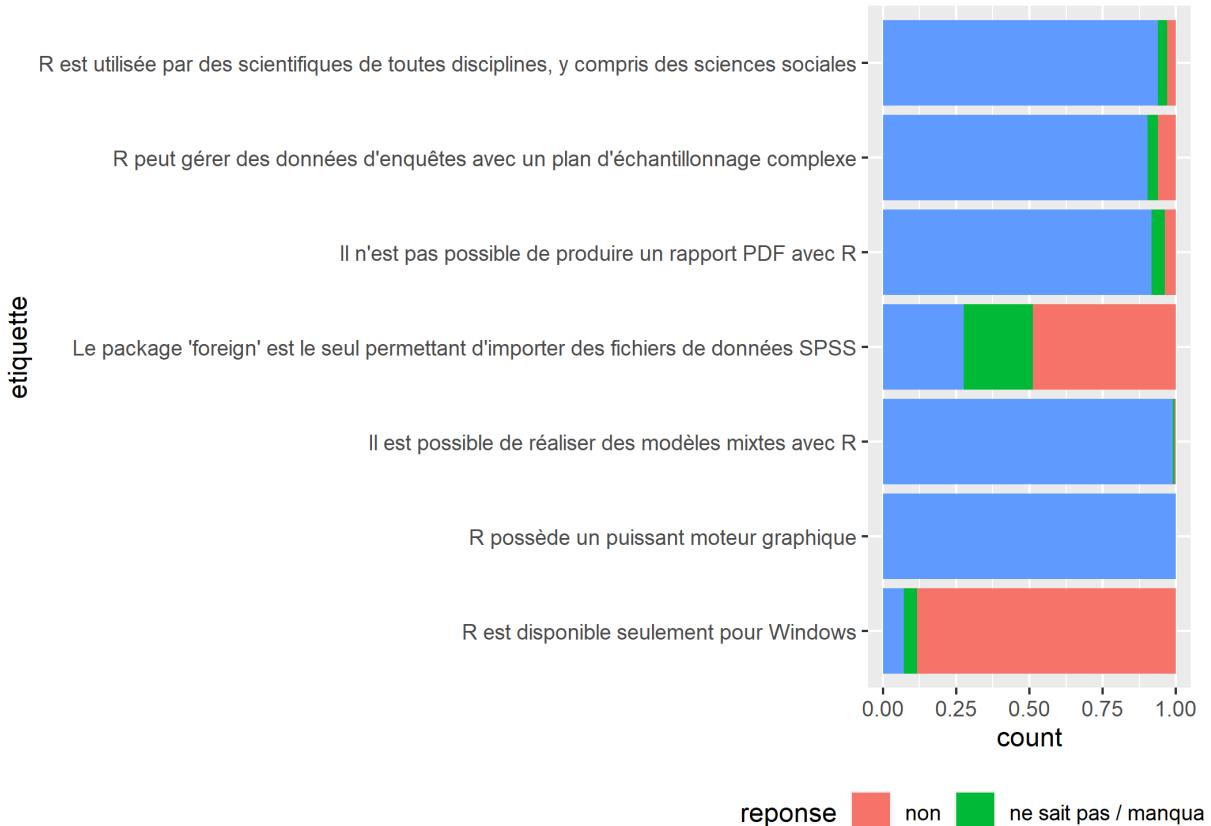
ggplot(conn) +
  aes(x = etiquette, fill = reponse) +
  geom_bar(position = "fill") +
  theme(legend.position = "bottom")
```



Malheureusement, avec un graphique en barres verticales, les étiquettes de l'axe des X sont tout

bonnement illisibles. Mais nous pouvons facilement transformer notre graphique en barres horizontales avec `coord_flip`. Une autre solution consiste à appliquer une rotation de 90 degrés aux étiquettes de l'axe des x, mais cette approche est moins lisible que le passage à des barres horizontales.

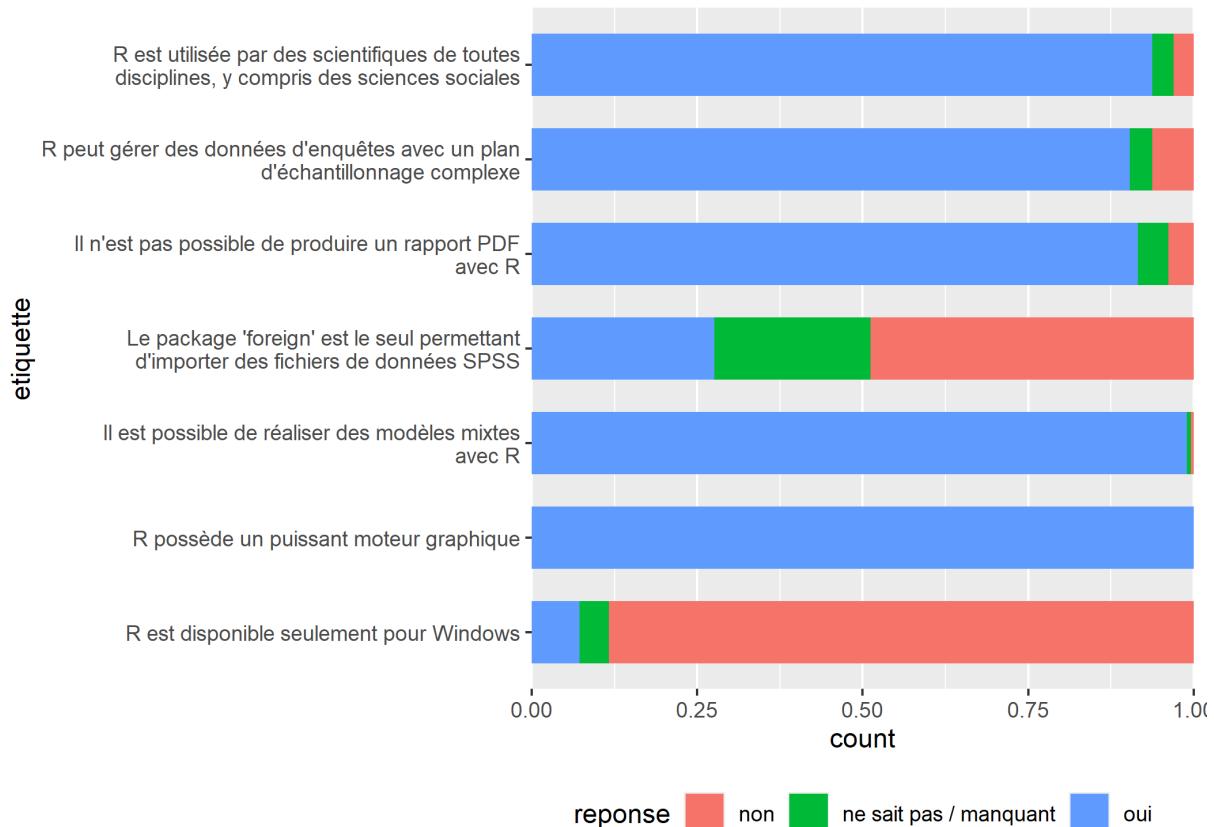
```
R> ggplot(conn) +
  aes(x = etiquette, fill = reponse) +
  geom_bar(position = "fill") +
  coord_flip() +
  theme(legend.position = "bottom")
```



C'est déjà mieux. Cependant, comme certaines étiquettes sont très longues, l'espace restant pour le graphique est réduit. Nous allons donc afficher ces étiquettes trop longues sur plusieurs lignes, grâce à la fonction `label_wrap` de l'extension `scales` que nous allons appeler à l'intérieur de `scale_x_discrete`. Le nombre passé à `label_wrap` indique le nombre de caractères à afficher avant retour à la ligne.

Nous allons également faire deux petites améliorations à notre graphique : (i) nous allons réduire l'épaisseur des barres en ajoutant `width = .66` à `geom_bar` ; (ii) nous allons éviter que l'axe des y (devenu l'axe horizontal) se soit étendu grâce à la commande `scale_y_continuous(expand = c(0, 0))`.

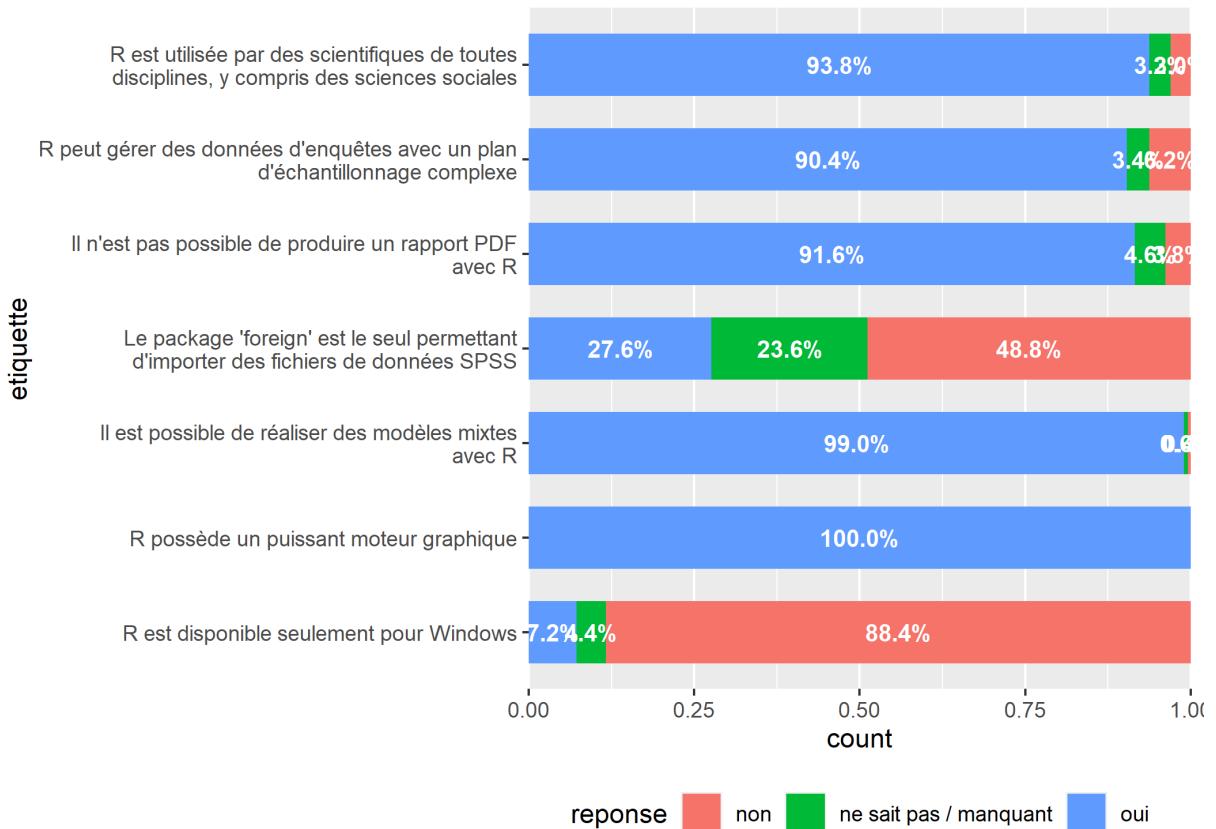
```
R> ggplot(conn) +
  aes(x = etiquette, fill = reponse) +
  geom_bar(position = "fill", width = .66) +
  scale_x_discrete(labels = scales::label_wrap(50)) +
  scale_y_continuous(expand = c(0, 0)) +
  coord_flip() +
  theme(legend.position = "bottom")
```



Pour améliorer la lisibilité du graphique, nous allons ajouter des étiquettes avec les pourcentages calculés (voir le chapitre sur les graphiques bivariés, page 450). Pour cela, nous aurons besoin de l'extension **GGally** qui fournie la statistique `stat_prop`.

Nous allons donc appeler `geom_text` avec cette statistique. Dans l'appel à `aes`, nous devons ajouter `by = etiquette` pour indiquer que nous voulons que nos pourcentages soit calculés pour chaque valeur de la variable `etiquette`. Dans l'appel à `geom_text`, nous allons préciser `position = position_fill(.5)` pour que nos étiquettes soit positionnées au milieu des rectangles. `colour = "white"` permet de préciser la couleur des étiquettes, `fontface = "bold"` pour les afficher en gras et `size = 3.5` pour contrôler leur taille.

```
R> library(GGally)
ggplot(conn) +
  aes(x = etiquette, fill = reponse, by = etiquette) +
  geom_bar(position = "fill", width = .66) +
  geom_text(
    stat = "prop", position = position_fill(.5),
    colour = "white", fontface = "bold", size = 3.5
  ) +
  scale_x_discrete(labels = scales::label_wrap(50)) +
  scale_y_continuous(expand = c(0, 0)) +
  coord_flip() +
  theme(legend.position = "bottom")
```



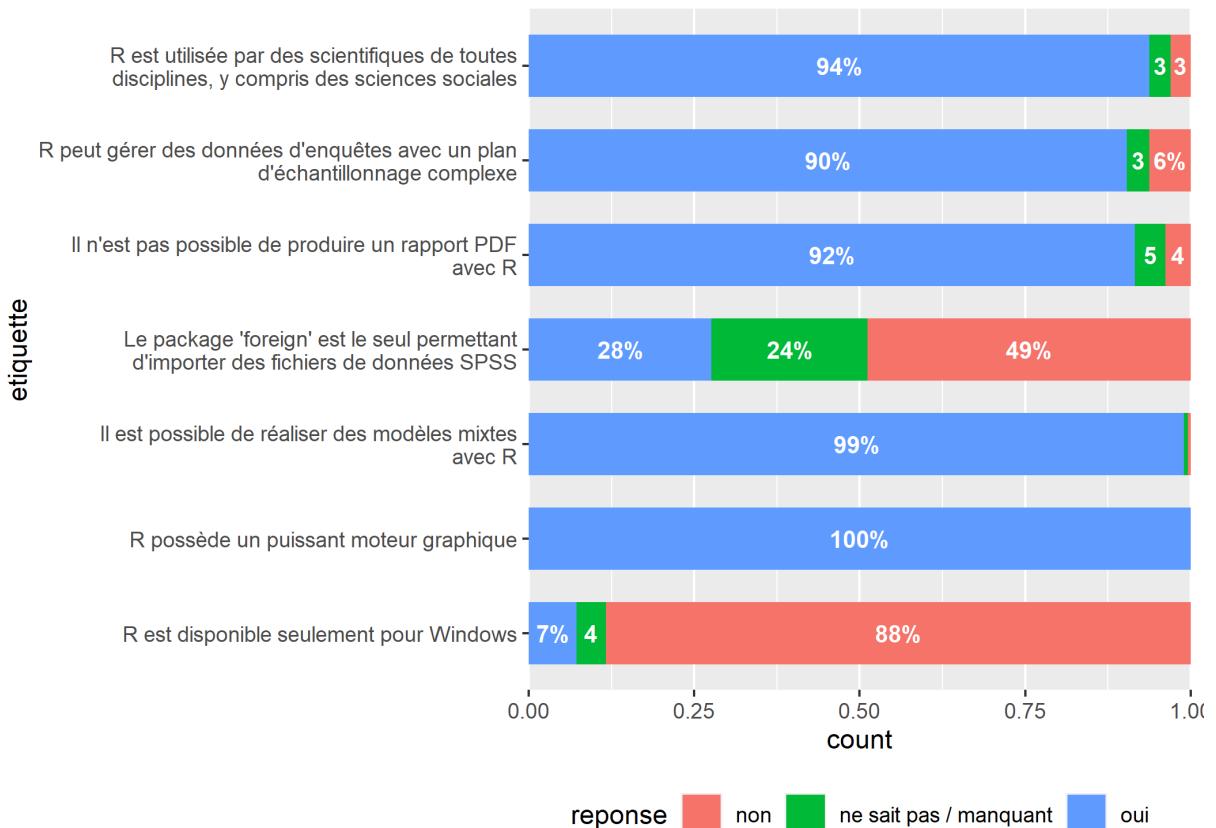
Les résultat est encourageant mais certaines étiquettes sont pas ou peu lisibles. Tout d'abord, nous n'allons pas afficher de décimale car c'est une précision inutile. Pour cela, nous allons utiliser la fonction `percent` de `scales` qui permet de mettre en forme des pourcentages. Nous allons préciser `accuracy = 1` pour indiquer que nous souhaitons arrondir à l'unité (pour une précision de deux décimales, nous aurions donc indiqué `accuracy = .01`).

De plus, pour les valeurs inférieures à 5% nous allons masquer le symbole % et pour les valeurs inférieures à 1% nous n'allons rien afficher. L'astuce consiste à créer une petite fonction personnalisée, que nous

allons appeler `f` et qui va s’occuper de la mise en forme. Puis, dans `aes`, l’esthétique `label` sera définie comme égale à `f(after_stat(prop))` (note : `after_stat` permet d’appeler la variable `prop` calculée par `stat_prop`).

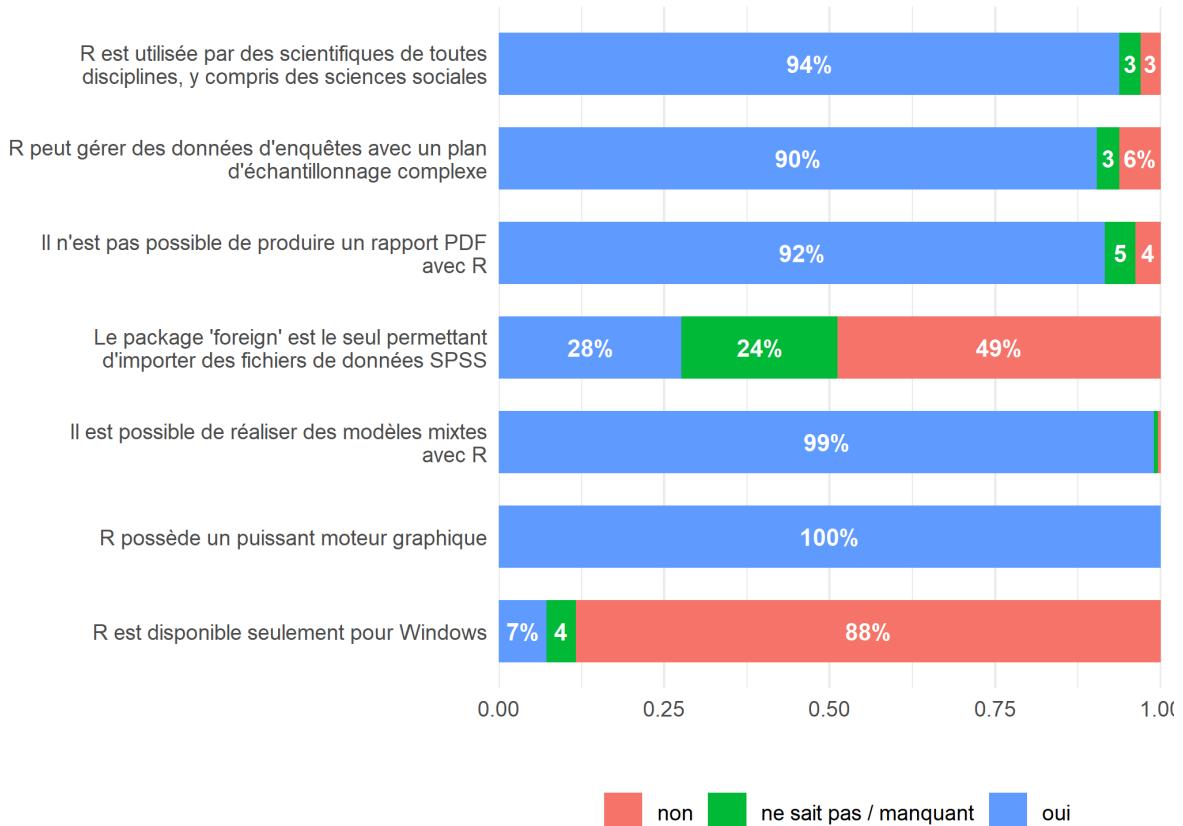
```
R> f <- function(x) {
  res <- scales::percent(x, accuracy = 1)
  res[x < .05] <- scales::percent(x[x < .05], accuracy = 1, suffix = "")
  res[x < .01] <- ""
  res
}

ggplot(conn) +
  aes(
    x = etiquette, fill = reponse,
    by = etiquette, label = f(after_stat(prop)))
  ) +
  geom_bar(position = "fill", width = .66) +
  geom_text(
    stat = "prop", position = position_fill(.5),
    colour = "white", fontface = "bold", size = 3.5
  ) +
  scale_x_discrete(labels = scales::label_wrap(50)) +
  scale_y_continuous(expand = c(0, 0)) +
  coord_flip() +
  theme(legend.position = "bottom")
```



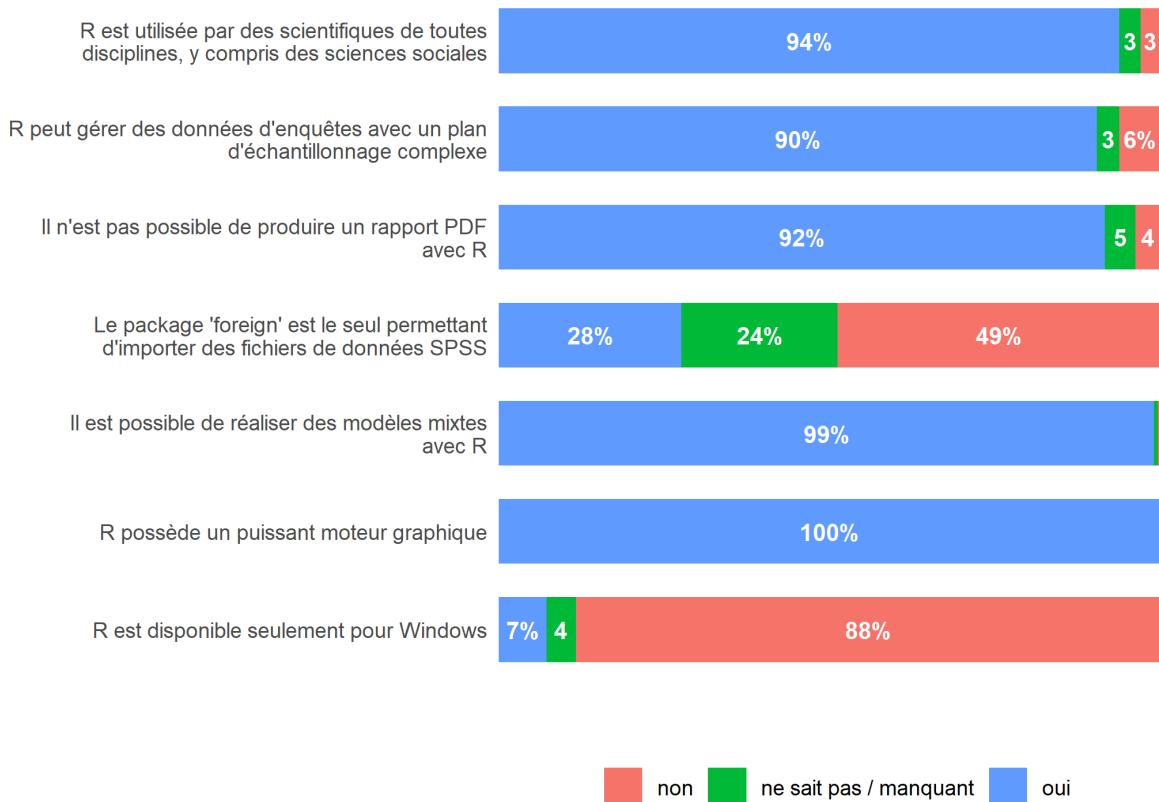
Nous commençons à avoir tous les éléments de notre graphique. Il est temps de faire un peu de nettoyage. Appliquons déjà `theme_minimal` pour alléger le graphique et supprimer les titres des axes et de la légende avec `labs`.

```
R> ggplot(conn) +
  aes(
    x = etiquette, fill = reponse,
    by = etiquette, label = f(after_stat(prop)))
  ) +
  geom_bar(position = "fill", width = .66) +
  geom_text(
    stat = "prop", position = position_fill(.5),
    colour = "white", fontface = "bold", size = 3.5
  ) +
  scale_x_discrete(labels = scales::label_wrap(50)) +
  scale_y_continuous(expand = c(0, 0)) +
  coord_flip() +
  labs(x = "", y = "", fill = "") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



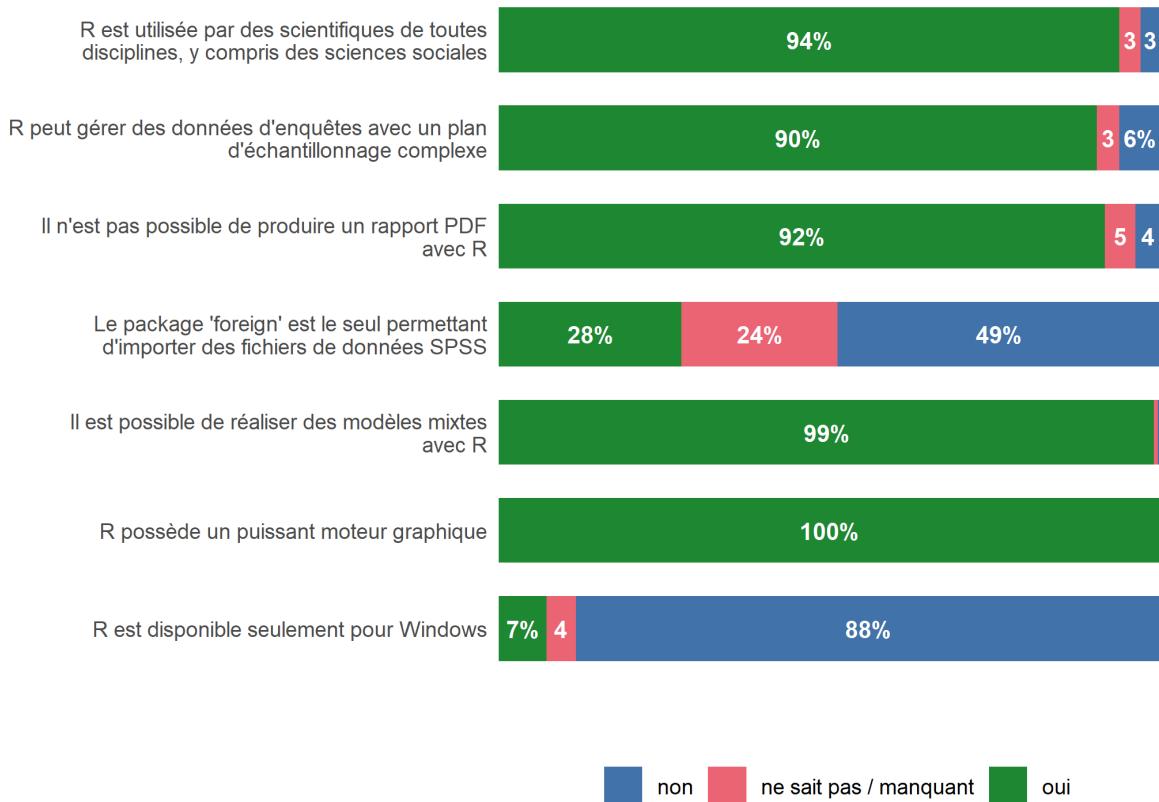
Comme les valeurs sont directement affichées sur le graphique, il est encore possible de l'alléger en supprimant la grille (avec `panel.grid = element_blank()`) et les étiquettes de l'axe horizontal (avec `axis.text.x = element_blank()`) via la fonction `theme`.

```
R> ggplot(conn) +
  aes(
    x = etiquette, fill = reponse,
    by = etiquette, label = f(after_stat(prop)))
  ) +
  geom_bar(position = "fill", width = .66) +
  geom_text(
    stat = "prop", position = position_fill(.5),
    colour = "white", fontface = "bold", size = 3.5
  ) +
  scale_x_discrete(labels = scales::label_wrap(50)) +
  scale_y_continuous(expand = c(0, 0)) +
  coord_flip() +
  labs(x = "", y = "", fill = "") +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    panel.grid = element_blank(),
    axis.text.x = element_blank()
  )
)
```



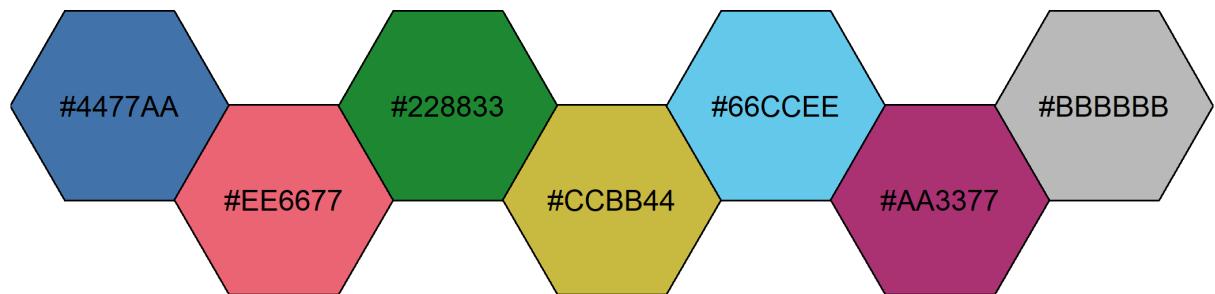
Nous allons également personnaliser la palette de couleur pour adopter une palette adaptée aux personnes daltoniennes. Il en existe plusieurs, dont [celles développées par Paul Tol](#) et disponibles dans l'extension **khroma** (voir aussi le chapitre sur les palettes de couleurs, page 1021). On peut appliquer cette palette avec `scale_fill_bright`.

```
R> library(khroma)
R> ggplot(conn) +
  aes(
    x = etiquette, fill = reponse,
    by = etiquette, label = f(after_stat(prop)))
  ) +
  geom_bar(position = "fill", width = .66) +
  geom_text(
    stat = "prop", position = position_fill(.5),
    colour = "white", fontface = "bold", size = 3.5
  ) +
  scale_x_discrete(labels = scales::label_wrap(50)) +
  scale_y_continuous(expand = c(0, 0)) +
  scale_fill_bright() +
  coord_flip() +
  labs(x = "", y = "", fill = "") +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    panel.grid = element_blank(),
    axis.text.x = element_blank()
  )
```



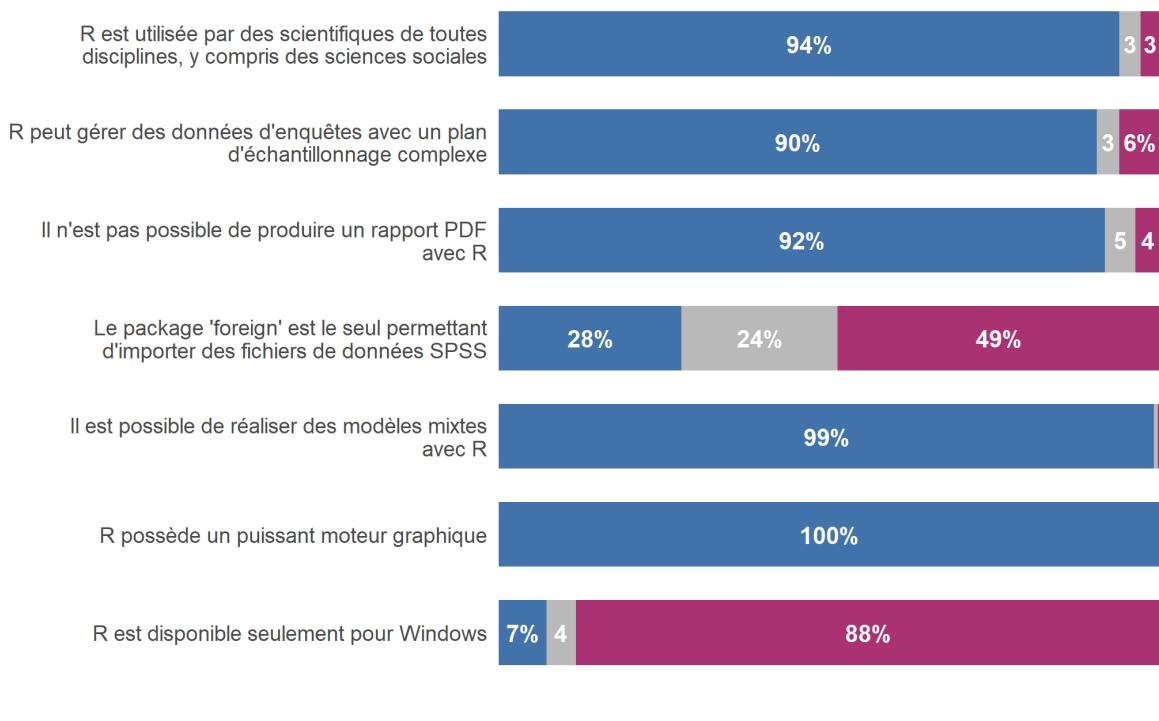
Cependant, ce choix de couleur n'est peut-être pas optimal. Une couleur neutre (proche du gris) serait peut être plus appropriée pour les "ne sait pas / manquant". Regardons les codes couleurs de la palette *bright*.

```
R> plot_scheme(colour("bright")(7), colours = TRUE)
```



Nous allons donc choisir des couleurs plus pertinentes et les définir manuellement avec `scale_fill_manual`.

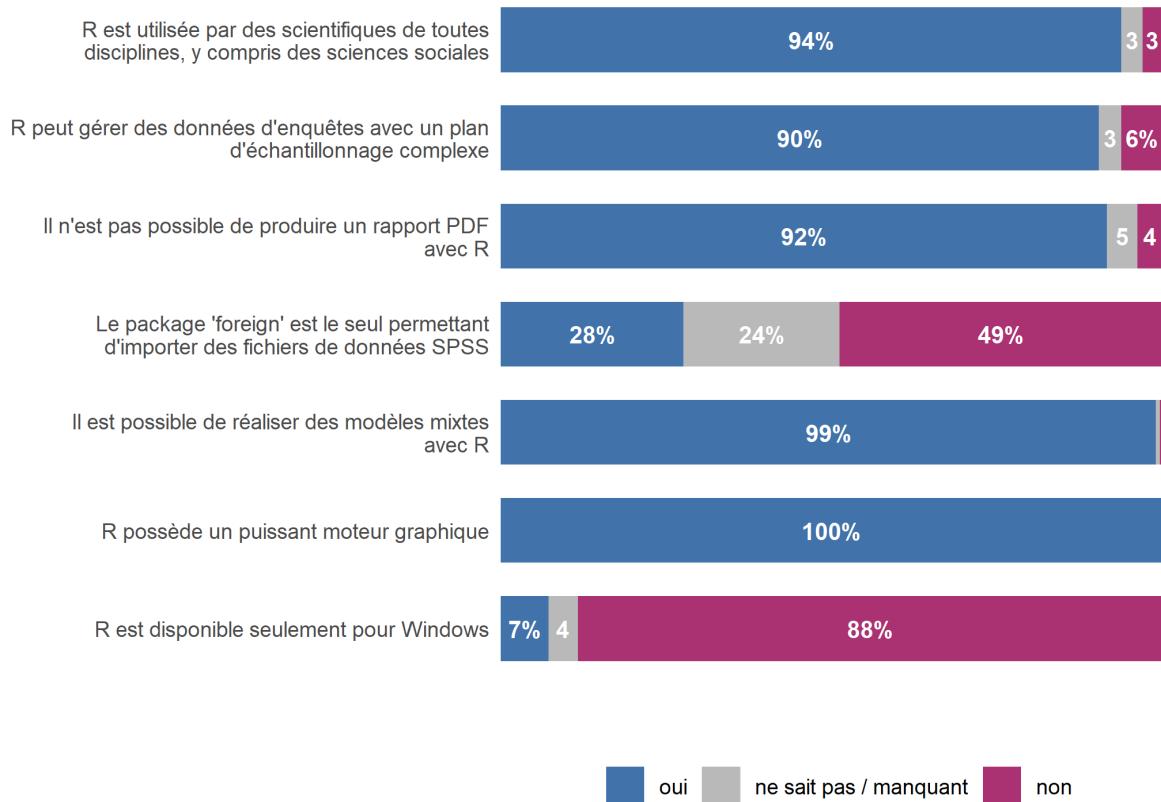
```
R> ggplot(conn) +
  aes(
    x = etiquette, fill = reponse,
    by = etiquette, label = f(after_stat(prop)))
  ) +
  geom_bar(position = "fill", width = .66) +
  geom_text(
    stat = "prop", position = position_fill(.5),
    colour = "white", fontface = "bold", size = 3.5
  ) +
  scale_x_discrete(labels = scales::label_wrap(50)) +
  scale_y_continuous(expand = c(0, 0)) +
  scale_fill_manual(values = c("#AA3377", "#BBBBBB", "#4477AA")) +
  coord_flip() +
  labs(x = "", y = "", fill = "") +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    panel.grid = element_blank(),
    axis.text.x = element_blank()
  )
)
```



[Marqueur] non [Marqueur] ne sait pas / manquant [Marqueur] oui

Pour faciliter la lecture, il serait pertinent que la légende soit dans le même ordre que le graphique (i.e. “oui” à gauche et “non” à droite). Nous allons donc inverser l’ordre de la légende en passant `fill = guide_legend(reverse = TRUE)` à `guides`.

```
R> ggplot(conn) +
  aes(
    x = etiquette, fill = reponse,
    by = etiquette, label = f(after_stat(prop)))
  ) +
  geom_bar(position = "fill", width = .66) +
  geom_text(
    stat = "prop", position = position_fill(.5),
    colour = "white", fontface = "bold", size = 3.5
  ) +
  scale_x_discrete(labels = scales::label_wrap(50)) +
  scale_y_continuous(expand = c(0, 0)) +
  scale_fill_manual(values = c("#AA3377", "#BBBBBB", "#4477AA")) +
  coord_flip() +
  labs(x = "", y = "", fill = "") +
  guides(fill = guide_legend(reverse = TRUE)) +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    panel.grid = element_blank(),
    axis.text.x = element_blank()
  )
```

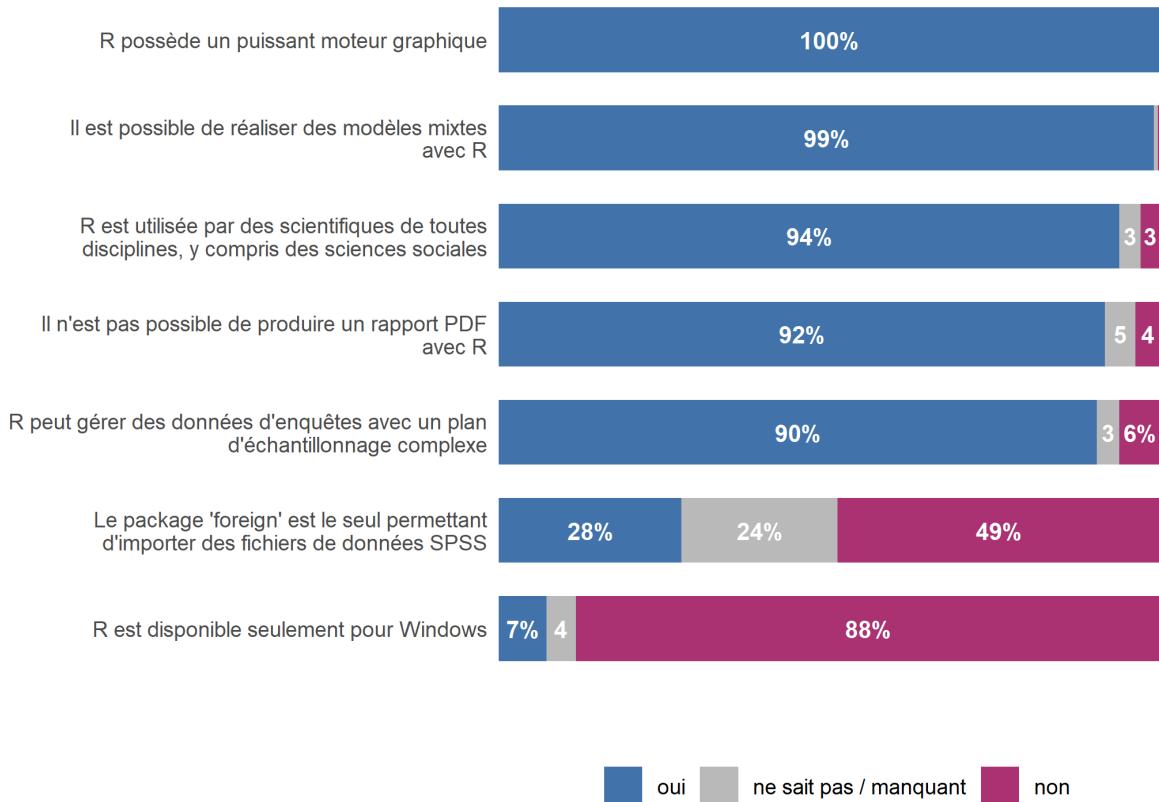


█ oui █ ne sait pas / manquant █ non

Il n'est pas évident de repérer ici au premier coup d'oeil quelle est la question qui a eu le plus de "oui". Nous allons donc ordonner les questions en fonction du pourcentage de "oui" grâce à la fonction `fct_reorder`.

```
R> conn$etiquette <- conn$etiquette %>%
  fct_reorder(conn$reponse == "oui", .fun = "sum")

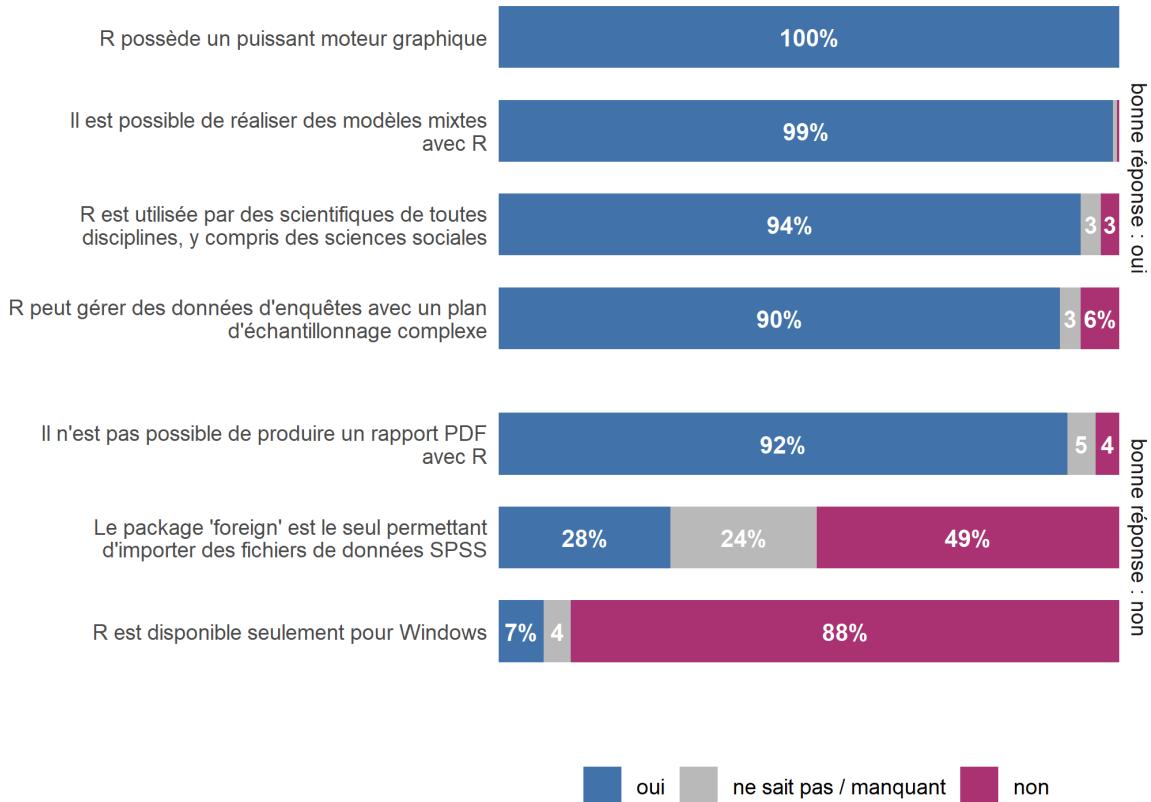
ggplot(conn) +
  aes(
    x = etiquette, fill = reponse,
    by = etiquette, label = f(after_stat(prop)))
  ) +
  geom_bar(position = "fill", width = .66) +
  geom_text(
    stat = "prop", position = position_fill(.5),
    colour = "white", fontface = "bold", size = 3.5
  ) +
  scale_x_discrete(labels = scales::label_wrap(50)) +
  scale_y_continuous(expand = c(0, 0)) +
  scale_fill_manual(values = c("#AA3377", "#BBBBBB", "#4477AA")) +
  coord_flip() +
  labs(x = "", y = "", fill = "") +
  guides(fill = guide_legend(reverse = TRUE)) +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    panel.grid = element_blank(),
    axis.text.x = element_blank()
  )
```



Toutes ces affirmations ne sont pas justes. Il serait donc pertinent de distinguer les affirmations justes (pour lesquelles la bonne réponse est "oui" des autres). Nous allons donc créer une nouvelle variable pour séparer ensuite les réponses avec `facet_grid` (attention : il est important de préciser `scales = "free"`, `space = "free"` ).

```
R> conn$correcte <- conn$question %>%
  fct_recode(
    "bonne réponse : non" = "conn_a",
    "bonne réponse : oui" = "conn_b",
    "bonne réponse : oui" = "conn_c",
    "bonne réponse : non" = "conn_d",
    "bonne réponse : non" = "conn_e",
    "bonne réponse : oui" = "conn_f",
    "bonne réponse : oui" = "conn_g"
  ) %>%
  fct_relevel("bonne réponse : oui")

ggplot(conn) +
  aes(
    x = etiquette, fill = reponse,
    by = etiquette, label = f(after_stat(prop))
  ) +
  geom_bar(position = "fill", width = .66) +
  geom_text(
    stat = "prop", position = position_fill(.5),
    colour = "white", fontface = "bold", size = 3.5
  ) +
  scale_x_discrete(labels = scales::label_wrap(50)) +
  scale_y_continuous(expand = c(0, 0)) +
  scale_fill_manual(values = c("#AA3377", "#BBBBBB", "#4477AA")) +
  coord_flip() +
  labs(x = "", y = "", fill = "") +
  guides(fill = guide_legend(reverse = TRUE)) +
  facet_grid(rows = vars(correcte), scales = "free", space = "free") +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    panel.grid = element_blank(),
    axis.text.x = element_blank()
  )
```



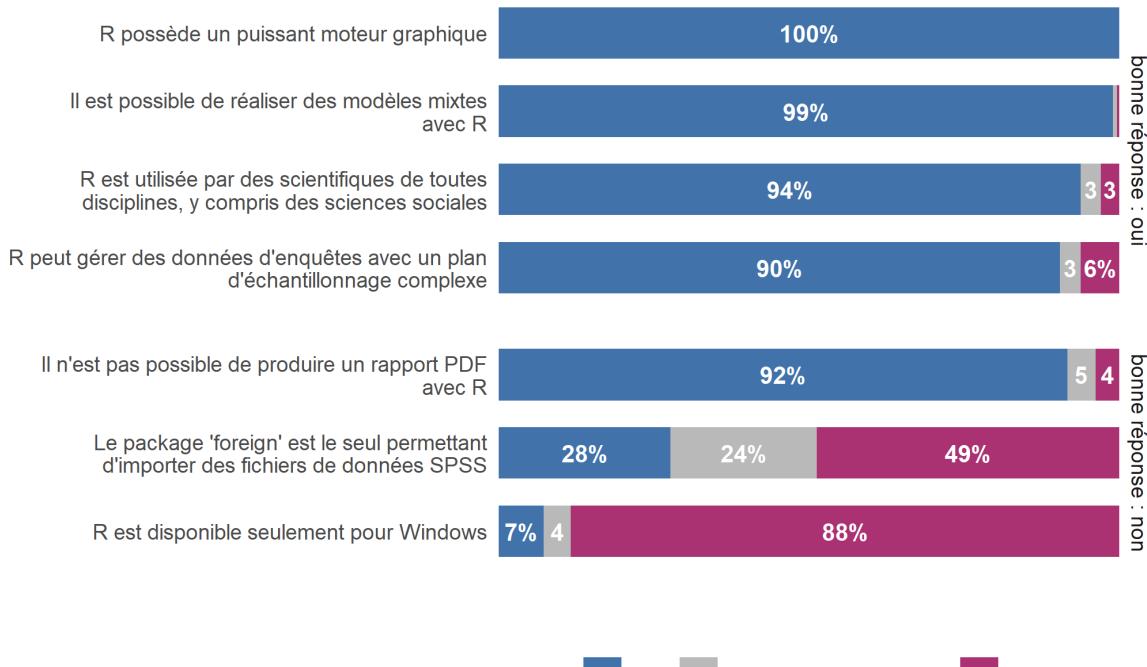
Il est maintenant bien visible que les étudiants ont en général bien répondu aux affirmations, mais qu'ils ont une connaissance erronée concernant la possibilité de réaliser des rapports automatisés en PDF avec R.

Ajoutons maintenant un titre et un sous-titre avec `ggtitle` et une note avec l'option `caption` de `labs`.

```
R> ggplot(conn) +
  aes(
    x = etiquette, fill = reponse,
    by = etiquette, label = f(after_stat(prop)))
  ) +
  geom_bar(position = "fill", width = .66) +
  geom_text(
    stat = "prop", position = position_fill(.5),
    colour = "white", fontface = "bold", size = 3.5
  ) +
  scale_x_discrete(labels = scales::label_wrap(50)) +
  scale_y_continuous(expand = c(0, 0)) +
  scale_fill_manual(values = c("#AA3377", "#BBBBBB", "#4477AA")) +
  coord_flip() +
  labs(
    x = "", y = "", fill = "",
    caption = "Enquête réalisée auprès de 500 étudiants"
  ) +
  ggtitle(
    "CONNAISSANCES SUR R",
    subtitle = "Pour chacune de ces affirmations, diriez-vous qu'elle est correcte ?"
  ) +
  guides(fill = guide_legend(reverse = TRUE)) +
  facet_grid(rows = vars(correcte), scales = "free", space = "free") +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    panel.grid = element_blank(),
    axis.text.x = element_blank()
  )
```

## CONNAISSANCES SUR R

Pour chacune de ces affirmations, diriez-vous qu'elle est



Legend:  oui  ne sait pas / manquant  non

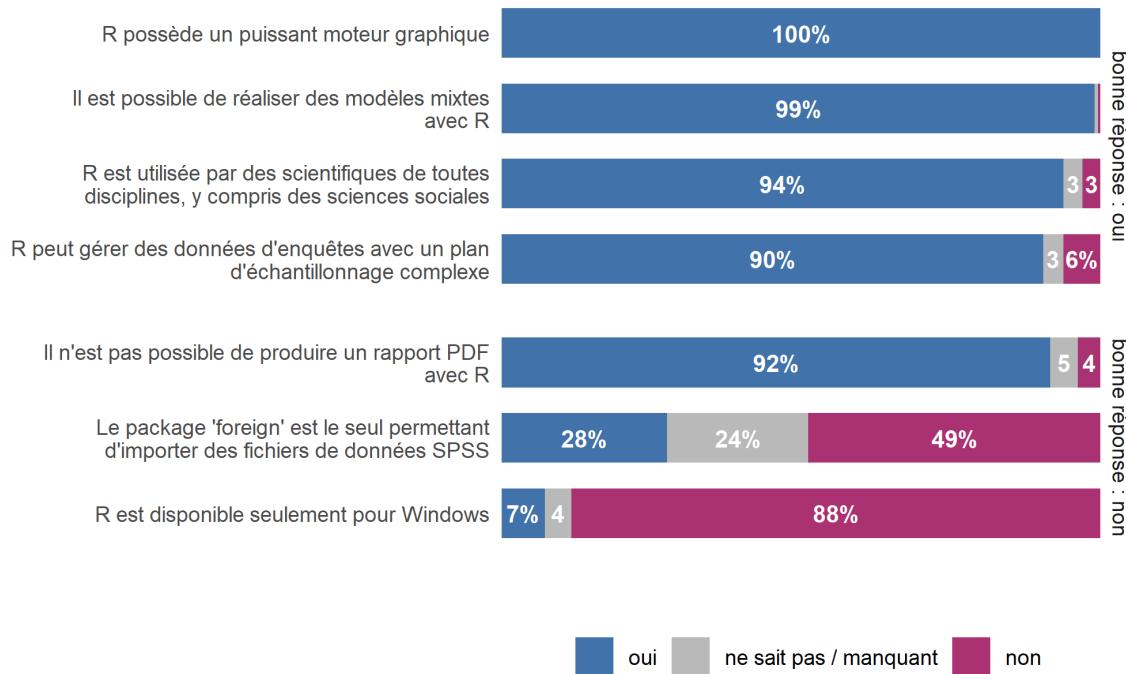
Enquête réalisée auprès de 500 étudiants

Nous pouvons procéder à quelques derniers ajustements (position du titre et de la note, note en italique, marges du graphique) en ajoutant quelques arguments additionnels à `theme`.

```
R> ggplot(conn) +
  aes(
    x = etiquette, fill = reponse,
    by = etiquette, label = f(after_stat(prop)))
  ) +
  geom_bar(position = "fill", width = .66) +
  geom_text(
    stat = "prop", position = position_fill(.5),
    colour = "white", fontface = "bold", size = 3.5
  ) +
  scale_x_discrete(labels = scales::label_wrap(50)) +
  scale_y_continuous(expand = c(0, 0)) +
  scale_fill_manual(values = c("#AA3377", "#BBBBBB", "#4477AA")) +
  coord_flip() +
  labs(
    x = "", y = "", fill = "",
    caption = "Enquête réalisée auprès de 500 étudiants"
  ) +
  ggtitle(
    "CONNAISSANCES SUR R",
    subtitle = "Pour chacune de ces affirmations, diriez-vous qu'elle est correcte ?"
  ) +
  guides(fill = guide_legend(reverse = TRUE)) +
  facet_grid(rows = vars(correcte), scales = "free", space = "free") +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    panel.grid = element_blank(),
    axis.text.x = element_blank(),
    plot.title.position = "plot",
    plot.caption.position = "plot",
    plot.caption = element_text(face = "italic", hjust = 0),
    plot.margin = margin(10, 10, 10, 10)
  )
```

## CONNAISSANCES SUR R

Pour chacune de ces affirmations, diriez-vous qu'elle est correcte ?



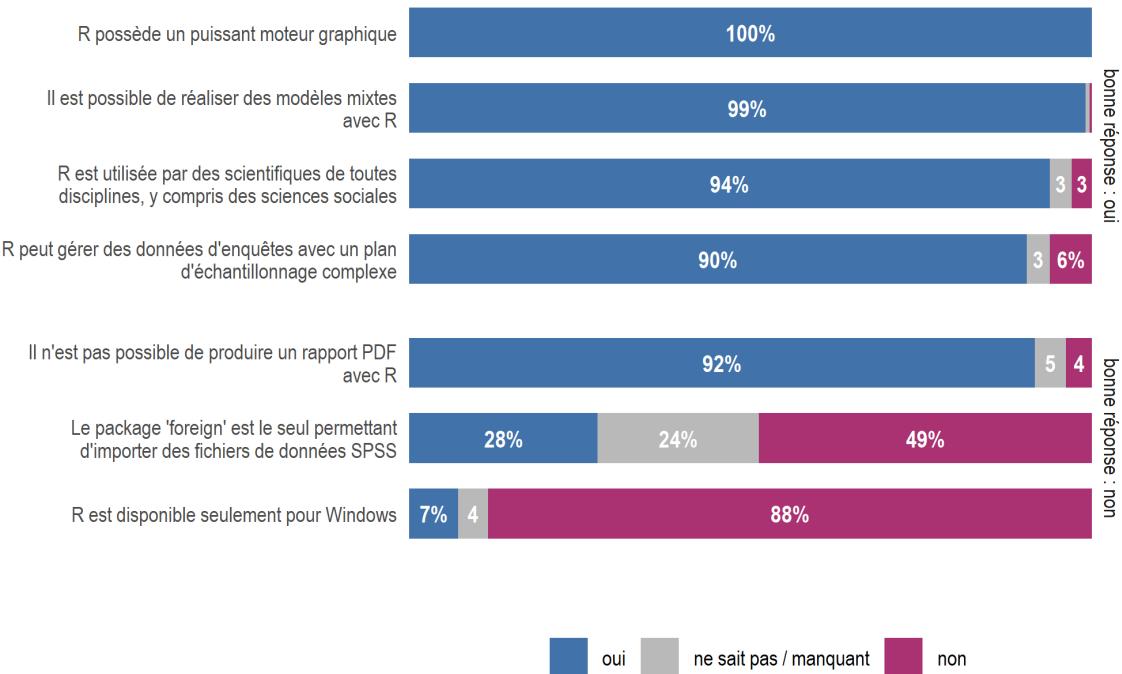
Enquête réalisée auprès de 500 étudiants

Enfin, pour un rendu un peu plus moderne, nous allons opter pour une autre police de caractères, ici "Arial Narrow". Afin de pouvoir utiliser des polices systèmes, nous aurons besoin de l'extension `extrafont`. La police doit être précisée à la fois dans `theme_minimal` et dans `geom_text`.

```
R> library(extrafont)
ggplot(conn) +
  aes(
    x = etiquette, fill = reponse,
    by = etiquette, label = f(after_stat(prop)))
  ) +
  geom_bar(position = "fill", width = .66) +
  geom_text(
    stat = "prop", position = position_fill(.5),
    colour = "white", fontface = "bold", size = 3.5,
    family = "Arial Narrow"
  ) +
  scale_x_discrete(labels = scales::label_wrap(50)) +
  scale_y_continuous(expand = c(0, 0)) +
  scale_fill_manual(values = c("#AA3377", "#BBBBBB", "#4477AA")) +
  coord_flip() +
  labs(
    x = "", y = "", fill = "",
    caption = "Enquête réalisée auprès de 500 étudiants"
  ) +
  ggtitle(
    "CONNAISSANCES SUR R",
    subtitle = "Pour chacune de ces affirmations, diriez-vous qu'elle est correcte ?"
  ) +
  guides(fill = guide_legend(reverse = TRUE)) +
  facet_grid(rows = vars(correcte), scales = "free", space = "free") +
  theme_minimal(base_family = "Arial Narrow") +
  theme(
    legend.position = "bottom",
    panel.grid = element_blank(),
    axis.text.x = element_blank(),
    plot.title.position = "plot",
    plot.caption.position = "plot",
    plot.caption = element_text(face = "italic", hjust = 0),
    plot.margin = margin(10, 10, 10, 10)
  )
)
```

## CONNAISSANCES SUR R

Pour chacune de ces affirmations, diriez-vous qu'elle est correcte ?



Enquête réalisée auprès de 500 étudiants

## Code final du graphique

Nous y voilà !

```
R> load(url("https://larmarange.github.io/analyse-R/data/connaissances.RData"))
```

```
R> library(tidyverse)
library(GGally)
library(extrafont)

conn <- quest %>%
  select(starts_with("conn_")) %>%
  pivot_longer(
    cols = starts_with("conn_"),
    names_to = "question",
    values_to = "reponse"
  )

conn$reponse <- conn$reponse %>%
  fct_explicit_na("NSP") %>%
  fct_relevel("non", "NSP", "oui") %>%
  fct_recode("ne sait pas / manquant" = "NSP")

conn$etiquette <- conn$question %>%
  fct_recode(
    "R est disponible seulement pour Windows" = "conn_a",
    "R possède un puissant moteur graphique" = "conn_b",
    "Il est possible de réaliser des modèles mixtes avec R" = "conn_c",
    "Le package 'foreign' est le seul permettant d'importer des fichiers de données SPSS" = "conn_d",
    "Il n'est pas possible de produire un rapport PDF avec R" = "conn_e",
    "R peut gérer des données d'enquêtes avec un plan d'échantillonnage complexe" = "conn_f",
    "R est utilisée par des scientifiques de toutes disciplines, y compris de sciences sociales" = "conn_g"
  ) %>%
  fct_reorder(conn$reponse == "oui", .fun = "sum")

conn$correcte <- conn$question %>%
  fct_recode(
    "bonne réponse : non" = "conn_a",
    "bonne réponse : oui" = "conn_b",
    "bonne réponse : oui" = "conn_c",
    "bonne réponse : non" = "conn_d",
    "bonne réponse : non" = "conn_e",
    "bonne réponse : oui" = "conn_f",
    "bonne réponse : oui" = "conn_g"
  ) %>%
  fct_relevel("bonne réponse : oui")

f <- function(x) {
  res <- scales::percent(x, accuracy = 1)
  res[x < .05] <- scales::percent(x[x < .05], accuracy = 1, suffix = "")
```

```

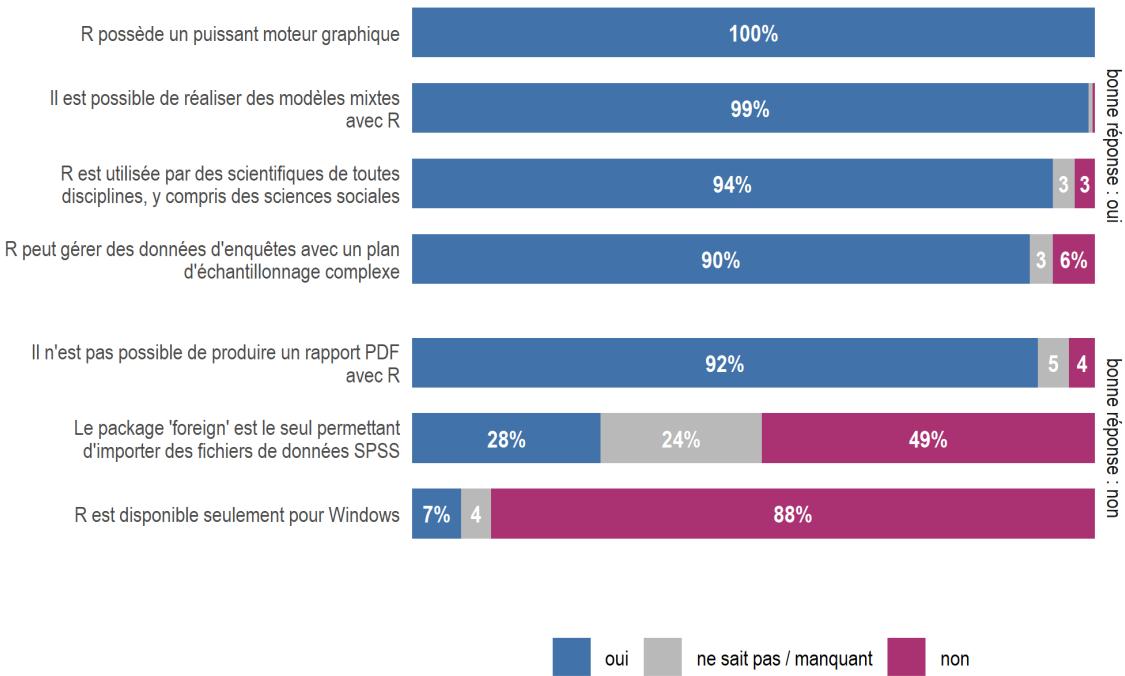
res[x < .01] <- ""
res
}

ggplot(conn) +
  aes(
    x = etiquette, fill = reponse,
    by = etiquette, label = f(after_stat(prop)))
  ) +
  geom_bar(position = "fill", width = .66) +
  geom_text(
    stat = "prop", position = position_fill(.5),
    colour = "white", fontface = "bold", size = 3.5,
    family = "Arial Narrow"
  ) +
  scale_x_discrete(labels = scales::label_wrap(50)) +
  scale_y_continuous(expand = c(0, 0)) +
  scale_fill_manual(values = c("#AA3377", "#BBBBBB", "#4477AA")) +
  coord_flip() +
  labs(
    x = "", y = "", fill = "",
    caption = "Enquête réalisée auprès de 500 étudiants"
  ) +
  ggtitle(
    "CONNAISSANCES SUR R",
    subtitle = "Pour chacune de ces affirmations, diriez-vous qu'elle est correcte ?"
  ) +
  guides(fill = guide_legend(reverse = TRUE)) +
  facet_grid(rows = vars(correcte), scales = "free", space = "free") +
  theme_minimal(base_family = "Arial Narrow") +
  theme(
    legend.position = "bottom",
    panel.grid = element_blank(),
    axis.text.x = element_blank(),
    plot.title.position = "plot",
    plot.caption.position = "plot",
    plot.caption = element_text(face = "italic", hjust = 0),
    plot.margin = margin(10, 10, 10, 10)
  )
)

```

## CONNAISSANCES SUR R

Pour chacune de ces affirmations, diriez-vous qu'elle est correcte ?



Enquête réalisée auprès de 500 étudiants

# Graphiques interactifs

ggplotly (plotly) . . . . .	875
ggvis . . . . .	878

## ggplotly (plotly)

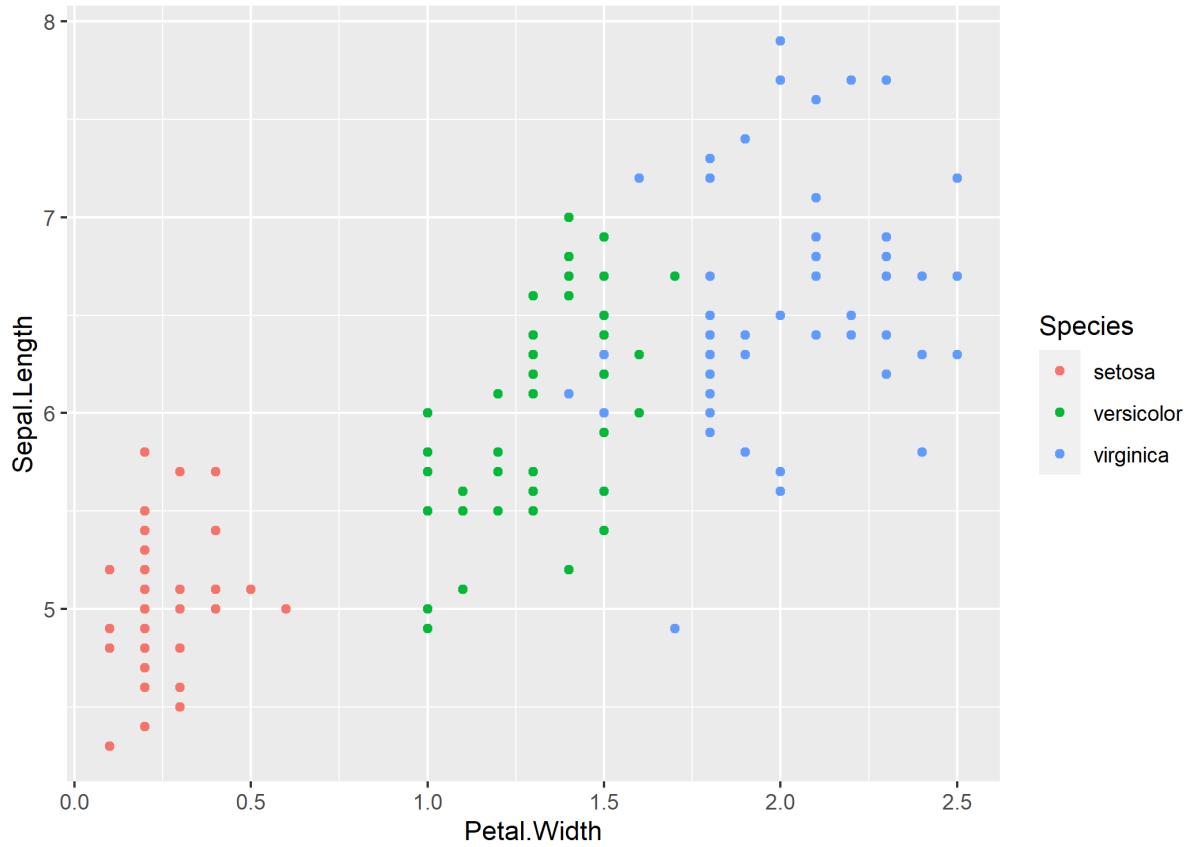
Vous savez réaliser des graphiques avec **ggplot2**, page 395 ? Vous savez faire un graphique interactif. Rien de plus facile avec la fonction `ggplotly` de l'extension **plotly**.

Créons un graphique.

```
R> library(ggplot2)
p <- ggplot(iris) +
  aes(x = Petal.Width, y = Sepal.Length, color = Species) +
  geom_point()
```

Voici son rendu classique avec **ggplot2**.

```
R> p
```



Et si on passe notre graphique à `ggplotly`. (N'hésitez pas à faire passer le curseur de votre souris sur le graphique.)

```
R> library(plotly)
```

```
Attaching package: 'plotly'
```

```
The following object is masked from 'package:MASS':
```

```
select
```

```
The following object is masked from 'package:Hmisc':
```

```
subplot
```

```
The following objects are masked from 'package:plyr':
```

```
arrange, mutate, rename, summarise
```

```
The following object is masked from 'package:ggplot2':
```

```
last_plot
```

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

```
filter
```

```
The following object is masked from 'package:graphics':
```

```
layout
```

```
R> ggplotly(p)
```

Une documentation complète sur [ggplotly](https://plot.ly/ggplot2/) est disponible sur <https://plot.ly/ggplot2/>.

## ggvis

Il existe également une extension [ggvis](https://ggvis.rstudio.com/) dédiée aux graphiques interactifs. Sa documentation complète est disponible sur <https://ggvis.rstudio.com/>.

# **lattice : graphiques et formules**

Les bases des graphiques <b>lattice</b> . . . . .	0 . 879
De l'intérêt des formules R . . . . .	880
Les formules R . . . . .	883
Principaux types de graphiques avec <code>lattice</code> (et <code>ggplot2</code> ) . . . . .	884
Histogramme . . . . .	885
Courbe de densité . . . . .	887
Diagramme en barres . . . . .	890
Diagramme de type boîtes à moustaches . . . . .	892
Diagramme en points . . . . .	893
Diagramme de dispersion . . . . .	893

Bien que l'on ait fait le choix de présenter principalement l'extension `ggplot2` plutôt que l'extension `lattice`, celle-ci reste un excellent choix pour la visualisation, notamment, de `panels` et de `séries temporelles`. On trouve de `très beaux exemples` d'utilisation de `lattice` en ligne, mais un peu moins de documentation, et beaucoup moins d'extensions, que pour `ggplot2`.

On peut trouver en ligne un support de cours détaillé (en anglais) de Deepayan Sarkar ([https://www.isid.ac.in/~deepayan/R-tutorials/labs/04\\_lattice\\_lab.pdf](https://www.isid.ac.in/~deepayan/R-tutorials/labs/04_lattice_lab.pdf)), également l'auteur de l'ouvrage *Lattice: Multivariate Data Visualization with R* (<http://lmdvr.r-forge.r-project.org/>).

## **Les bases des graphiques `lattice`**

R dispose de deux principaux systèmes graphiques : un système de base contrôlé par le package `graphics` et le système `grid`, sur lequel se basent à la fois les packages `lattice` et `ggplot2`. Ce système fournit les mêmes fonctionnalités de base que `graphics` mais offre une gestion de l'arrangement des objets graphiques plus développée, et surtout la possibilité d'utiliser ce que l'on appelle des *graphiques en treillis*. De plus, les graphiques peuvent être mis à jour très simplement, disposent de thèmes de couleur pré-définis, et offrent un certain degré d'interactivité, avec ou sans le package `plotly`. Enfin, la syntaxe est plus homogène et grandement simplifiée, grâce à l'usage de formules.

## De l’intérêt des formules R

Voici par exemple comment afficher la courbe de densité (i.e., la version continue et “lissée” d’un histogramme) de deux séries d’observations définies par les niveaux du facteur `supp` dans le data frame `ToothGrowth`, disponible dans les exemples de base de R. Notons que l’on souhaite également faire apparaître les distributions univariées, un peu à l’image de ce que fournit `rug`. Or cette fonction ne permet pas d’exploiter une variable de groupement, donc il sera nécessaire de gérer tout cela manuellement. Voici les instructions permettant de générer le graphique désiré :

```
R> plot(density(ToothGrowth$len[ToothGrowth$supp == "OJ"]),
  main = "", xlab = "len", las = 1, lwd = 2, col = "coral")
lines(density(ToothGrowth$len[ToothGrowth$supp == "VC"]),
  lwd = 2, col = "cornflowerblue")
points(
  x = ToothGrowth$len[ToothGrowth$supp == "OJ"],
  y = runif(length(ToothGrowth$len[ToothGrowth$supp == "OJ"]),
    min = -0.001, max = 0.001
  ), col = "coral"
)
points(
  x = ToothGrowth$len[ToothGrowth$supp == "VC"],
  y = runif(length(ToothGrowth$len[ToothGrowth$supp == "VC"]),
    min = -0.001, max = 0.001
  ), col = "cornflowerblue"
)
legend("top", levels(ToothGrowth$supp), col = c("coral", "cornflowerblue"),
  lty = 1, bty = "n")
```

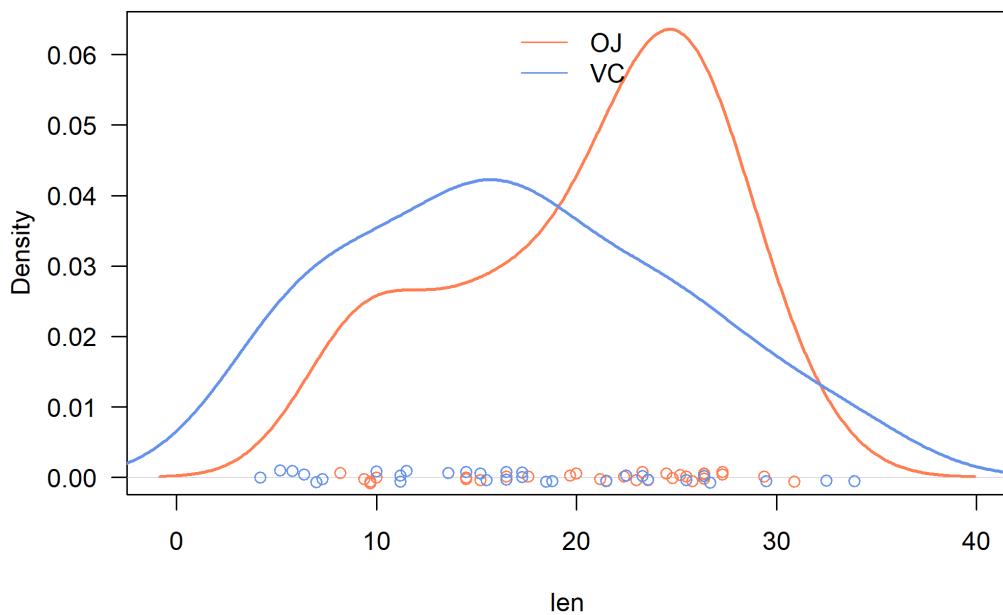


Figure 1. Courbes de densité avec les graphiques de base

Il y a plusieurs points à retenir dans les instructions ci-dessus : (1) il est nécessaire de définir les deux

courbes de densité et les deux distributions univariées, en prenant garde à bien indiquer comment sélectionner les observations ( OJ ou VC ) en préfixant systématiquement le nom des variables par le nom du data frame ; (2) la définition des couleurs se fait manuellement et si l'on souhaite changer de thème de couleur, il faudra mettre à jour l'ensemble des instructions, en prenant garde à ce que la légende reste synchronisée avec les courbes de densité et les nuages de points ; et, bien entendu, (3) il est nécessaire de gérer soi-même la légende, ce qui signifie se rappeler les couleurs et l'ordre des niveaux du facteur considéré, ainsi que les axes graphiques dans le cas où l'on souhaite les maintenir coordonnés sur plusieurs panneaux graphiques.

Voici le même graphique avec **lattice** :

```
R> library(lattice)
densityplot(~len, data = ToothGrowth, group = supp, auto.key = TRUE)
```

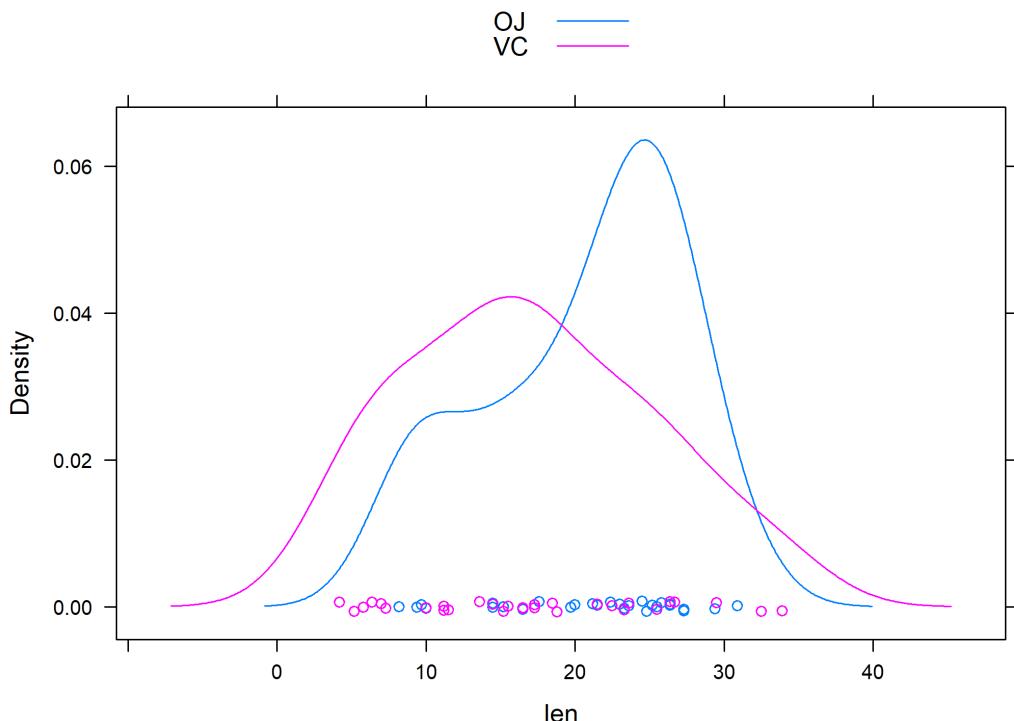


Figure 2. Courbes de densité avec le package **lattice**

Avec **ggplot2**, cela donnerait :

```
R> library(ggplot2)
ggplot(data = ToothGrowth, aes(x = len, color = supp)) +
  geom_line(stat = "density") +
  geom_rug()
```

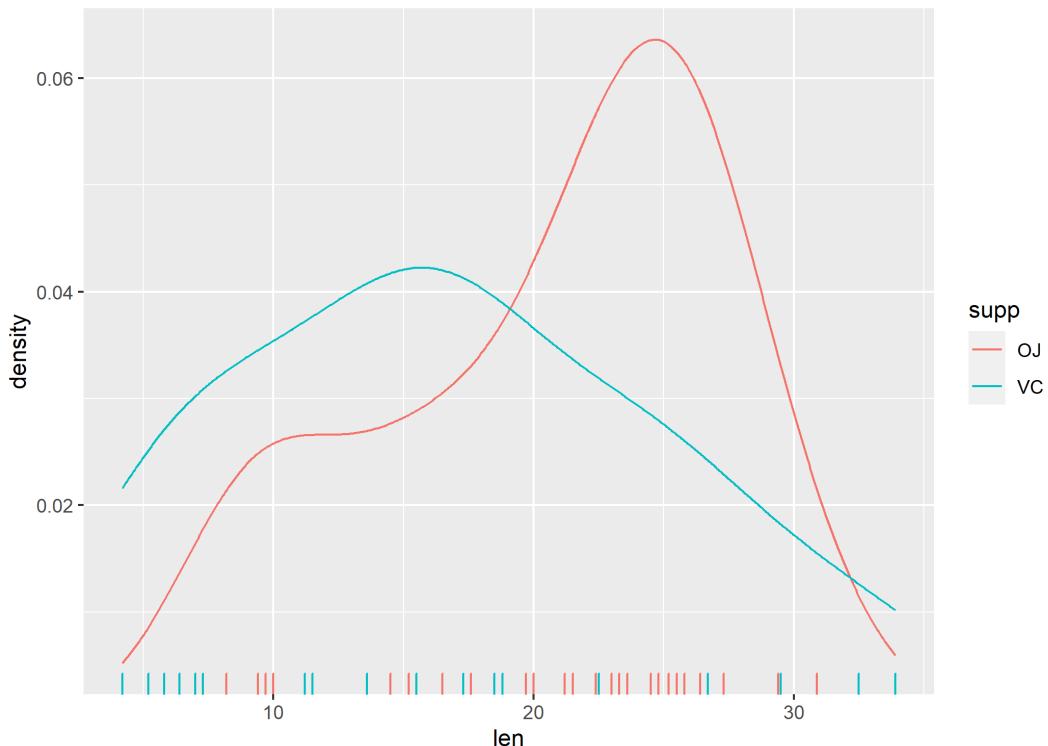


Figure 3. Courbes de densité avec le package `ggplot2`

Clairement, on gagne en nombre d'instructions à taper dans la console et en clarté d'expression également, grâce notamment à l'usage de formules permettant de décrire la relation entre chacune des variables utilisées pour construire la représentation graphique.

## Les formules R

Les formules utilisées dans le système `lattice` sont presque identiques à celles retrouvées dans les modèles d'analyse de variance (`aov`) ou de régression (`lm`). En réalité, la notation par formule qu'utilise R est celle proposée par Wilkinson et coll. dans les années 70 pour schématiser la relation entre plusieurs variables dans un plan d'expérience. Plus spécifiquement, l'idée revient à exprimer une relation «fonctionnelle», symbolisée par l'opérateur `~`, entre une variable réponse `y` et une ou plusieurs variables explicatives. Disons, pour simplifier, que `y` est une variable numérique, de même que `x`, et que

`a` et `b` sont des variables catégorielles (des facteurs dans le langage R). Voici les principales relations auxquelles on peut s'intéresser dans un modèle statistique linéaire :

- `y ~ x` : régression linéaire simple,
- `y ~ x + 0` : idem avec suppression du terme d'ordonnée à l'origine,
- `y ~ a + b` : ANOVA avec deux effets principaux,
- `y ~ a * b` : idem avec interaction (équivalent à `1 + a + b + a:b`),
- `y ~ a / b` : idem en considérant une relation d'emboîtement (équivalent à `1 + a + b + a %in% b`).

Un exemple typique d'utilisation pour un modèle d'ANOVA à trois facteurs est donné ci-dessous :

```
R> fm <- y ~ a * b * c # modèle de base (A, B, C, AB, AC, BC, ABC)
    mod1 <- aov(fm, data = dfrm) # estimation des paramètres du modèle
    update(mod1, . ~ . - a:b:c) # suppression de l'interaction ABC
```

Quant on y réfléchit un peu, les relations ci-dessus peuvent très bien s'appliquer au cas de la composition graphique : `y ~ x` signifie dans ce cas que l'on souhaite représenter l'évolution de `y` en fonction de `x`. En d'autres termes, on s'intéresse à un nuage de dispersion. Le package `lattice` ajoute les notations suivantes :

- `~ x` : dans le cas où l'on ne décrit qu'une seule variable (i.e., sa distribution),
- `a | b` : dans le cas où l'on considère la variable `a`, conditionnellement à la variable `b`, c'est-à-dire les niveaux de `a` pour chacun des niveaux de `b` (ce qui revient à l'interaction `a:b` citée ci-dessus).

Cette dernière notation se révèlera être très utile dans le cas des représentations graphiques conditionnelles, par exemple lorsque l'on souhaite afficher la distribution d'une variable numérique dans différents groupes d'individus définis par les niveaux d'une variable catégorielle, ou lorsque l'on souhaite surigner d'une couleur différentes les points d'un diagramme de dispersion selon la valeur prise par une troisième variable.

Les formules R sont omniprésentes dans les modèles statistiques, dans les graphiques, mais également dans certaines commandes d'agrégation. Au bout du compte, avec une même formule il est possible de calculer des moyennes de groupes, réaliser une ANOVA et construire la représentation graphique associée. En voici un exemple :

```
R> fm <- len ~ supp
    m <- aggregate(fm, data = ToothGrowth, mean)
    summary(aov(fm, data = ToothGrowth))
    bwplot(fm, data = ToothGrowth)
```

## Principaux types de graphiques avec `lattice` (et

## ggplot2 )

Même si le package `lattice` fournit moins de commandes que `ggplot2`, il n'en demeure pas moins qu'il est tout aussi facile de réaliser des représentations graphiques simples en un tour de main. Voici quelques exemples de représentations graphiques uni- et bivariées. Les données d'illustration sont les mêmes que celles utilisées plus haut ( `ToothGrowth` ): il s'agit d'une expérience de biologie dans laquelle on s'intéresse à la croissance des odontoblastes de cochons d'inde quantifiée par leur longueur (variable `len`) lorsqu'on administre à ces derniers de la vitamine C soit sous forme d'acide ascorbique soit sous forme de jus d'orange ( `supp` , `OJ` = jus d'orange), à différentes doses ( `dose` , en mg).

## Histogramme

Un histogramme d'effectifs se construit avec `histogram`. Puisqu'il s'agit de décrire une seule variable, ou sa distribution plus précisément, la formule à employer ne contient pas de variable à gauche du symbole `~` et l'on se contente d'écrire la variable à résumer à droite dans la formule :

```
R> histogram(~len, data = ToothGrowth, type = "count")
```

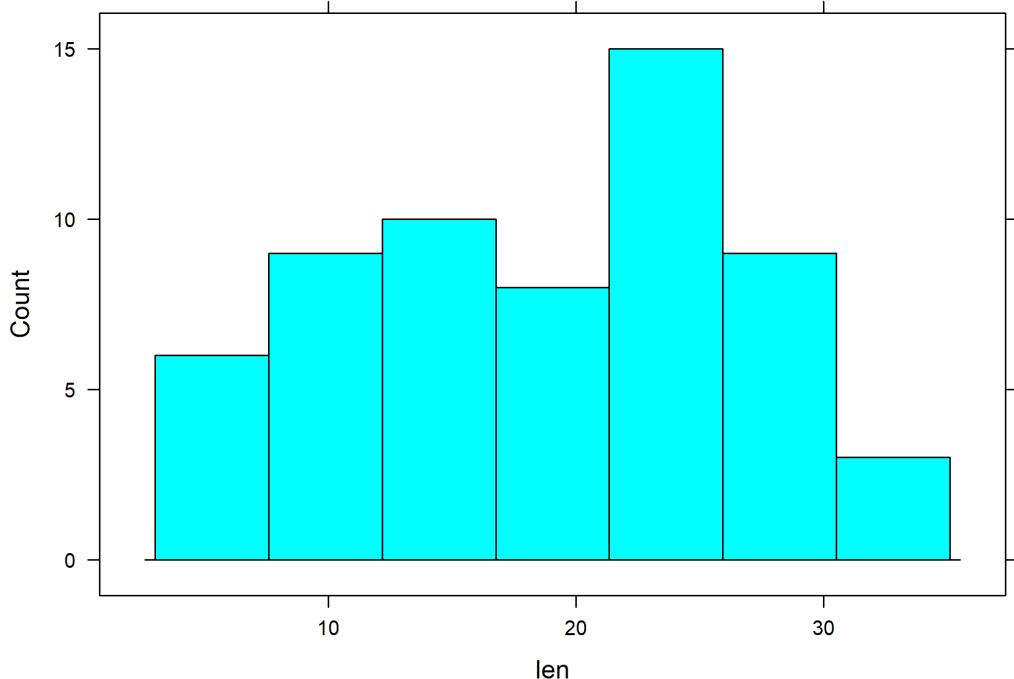


Figure 4. Histogramme d'effectifs

L'option `type = "count"` permet de forcer la représentation sous forme d'effectifs puisque, par défaut, c'est la densité qui est représentée. La formulation équivalente sous `ggplot2` serait :

```
R> ggplot(data = ToothGrowth, aes(x = len)) +  
  geom_histogram(binwidth = 5)
```

(Ou alors `qplot(x = len, data = ToothGrowth, geom = "histogram", binwidth = 5)`.)

En ajoutant une «facette» pour tenir compte de la variable `supp`, cela donne :

```
R> histogram(~ len | supp, data = ToothGrowth, breaks = seq(0, 40, by = 5))
```

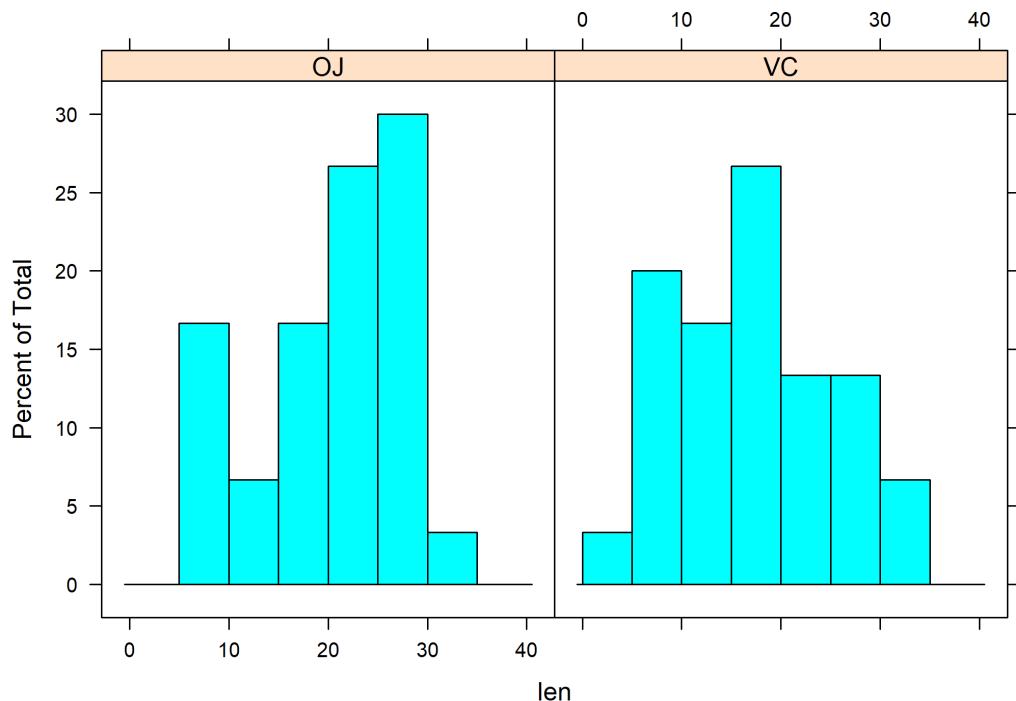


Figure 5. Histogramme d'effectifs conditionné sur une variable catégorielle

Avec `ggplot2`, les facettes sont gérées grâce aux commandes `facet_grid` et `facet_wrap`.

## Courbe de densité

Une courbe de densité se construit à l'aide de `densityplot` et la syntaxe est strictement identique à celle de `histogram`, à l'option `type=` près.

```
R> densityplot(~len, data = ToothGrowth, plot.points = FALSE, from = 0, to = 40)
```

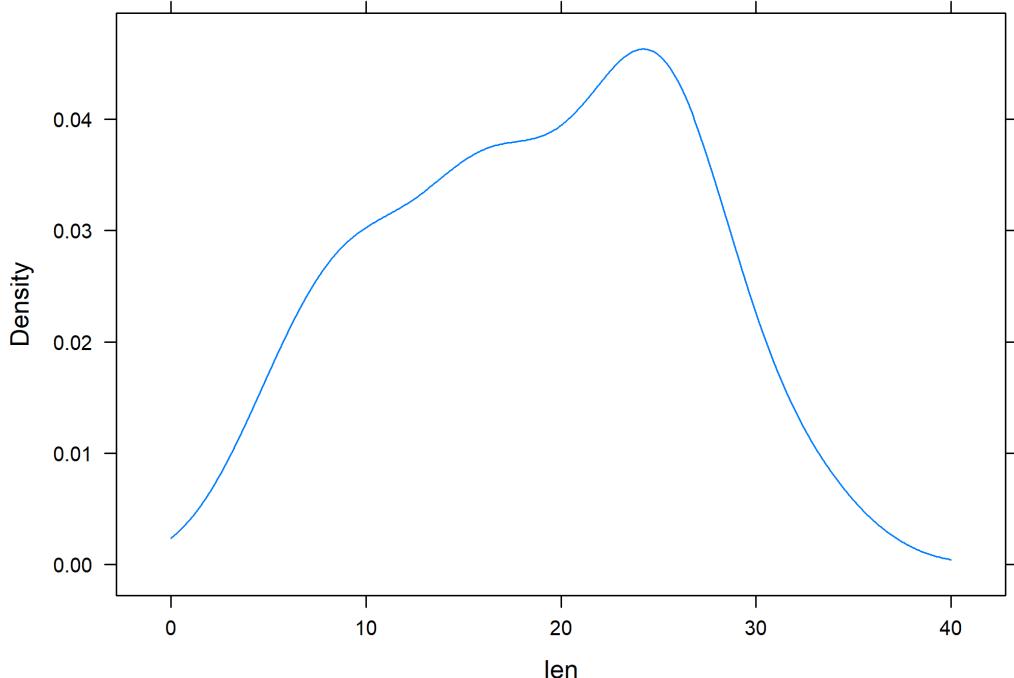


Figure 6. Courbe de densité

Il est possible de régler le paramètre de lissage à l'aide de l'option `bw=` : des valeurs plus élevées résultent en une courbe beaucoup plus lissée (essayez avec `bw = 10` !) et donc beaucoup moins sensible aux variations locales de la densité.

À ce stade, on peut en profiter pour discuter les options de conditionnement sur une variable catégorielle et la manière de gérer la présentation graphique : dans le cas d'un histogramme, il est délicat de superposer deux distributions ou plus sur le même graphique, même en ajoutant de la transparence, d'où l'idée de représenter les distributions dans des panneaux graphiques séparés. C'est ce qu'on a réalisé en indiquant que l'on souhaitait décrire la variable `len` conditionnellement aux valeurs prises par `supp` (`~ len | supp`). Dans ce cas, l'opérateur `|` invoque une facette et un découpage en autant de panneaux graphiques qu'il y a de valeurs uniques dans la variable `supp`. Une autre approche consiste à utiliser l'option `groups=`, et dans ce cas les différentes distributions seront affichées dans le même panneau graphique. Dans le cas d'une courbe de densité, cela revient à les superposer sur la même fenêtre graphique, avec un système de coordonnées unique. Les deux options de conditionnement peuvent être combinées naturellement.

Voici un exemple de graphique conditionnel un peu plus élaboré :

```
R> densityplot(~len,
  data = ToothGrowth, groups = supp, auto.key = TRUE, xlab = "len",
  par.settings = list(superpose.line = list(col = c("coral", "cornflowerblue"))))
)
```

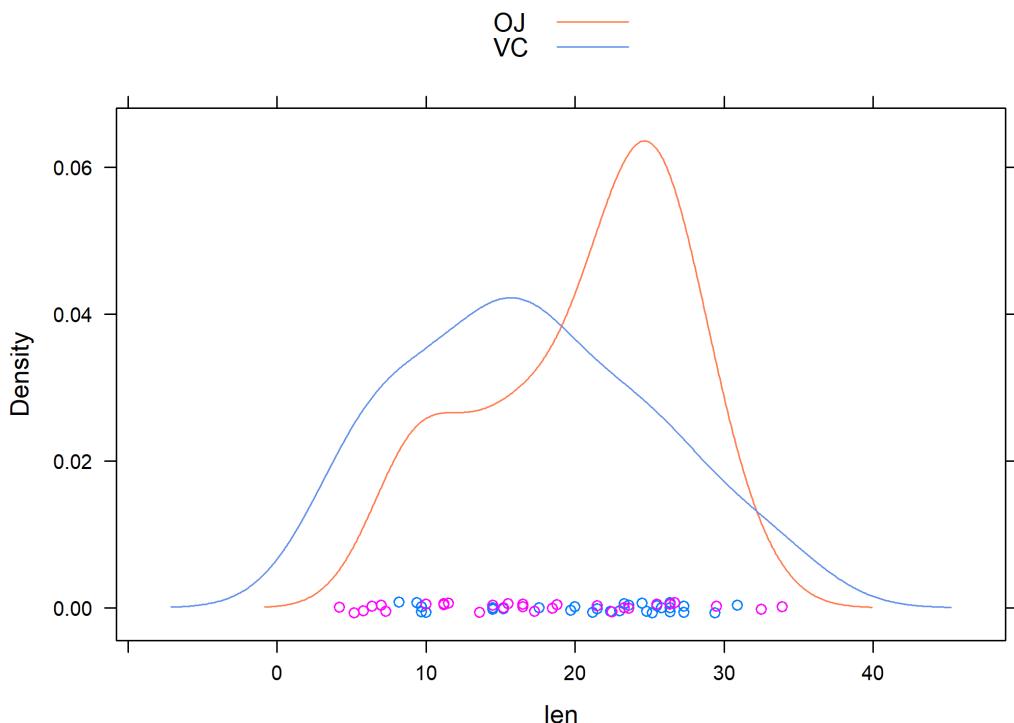
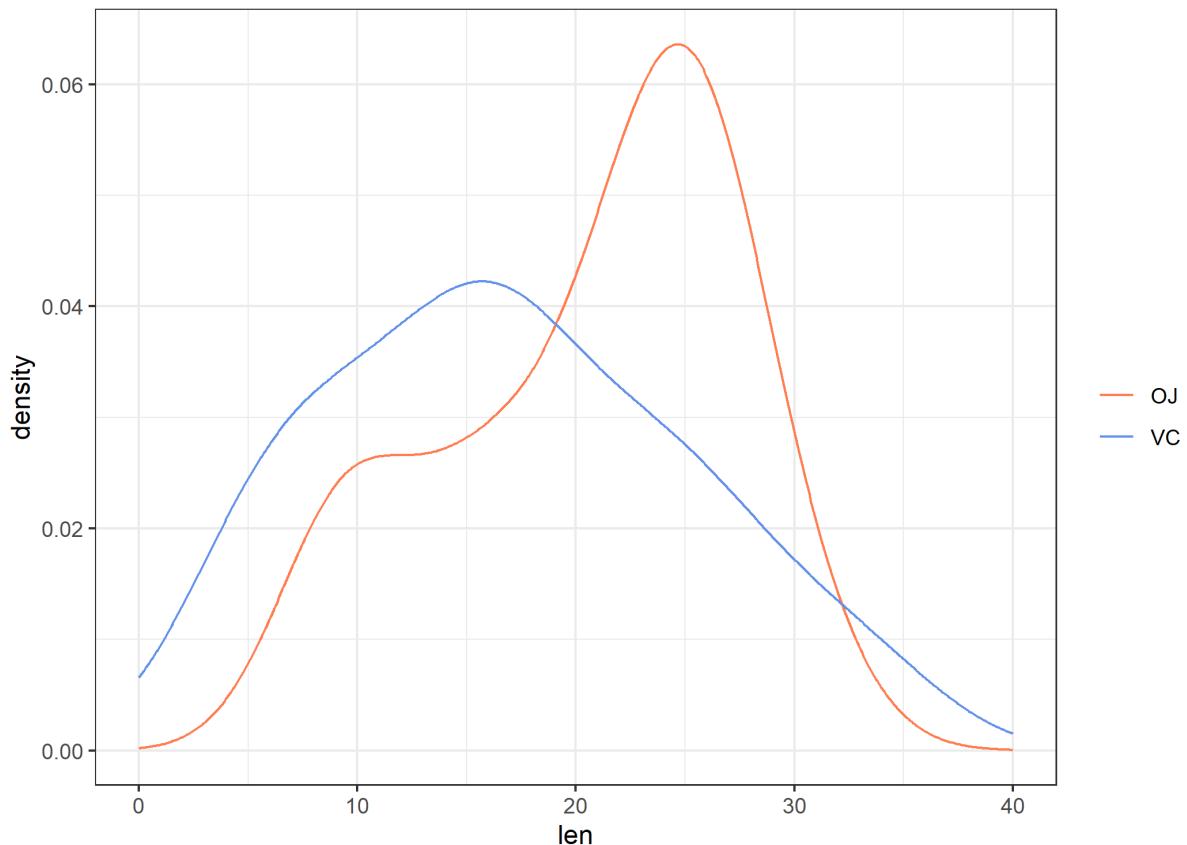


Figure 7. Courbe de densité conditionnelle

Au passage, on en a profité pour modifier le thème de couleur. Notez qu'en utilisant `par.settings=`, `lattice` se charge de coordonner les couleurs de la légende (`auto.key = TRUE`) avec celle des éléments graphiques correspondants.

L'équivalent sous `ggplot2` revient à peu près à l'instruction suivante :

```
R> ggplot(data = ToothGrowth) +  
  aes(x = len, colour = supp) +  
  geom_line(stat = "density") +  
  expand_limits(x = c(0, 40)) +  
  scale_colour_manual("", values = c("coral", "cornflowerblue")) +  
  theme_bw()
```



## Diagramme en barres

Les diagrammes en barres peuvent avantageusement être remplacés par des diagrammes en points, tels que les diagrammes de Cleveland (cf. plus loin), mais en attendant voici comment en réaliser un à l'aide de [barchart](#) à partir de données agrégées :

```
R> library(latticeExtra, quietly = TRUE)
```

Attaching package: 'latticeExtra'

The following object is masked from 'package:ggplot2':

layer

```
R> m <- aggregate(len ~ supp + dose, data = ToothGrowth, mean)
barchart(len ~ dose, data = m, groups = supp, horizontal = FALSE, auto.key
y = TRUE, par.settings = ggplot2like())
```

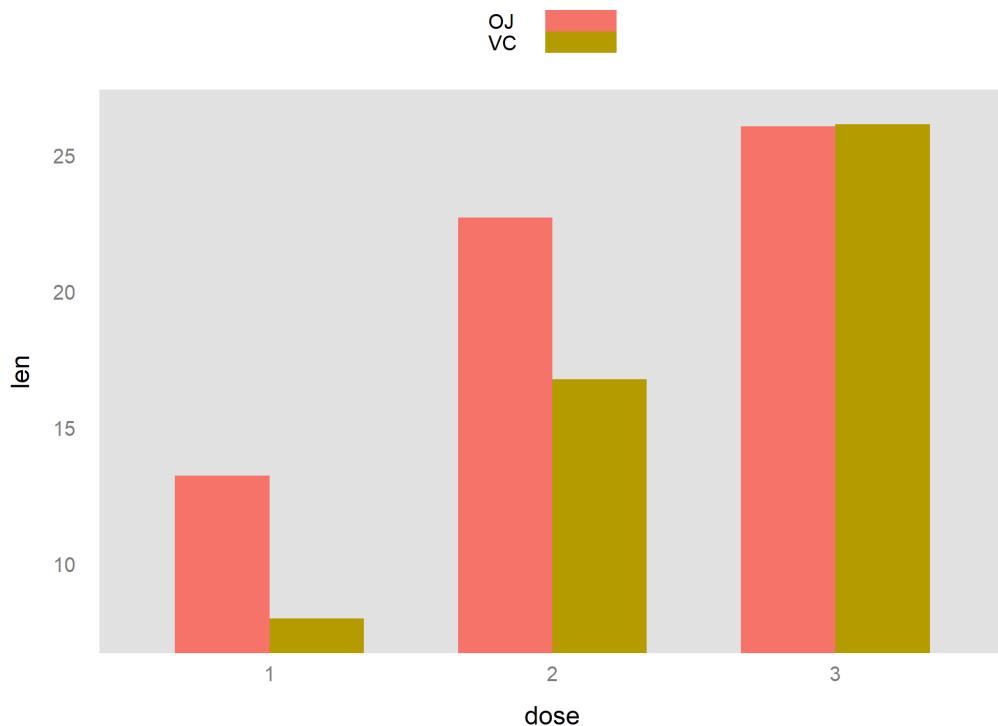


Figure 8. Diagramme en barres

Notons que `par.settings=` permet non seulement de fournir des options additionnelles pour contrôler le rendu des éléments graphiques (couleur, type de ligne ou de symboles, etc.) mais également d'utiliser des thèmes graphiques disponibles dans le package [latticeExtra](#).

## Diagramme de type boîtes à moustaches

Les diagrammes en forme de boîtes à moustaches sont obtenus à l’aide de la commande `bwplot`. Voici un exemple d’utilisation :

```
R> bwplot(len ~ supp, data = ToothGrowth, pch = "|")
```

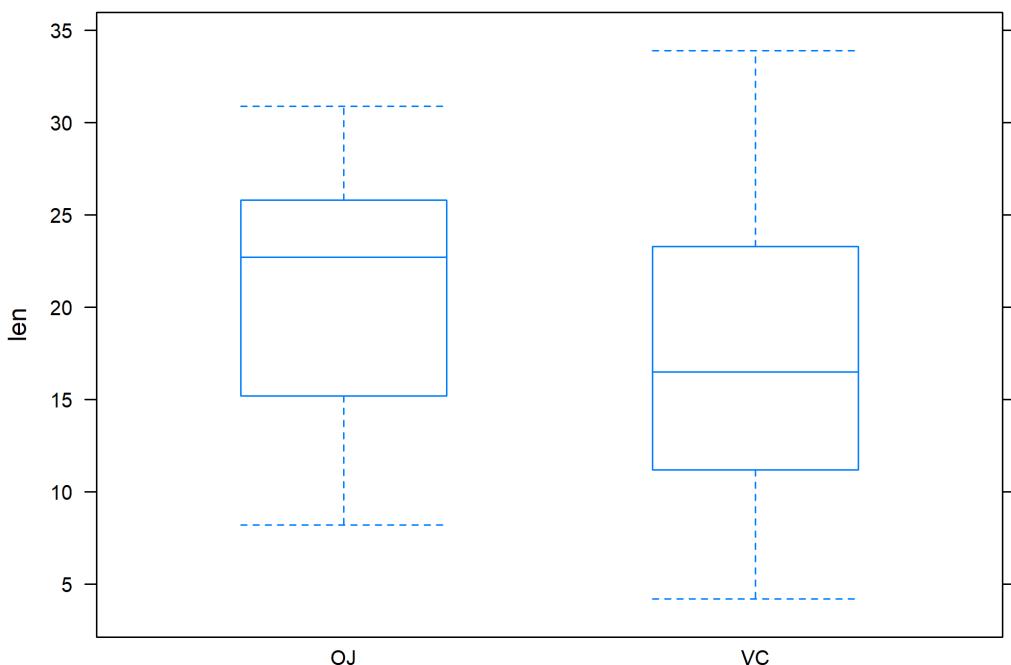


Figure 9. Diagramme en forme de boîtes à moustaches

L’option `pch=` permet de contrôler la manière dont la médiane est figurée dans la boîte. Par défaut il s’agit d’un simple point, mais si l’on souhaite utiliser les représentations plus classiques, telles que celles trouvées dans `boxplot` ou `geom_boxplot`, il suffit de suivre l’exemple ci-dessus. Notons que dans le cas de cette représentation graphique, le conditionnement sur la variable `supp` est d’emblée réalisé par l’utilisation d’une formule invoquant la variable de conditionnement à droite de l’opérateur `~`.

## Diagramme en points

Le même type de représentation graphique peut être obtenu en utilisant directement les données individuelles, et non leur résumé en cinq points (tel que fournit par `summary` et exploité par `bwplot`). Dans ce cas, il s'agit de la commande `dotplot`, qui permet de construire des diagrammes de Cleveland (moyenne ou effectif total calculé pour une variable en fonction des niveaux d'une autre variable) ou, dans le cas où la variable à résumer consiste en une série de mesures individuelles numériques, des diagrammes de dispersion. Voici une illustration pour ce dernier cas de figure :

```
R> dotplot(len ~ supp, ToothGrowth, jitter.x = TRUE)
```

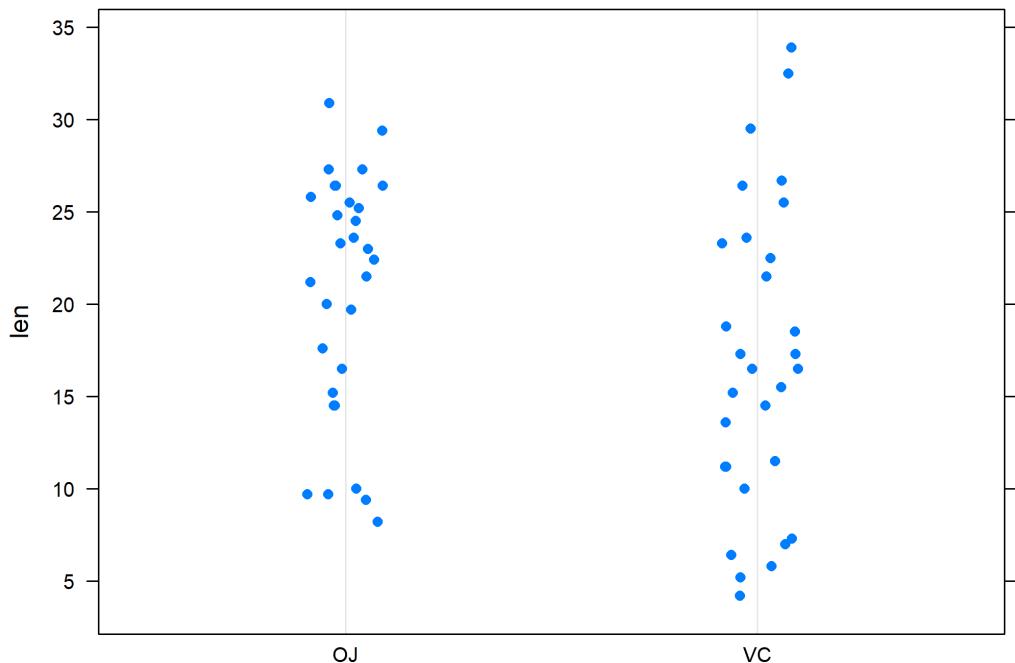


Figure 10. Diagramme en points

## Diagramme de dispersion

Enfin, un diagramme de dispersion est construit à l'aide de la commande `xypplot`.

```
R> xyplot(len ~ dose, ToothGrowth, type = c("p", "smooth"))
```

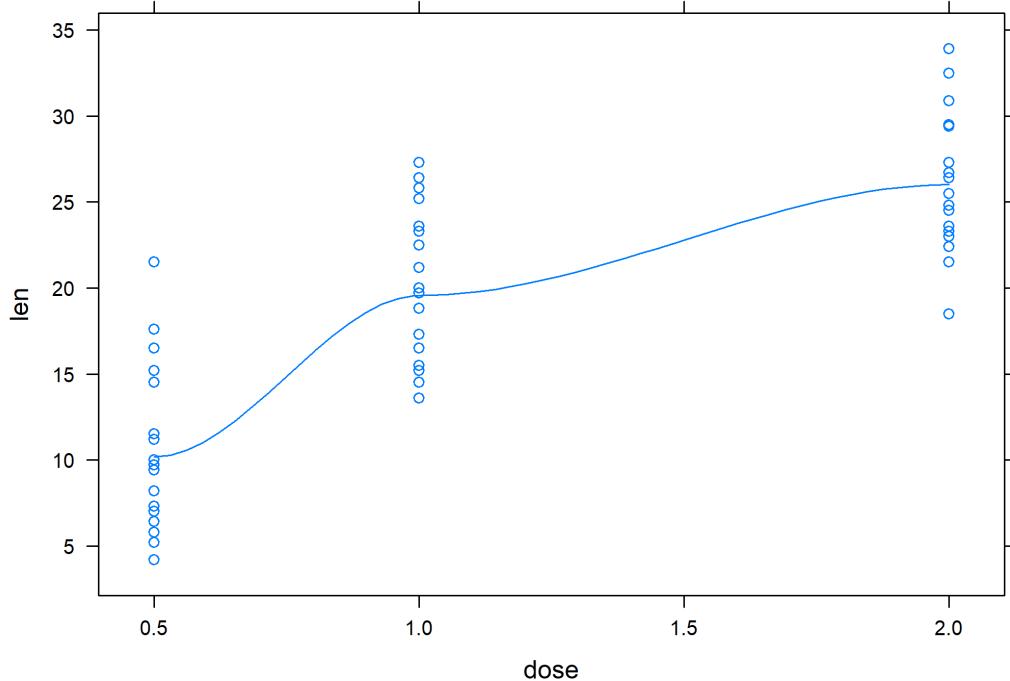


Figure 11. Diagramme de dispersion

Même si l’exemple ne s’y prête guère, on en a profité pour ajouter une **courbe lowess** de régression afin d’indiquer la tendance de covariation entre les deux variables numériques. L’aide en ligne pour `xyplot` n’est pas très utile dans ce cas, et il faut en fait aller regarder les options de personnalisation disponibles dans la sous-fonction correspondante : `panel.xyplot`.

# Cartes

Pour une présentation de l'analyse spatiale sous R, se référer au chapitre dédié, page 787.

Il existe de multiple approches pour réaliser des cartes sous R, y compris avec **ggplot2**, mais également de manière native avec les extensions **sp** et **sf**. Il existe également des extensions apportant des fonctionnalités additionnelles comme **ggmap**, **mapview** ou encore **tmap**.

Pour une introduction succincte en français, on pourra se référer à la section 5.4 du support de cours **Logiciel R et programmation** d'Ewan Gallic.

Voir également l'excellente présentation *Données géospatiales et cartographie avec R* de Nicolas Roelandt : [https://roelandtn.frama.io/slides/2090628\\_meetup\\_Raddict\\_datageo.html](https://roelandtn.frama.io/slides/2090628_meetup_Raddict_datageo.html).

En complément (en anglais), la vignette **Plotting Simple Features** de l'extension **sf** ou encore le chapitre **Making maps with R** de l'ouvrage *Geocomputation with R* de Robin Lovelace, Jakub Nowosad et Jannes Muenchow.

On pourra également se référer à l'excellent package **cartography** et à son site dédié : <https://github.com/riatelab/cartography/> (pour une présentation en vidéo et en français : [https://youtu.be/OI3\\_AOg6pfc](https://youtu.be/OI3_AOg6pfc)).



# Autres extensions graphiques

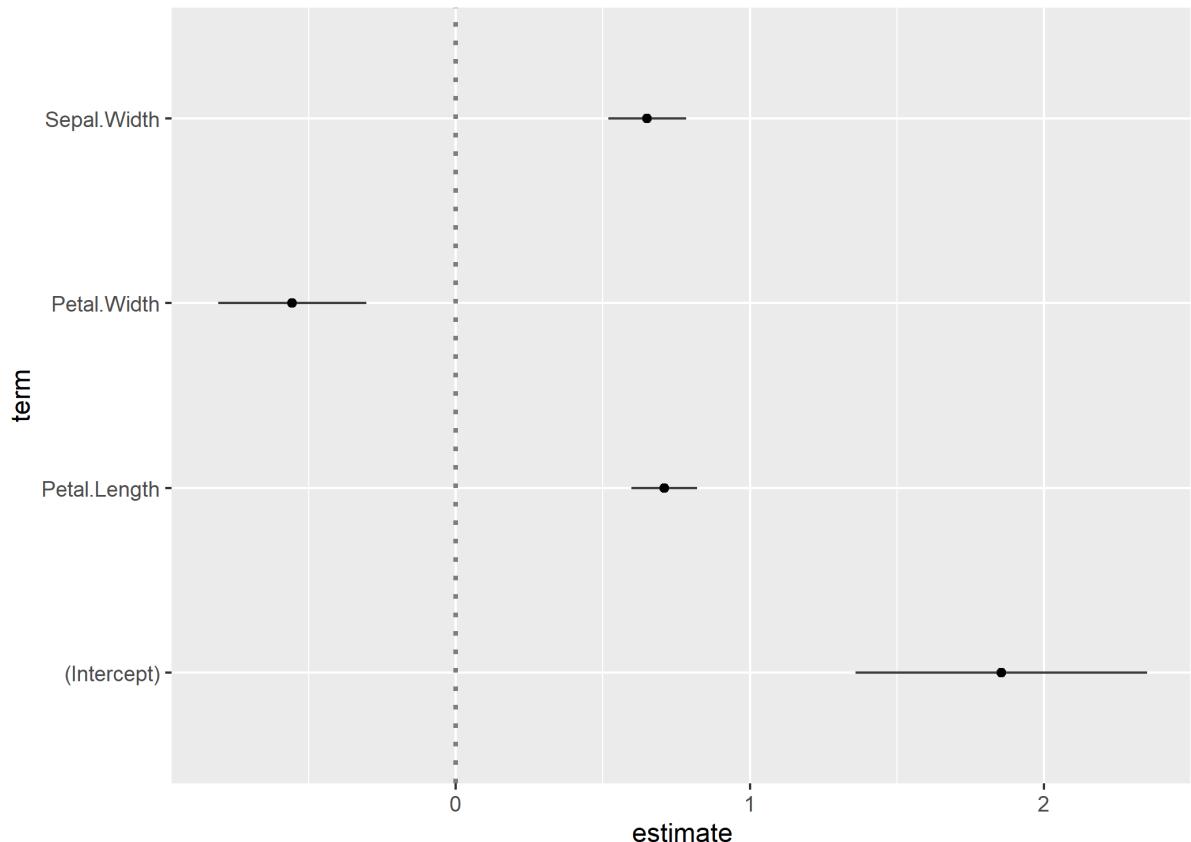
GGally .....	897
ggnetwork .....	899
ggdendro .....	904
circlize .....	906
Diagrammes de Sankey .....	911
DiagrammeR .....	914
highcharter .....	920

Pour trouver l'inspiration et des exemples de code, rien ne vaut l'excellent site <https://www.r-graph-gallery.com/>.

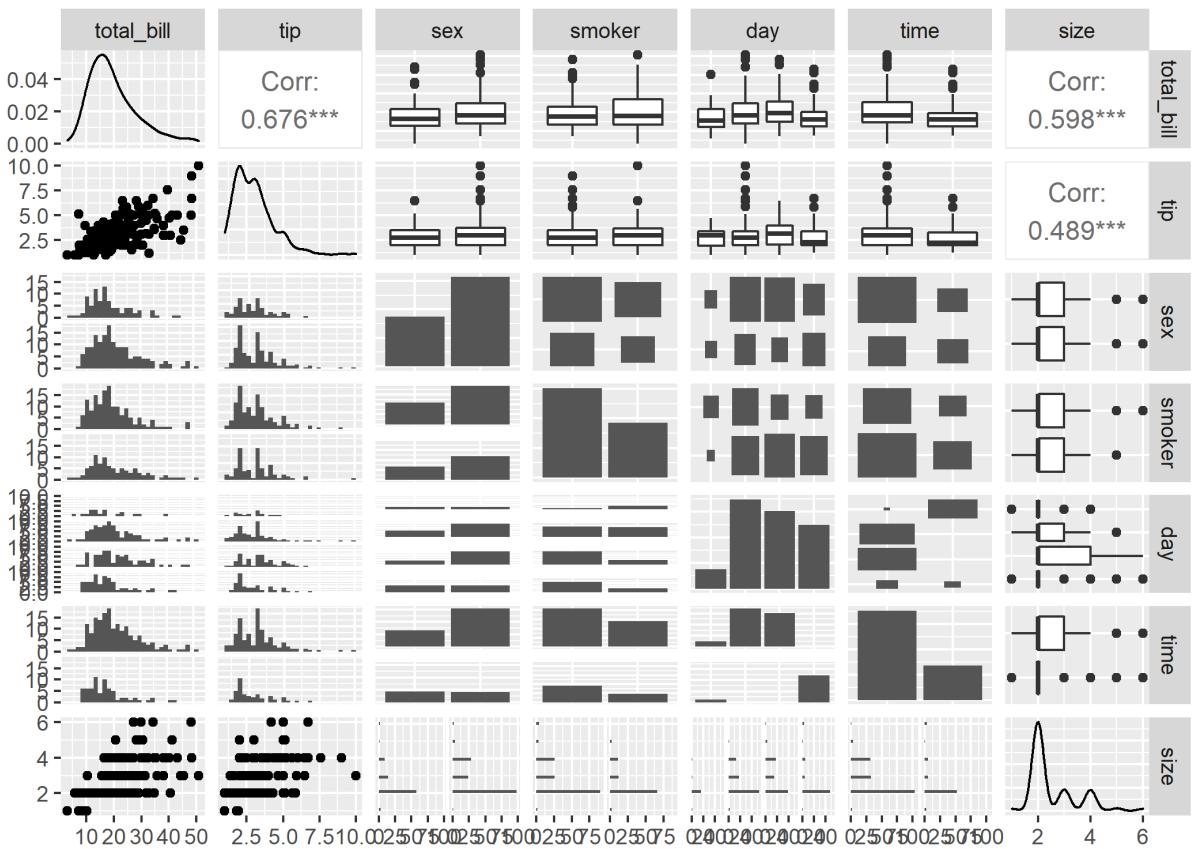
## GGally

L'extension **GGally**, déjà abordée dans d'autres chapitres, fournit plusieurs fonctions graphiques d'exploration des résultats d'un modèle ou des relations entre variables.

```
R> reg <- lm(Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width, data = iris)
  library(GGally)
  ggcoef(reg)
```



```
R> data(tips, package = "reshape")
ggpairs(tips)
```

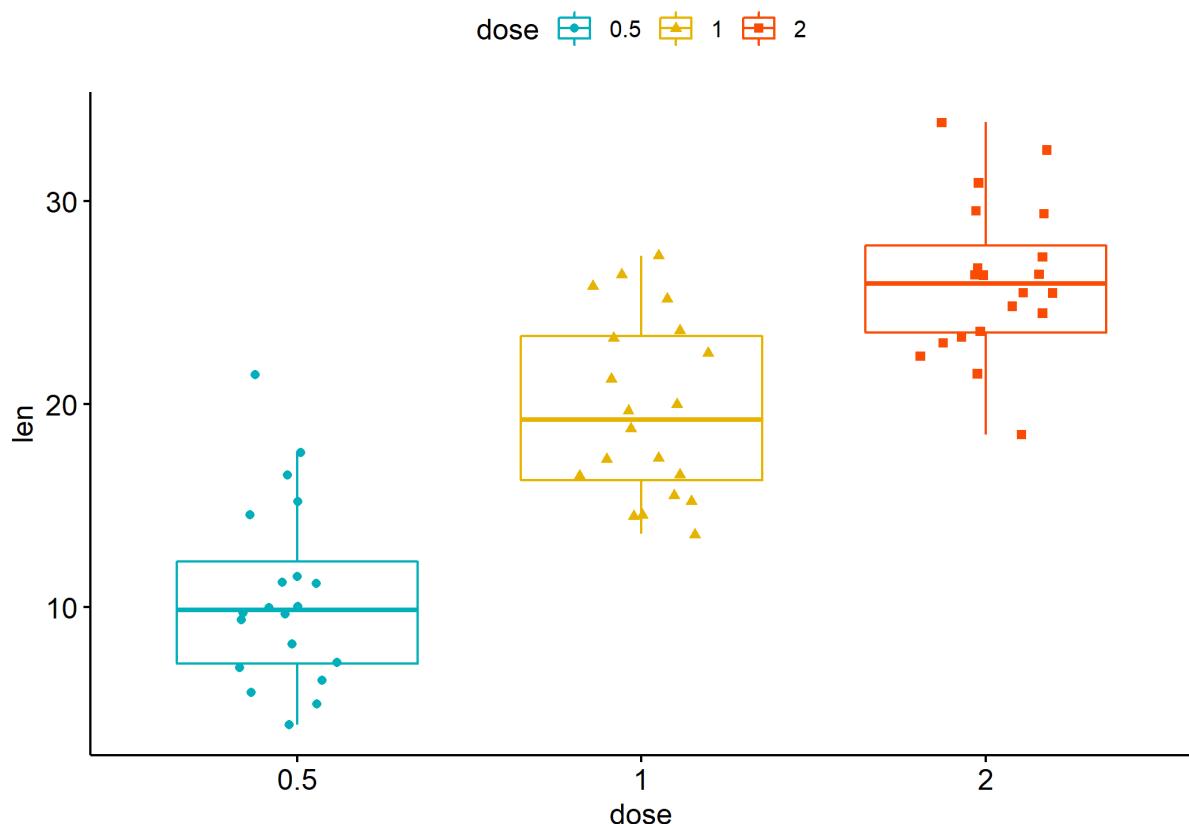


Plus d'information : <https://ggobi.github.io/ggally/>

## ggpubr

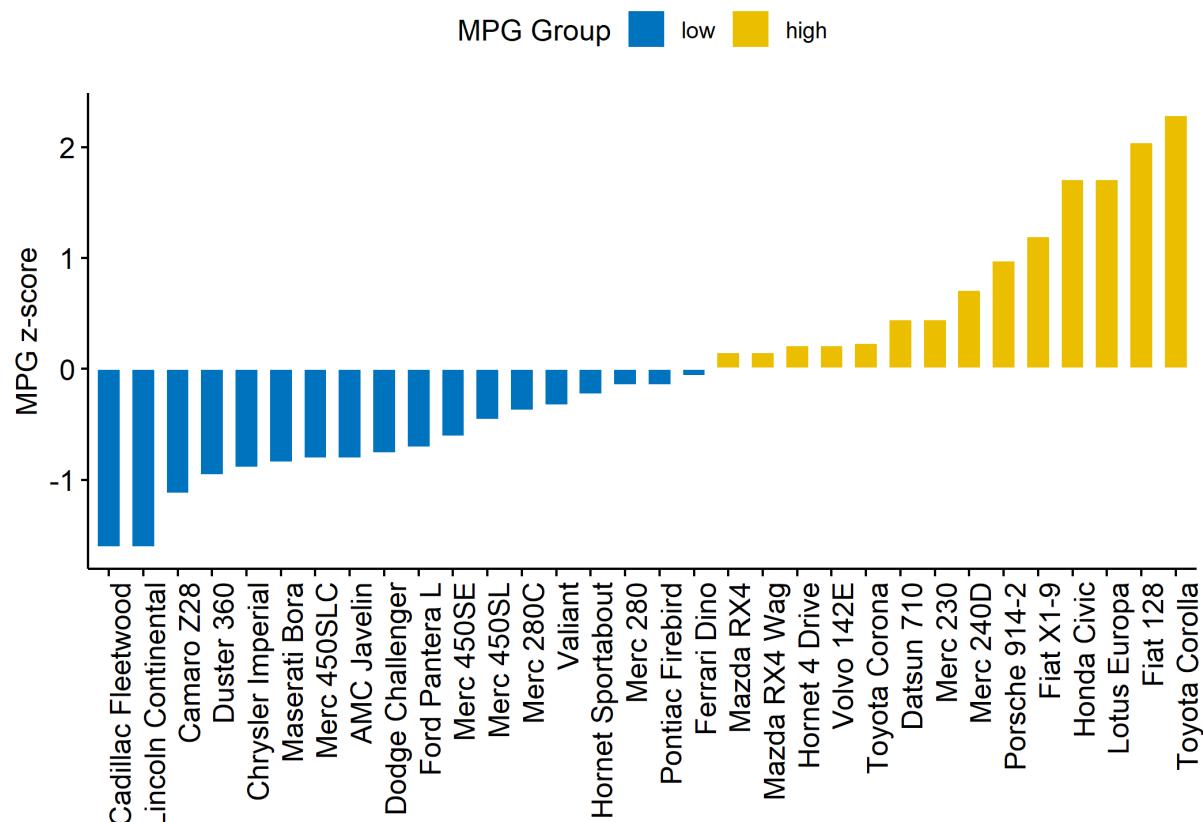
L'extension **ggpubr** fournit plusieurs fonctions pour produire «clés en main» différents graphiques bivariés avec une mise en forme allégée.

```
R> library(ggpubr)
data("ToothGrowth")
df <- ToothGrowth
ggboxplot(df,
  x = "dose", y = "len",
  color = "dose", palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  add = "jitter", shape = "dose"
)
```

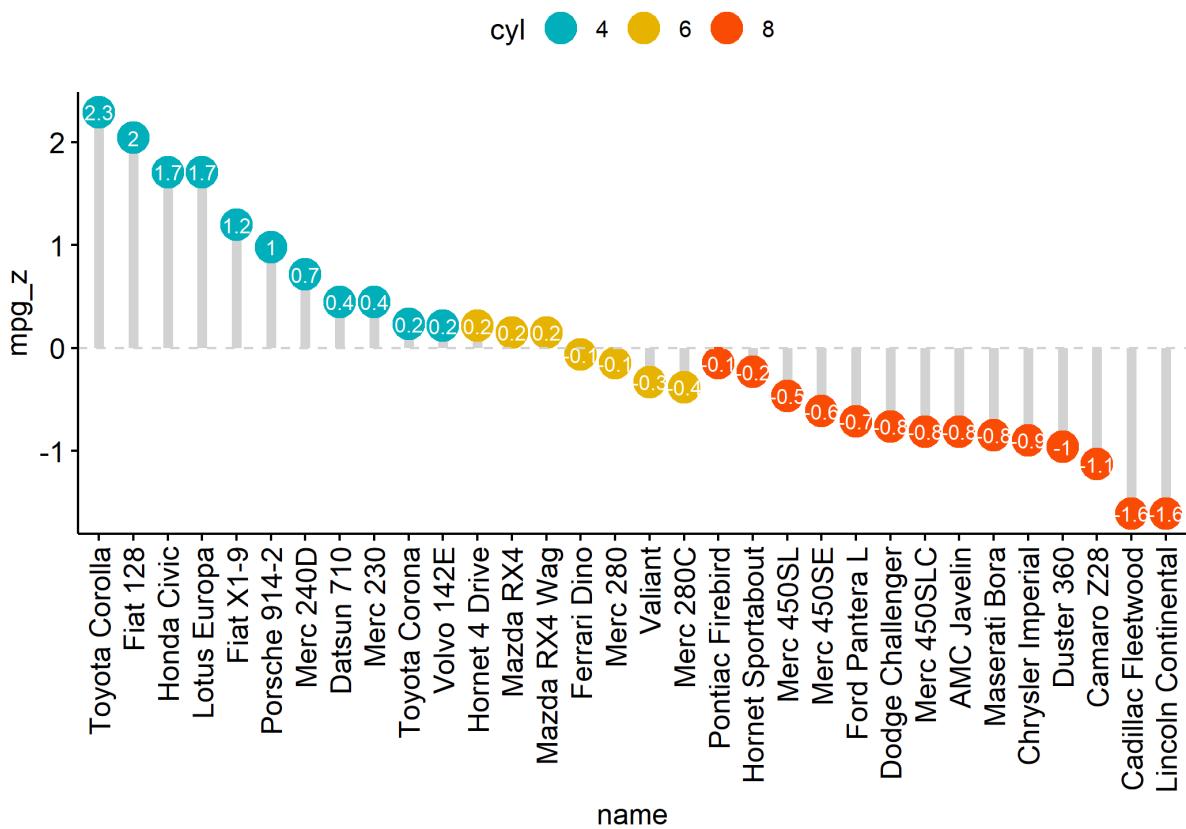


```
R> data("mtcars")
dfm <- mtcars
# Convert the cyl variable to a factor
dfm$cyl <- as.factor(dfm$cyl)
# Add the name columns
dfm$name <- rownames(dfm)
# Calculate the z-score of the mpg data
dfm$mpg_z <- (dfm$mpg - mean(dfm$mpg)) / sd(dfm$mpg)
dfm$mpg_grp <- factor(ifelse(dfm$mpg_z < 0, "low", "high"),
  levels = c("low", "high"))
)

ggbarplot(dfm,
  x = "name", y = "mpg_z",
  fill = "mpg_grp", # change fill color by mpg_level
  color = "white", # Set bar border colors to white
  palette = "jco", # jco journal color palett. see ?ggpar
  sort.val = "asc", # Sort the value in ascending order
  sort.by.groups = FALSE, # Don't sort inside each group
  x.text.angle = 90, # Rotate vertically x axis texts
  ylab = "MPG z-score",
  xlab = FALSE,
  legend.title = "MPG Group"
)
```



```
R> ggdotchart(dfm,
  x = "name", y = "mpg_z",
  color = "cyl", # Color by groups
  palette = c("#00AFBB", "#E7B800", "#FC4E07"), # Custom color palette
  sorting = "descending", # Sort value in descending order
  add = "segments", # Add segments from y = 0 to dots
  add.params = list(color = "lightgray", size = 2), # Change segment color and size
  group = "cyl", # Order by groups
  dot.size = 6, # Large dot size
  label = round(dfm$mpg_z, 1), # Add mpg values as dot labels
  font.label = list(
    color = "white", size = 9,
    vjust = 0.5
  ), # Adjust label parameters
  ggtheme = theme_pubr() # ggplot2 theme
) +
  geom_hline(yintercept = 0, linetype = 2, color = "lightgray")
```

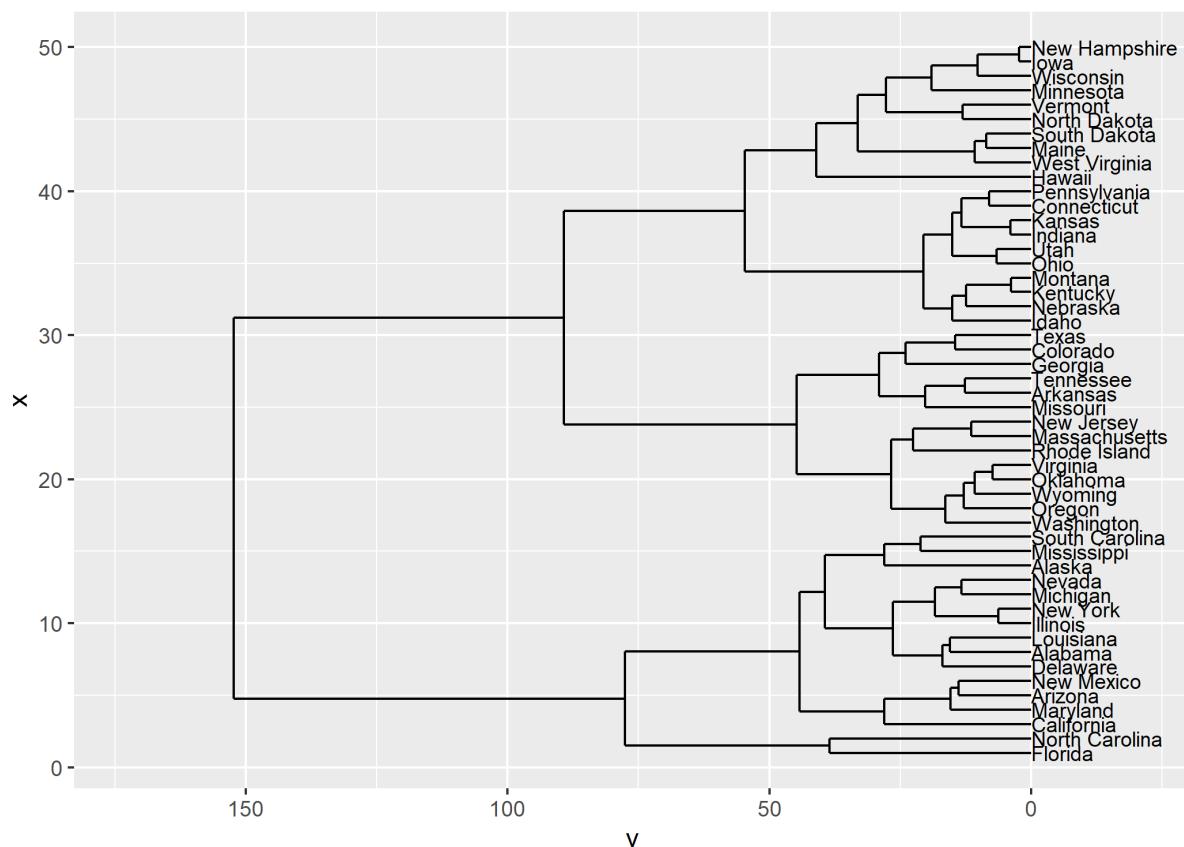


Plus d'informations : <https://rpkgs.datanovia.com/ggpubr/>

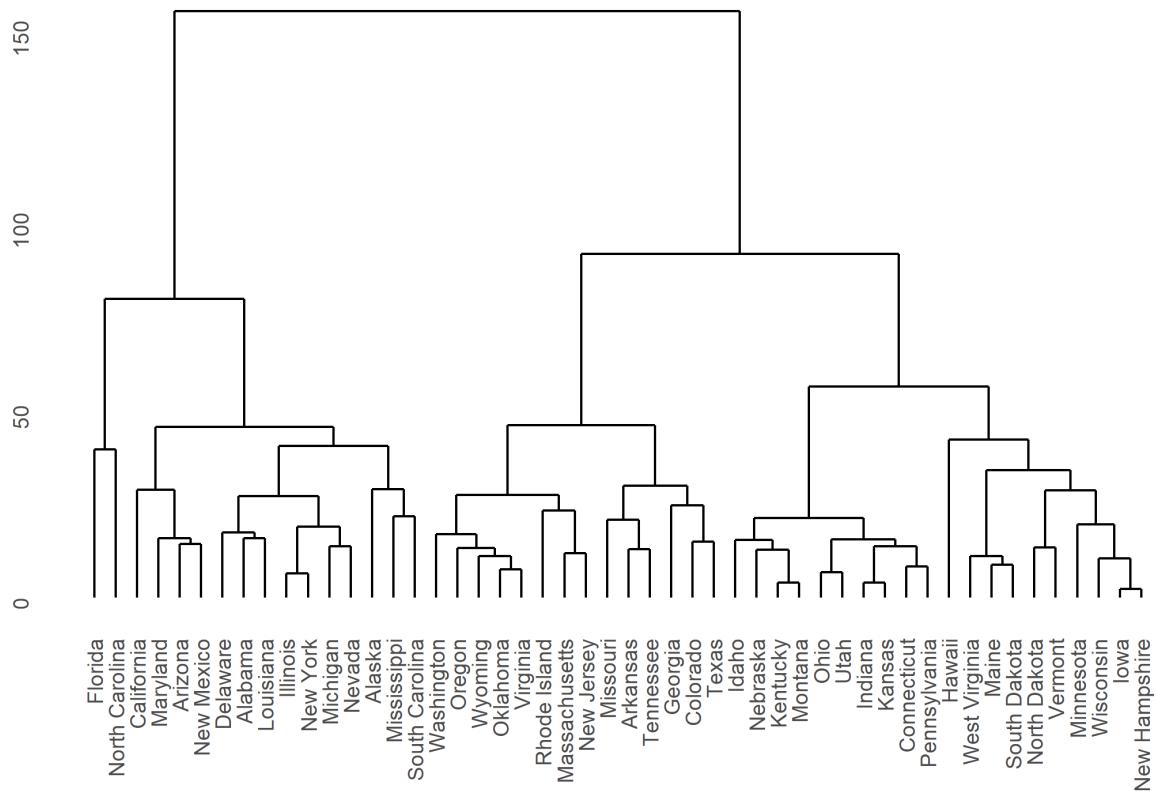
## ggdendro

L'extension **ggendro** avec sa fonction **ggdendrogram** permet de représenter facilement des dendrogrammes avec **ggplot2**.

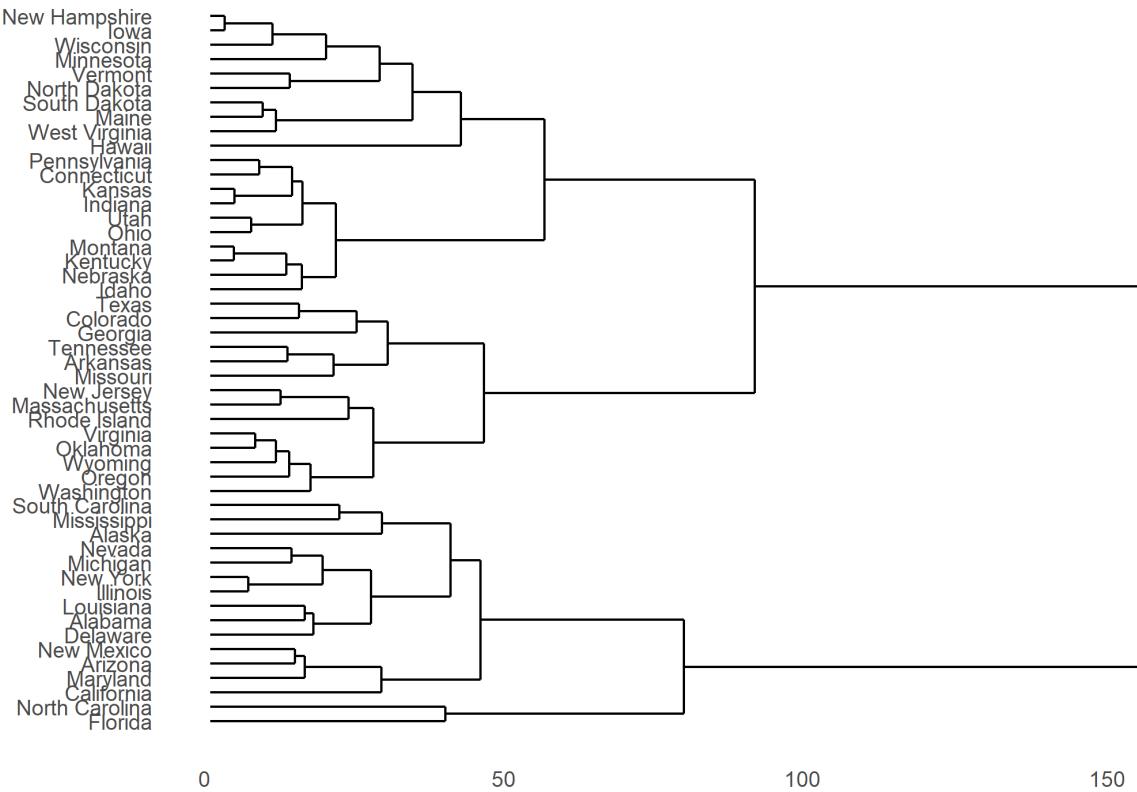
```
R> library(ggplot2)
library(ggdendro)
hc <- hclust(dist(USArrests), "ave")
hcdta <- dendro_data(hc, type = "rectangle")
ggplot() +
  geom_segment(data = segment(hcdta), aes(x = x, y = y, xend = xend, yend = yend)) +
  geom_text(data = label(hcdta), aes(x = x, y = y, label = label, hjust = 0), size = 3) +
  coord_flip() +
  scale_y_reverse(expand = c(0.2, 0))
```



```
R> ### demonstrate plotting directly from object class hclust  
ggdendrogram(hc)
```



```
R> ggdendrogram(hc, rotate = TRUE)
```



Plus d'informations : <https://cran.r-project.org/web/packages/ggdendro/vignettes/ggdendro.html>

## circlize

L'extension **circlize** est l'extension de référence quand il s'agit de représentations circulaires. Un ouvrage entier lui est dédié : [https://jokergoo.github.io/circlize\\_book/book/](https://jokergoo.github.io/circlize_book/book/).

Voici un exemple issu de <https://www.data-to-viz.com/story/AdjacencyMatrix.html>.

```
R> library(tidyverse)
```

```
-- Attaching packages ----- tidyverse 1.3.0 --
v tibble 3.0.4     v purrr   0.3.4
v readr   1.4.0     v forcats 0.5.0
```

```
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::arrange()      masks plyr::arrange()
x lubridate::as.difftime() masks base::as.difftime()
x dplyr::between()      masks data.table::between()
x purrr::compact()      masks plyr::compact()
x dplyr::count()        masks plyr::count()
x lubridate::date()     masks base::date()
x tidyverse::extract()   masks magrittr::extract()
x dplyr::failwith()    masks plyr::failwith()
x dplyr::filter()       masks stats::filter()
x dplyr::first()        masks data.table::first()
x lubridate::hour()     masks data.table::hour()
x dplyr::id()           masks plyr::id()
x lubridate::intersect() masks base::intersect()
x lubridate::isoweek()  masks data.table::isoweek()
x dplyr::lag()          masks stats::lag()
x dplyr::last()         masks data.table::last()
x lubridate::mday()     masks data.table::mday()
x lubridate::minute()   masks data.table::minute()
x lubridate::month()    masks data.table::month()
x ggpubr::mutate()     masks dplyr::mutate(), plyr::mutate()
x lubridate::quarter()  masks data.table::quarter()
x dplyr::rename()       masks plyr::rename()
x lubridate::second()   masks data.table::second()
x purrr::set_names()   masks magrittr::set_names()
x lubridate::setdiff()  masks base::setdiff()
x dplyr::summarise()   masks plyr::summarise()
x dplyr::summarize()   masks plyr::summarize()
x purrr::transpose()   masks data.table::transpose()
x lubridate::union()   masks base::union()
x lubridate::wday()     masks data.table::wday()
x lubridate::week()     masks data.table::week()
x lubridate::yday()     masks data.table::yday()
x lubridate::year()     masks data.table::year()
```

```
R> # Load data
  data <- read.table("https://raw.githubusercontent.com/holtzy/data_to_viz/master/Example_dataset/13_AdjacencyDirectedWeighted.csv", header = TRUE)
  # short names
  colnames(data) <- c("Africa", "East Asia", "Europe", "Latin Ame.", "North America.", "Oceania", "South Asia", "South East Asia", "Soviet Union", "West.Aisia")
  rownames(data) <- colnames(data)

  # I need a long format
  data_long <- data %>%
    rownames_to_column() %>%
    gather(key = "key", value = "value", -rowname)

library(circlize)
```

```
=====
circlize version 0.4.11
CRAN page: https://cran.r-project.org/package=circlize
Github page: https://github.com/jokergoo/circlize
Documentation: https://jokergoo.github.io/circlize_book/book/
```

If you use it in published research, please cite:  
Gu, Z. circlize implements and enhances circular visualization  
in R. Bioinformatics 2014.

This message can be suppressed by:  
suppressPackageStartupMessages(library(circlize))

```
=====
```

```
R> # parameters
  circos.clear()
  circos.par(start.degree = 90, gap.degree = 4, track.margin = c(-0.1, 0.1), points.overflow.warning = FALSE)
  par(mar = rep(0, 4))

  # color palette
  library(viridis)
```

Loading required package: viridisLite

```
R> mycolor <- viridis(10, alpha = 1, begin = 0, end = 1, option = "D")
mycolor <- mycolor[sample(1:10)]

# Base plot
chordDiagram(
  x = data_long,
  grid.col = mycolor,
  transparency = 0.25,
  directional = 1,
  direction.type = c("arrows", "diffHeight"),
  diffHeight = -0.04,
  annotationTrack = "grid",
  annotationTrackHeight = c(0.05, 0.1),
  link.arr.type = "big.arrow",
  link.sort = TRUE,
  link.largest.onTop = TRUE
)

# Add text and axis
circos.trackPlotRegion(
  track.index = 1,
  bg.border = NA,
  panel.fun = function(x, y) {
    xlim <- get.cell.meta.data("xlim")
    sector.index <- get.cell.meta.data("sector.index")

    # Add names to the sector.
    circos.text(
      x = mean(xlim),
      y = 3.2,
      labels = sector.index,
      facing = "bending",
      cex = 0.8
    )

    # Add graduation on axis
    circos.axis(
      h = "top",
      major.at = seq(from = 0, to = xlim[2], by = ifelse(test = xlim[2] > 10,
yes = 2, no = 1)),
      minor.ticks = 1,
      major.tick.percentage = 0.5,
      labels.niceFacing = FALSE
    )
  }
)
```

```
`major.tick.percentage` is not used any more, please directly use argument `majo  
r.tick.length`.
```

```
`major.tick.percentage` is not used any more, please directly use argument `majo  
r.tick.length`.
```

```
`major.tick.percentage` is not used any more, please directly use argument `majo  
r.tick.length`.
```

```
`major.tick.percentage` is not used any more, please directly use argument `majo  
r.tick.length`.
```

```
`major.tick.percentage` is not used any more, please directly use argument `majo  
r.tick.length`.
```

```
`major.tick.percentage` is not used any more, please directly use argument `majo  
r.tick.length`.
```

```
`major.tick.percentage` is not used any more, please directly use argument `majo  
r.tick.length`.
```

```
`major.tick.percentage` is not used any more, please directly use argument `majo  
r.tick.length`.
```

```
`major.tick.percentage` is not used any more, please directly use argument `majo  
r.tick.length`.
```

```
`major.tick.percentage` is not used any more, please directly use argument `majo  
r.tick.length`.
```



## Diagrammes de Sankey

Les [diagrammes de Sankey](#) sont un type alternatif de représentation de flux. Voici un premier exemple, qui reprend les données utilisées pour le diagramme circulaire précédent, avec la fonction `sankeyNetwork` de l'extension `sankeyNetwork`.

```
R> # Package
library(networkD3)

# I need a long format
data_long <- data %>%
  rownames_to_column() %>%
  gather(key = "key", value = "value", -rowname) %>%
  filter(value > 0)
colnames(data_long) <- c("source", "target", "value")
data_long$target <- paste(data_long$target, " ", sep = "")

# From these flows we need to create a node data frame: it lists every entities involved in the flow
nodes <- data.frame(name = c(as.character(data_long$source), as.character(data_long$target))) %>% unique()

# With networkD3, connection must be provided using id, not using real name like in the links dataframe.. So we need to reformat it.
data_long$IDsource <- match(data_long$source, nodes$name) - 1
data_long$IDtarget <- match(data_long$target, nodes$name) - 1

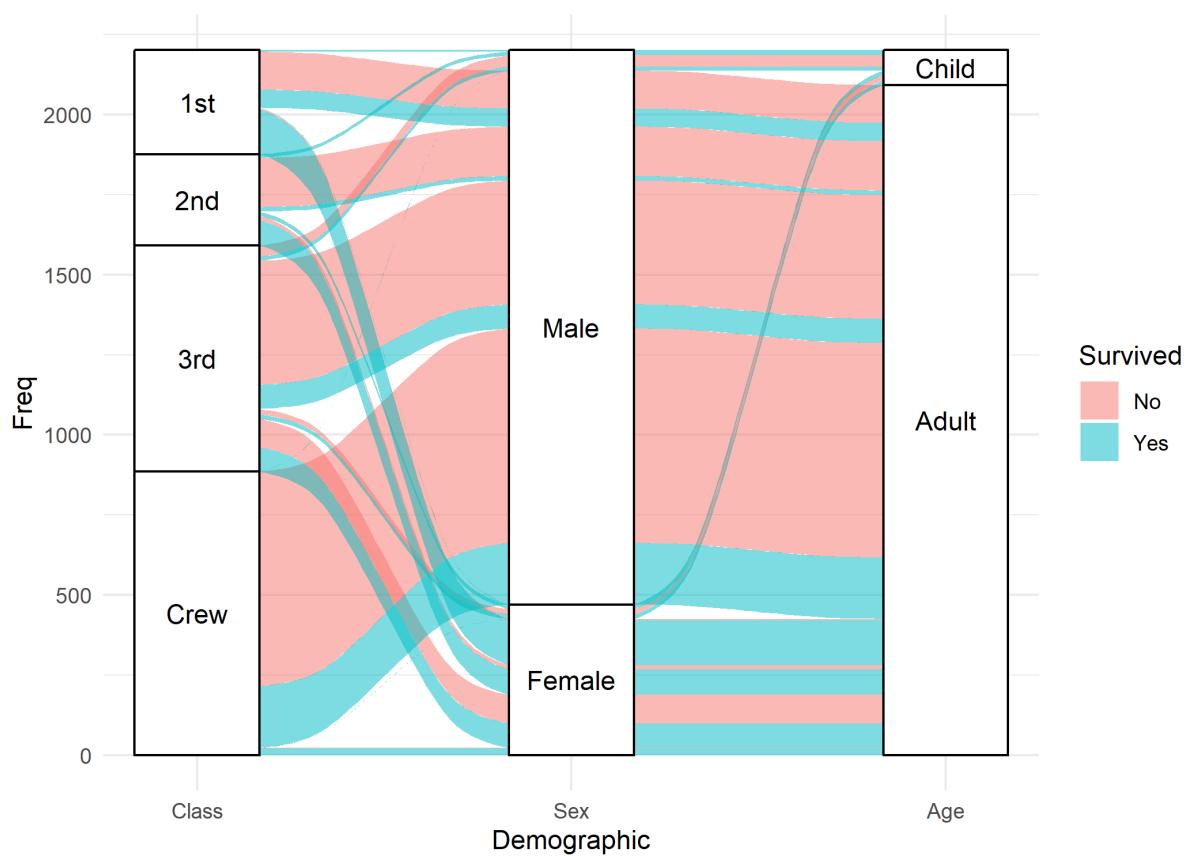
# prepare colour scale
ColourScal <- 'd3.scaleOrdinal() .range(["#FDE725FF","#B4DE2CFF","#6DCD59FF","#35B779FF","#1F9E89FF","#26828EFF","#31688EFF","#3E4A89FF","#482878FF","#40154FFF"])' 

# Make the Network
sankeyNetwork(
  Links = data_long, Nodes = nodes,
  Source = "IDsource", Target = "IDtarget",
  Value = "value", NodeID = "name",
  sinksRight = FALSE, colourScale = ColourScal, nodeWidth = 40, fontSize = 13, nodePadding = 20
)
```

Une alternative possible est fournie par l'extension `ggalluvial` et ses géométries `geom_alluvium` et `geom_stratum`.

```
R> library(ggalluvial)
ggplot(data = as.data.frame(Titanic)) +
  aes(axis1 = Class, axis2 = Sex, axis3 = Age, y = Freq) +
  scale_x_discrete(limits = c("Class", "Sex", "Age"), expand = c(.1, .05)) +
  xlab("Demographic") +
  geom_alluvium(aes(fill = Survived)) +
  geom_stratum() +
  geom_text(stat = "stratum", infer.label = TRUE) +
  theme_minimal()
```

Warning: The parameter `infer.label` is deprecated.  
Use `aes(label = after\_stat(stratum))`.



Mentionnons également l'extension [riverplot](#) pour la création de diagrammes de Sankey.

## DiagrammeR

**DiagrammeR** est dédiée à la réalisation de diagrammes en ayant recours à la syntaxe **Graphviz** (via la fonction [grViz](#)) ou encore à la syntaxe **Mermaid** (via la fonction [mermaid](#)).

```
R> library(DiagrammeR)
grViz("
digraph boxes_and_circles {

    # a 'graph' statement
    graph [overlap = true, fontsize = 10]

    # several 'node' statements
    node [shape = box,
          fontname = Helvetica]
    A; B; C; D; E; F

    node [shape = circle,
          fixedsize = true,
          width = 0.9] // sets as circles
    1; 2; 3; 4; 5; 6; 7; 8

    # several 'edge' statements
    A->1 B->2 B->3 B->4 C->A
    1->D E->A 2->4 1->5 1->F
    E->6 4->6 5->7 6->7 3->8
}

")
```

```
R> mermaid("graph LR\nA(Rounded) --> B[Rectangular]\nB --> C{A Rhombus}\nC --> D[Rectangle One]\nC --> E[Rectangle Two]\n")
```

```
R> mermaid("sequenceDiagram
    customer->>ticket seller: ask ticket
    ticket seller->>database: seats
    alt tickets available
        database->>ticket seller: ok
        ticket seller->>customer: confirm
        customer->>ticket seller: ok
        ticket seller->>database: book a seat
        ticket seller->>printer: print ticket
    else sold out
        database->>ticket seller: none left
        ticket seller->>customer: sorry
    end
")
```



```
R> mermaid("
gantt
    dateFormat  YYYY-MM-DD
    title Adding GANTT diagram functionality to mermaid

    section A section
    Completed task      :done,      des1, 2014-01-06,2014-01-08
    Active task         :active,   des2, 2014-01-09, 3d
    Future task         :         des3, after des2, 5d
    Future task2        :         des4, after des3, 5d

    section Critical tasks
    Completed task in the critical line :crit, done, 2014-01-06,24h
    Implement parser and jison          :crit, done, after des1, 2d
    Create tests for parser            :crit, active, 3d
    Future task in critical line       :crit, 5d
    Create tests for renderer          :2d
    Add to mermaid                  :1d

    section Documentation
    Describe gantt syntax           :active, a1, after des1, 3d
    Add gantt diagram to demo page  :after a1 , 20h
    Add another diagram to demo page :doc1, after a1 , 48h

    section Last section
    Describe gantt syntax           :after doc1, 3d
    Add gantt diagram to demo page  :20h
    Add another diagram to demo page :48h
")
```

Plus d'informations : <https://rich-iannone.github.io/DiagrammeR/>

## highcharter

L'extension **highcharter** permet de réaliser des graphiques **HTML** utilisant la librairie Javascript **Highcharts.js**.

```
R> library("highcharter")
  data(diamonds, mpg, package = "ggplot2")

  hchart(mpg, "scatter", hcaes(x = displ, y = hwy, group = class))
```

```
R> library(tidyverse)
  library(highcharter)
  mpgman3 <- mpg %>%
    group_by(manufacturer) %>%
    summarise(n = n(), unique = length(unique(model))) %>%
    arrange(-n, -unique)

hchart(mpgman3, "treemap", hcaes(x = manufacturer, value = n, color = unique))
```

```
R> data(unemployment)

hcmap("countries/us/us-all-all",
      data = unemployment,
      name = "Unemployment", value = "value", joinBy = c("hc-key", "code"),
      borderColor = "transparent"
) %>%
  hc_colorAxis(dataClasses = color_classes(c(seq(0, 10, by = 2), 50))) %>%
  hc_legend(
    layout = "vertical", align = "right",
    floating = TRUE, valueDecimals = 0, valueSuffix = "%"
)
```

Plus d'informations : <http://jkunst.com/highcharter/>



# Conditions et comparaisons

Une condition est une expression logique dont le résultat est soit `TRUE` (vrai) soit `FALSE` (faux).

Une condition comprend la plupart du temps un opérateur de comparaison. Les plus courants sont les suivants :

Opérateur de comparaison	Signification
<code>==</code>	égal à
<code>!=</code>	différent de
<code>&gt;</code>	strictement supérieur à
<code>&lt;</code>	strictement inférieur à
<code>&gt;=</code>	supérieur ou égal à
<code>&lt;=</code>	inférieur ou égal à

Voyons tout de suite un exemple :

```
R> library(questionr)
data(hdv2003)
d <- hdv2003
str(d$sexe == "Homme")
```

```
logi [1:2000] FALSE FALSE TRUE TRUE FALSE FALSE ...
```

Que s'est-il passé ? Nous avons fourni à R une condition qui signifie « la valeur de la variable `sexe` vaut "Homme" ». Et il nous a renvoyé un vecteur avec autant d'éléments qu'il y'a d'observations dans `d`, et dont la valeur est `TRUE` si l'observation correspond à un homme et `FALSE` dans les autres cas.

Prenons un autre exemple. On n'affichera cette fois que les premiers éléments de notre variable d'intérêt à l'aide de la fonction `head` :

```
R> head(d$age)
```

```
[1] 28 23 59 34 71 35
```

```
R> head(d$age > 40)
```

```
[1] FALSE FALSE TRUE FALSE TRUE FALSE
```

On voit bien ici qu'à chaque élément du vecteur `d$age` dont la valeur est supérieure à 40 correspond un élément `TRUE` dans le résultat de la condition.

On peut combiner ou modifier des conditions à l'aide des opérateurs logiques habituels :

Opérateur logique	Signification
&	et logique
	ou logique
!	négation logique

Comment les utilise-t-on ? Voyons tout de suite des exemples. Supposons que je veuille déterminer quels sont dans mon échantillon les hommes ouvriers spécialisés :

```
R> d$sex == "Homme" & d$qualif == "Ouvrier specialise"
```

Si je souhaite identifier les personnes qui bricolent ou qui font la cuisine :

```
R> d$bricol == "Oui" | d$cuisine == "Oui"
```

Si je souhaite isoler les femmes qui ont entre 20 et 34 ans :

```
R> d$sex == "Femme" & d$age >= 20 & d$age <= 34
```

Si je souhaite récupérer les enquêtés qui ne sont pas cadres, on peut utiliser l'une des deux formes suivantes :

```
R> d$qualif != "Cadre"  
!(d$qualif == "Cadre")
```

Lorsqu'on mélange « et » et « ou » il est nécessaire d'utiliser des parenthèses pour différencier les blocs. La

condition suivante identifie les femmes qui sont soit cadre, soit employée :

```
R> d$sex == "Femme" & (d$qualif == "Employe" | d$qualif == "Cadre")
```

L'opérateur `%in%` peut être très utile : il teste si une valeur fait partie des éléments d'un vecteur. Ainsi on pourrait remplacer la condition précédente par :

```
R> d$sex == "Femme" & d$qualif %in% c("Employe", "Cadre")
```

Enfin, signalons qu'on peut utiliser les fonctions `table` ou `summary` pour avoir une idée du résultat de notre condition :

```
R> table(d$sex)
```

```
Homme Femme  
899 1101
```

```
R> table(d$sex == "Homme")
```

```
FALSE TRUE  
1101 899
```

```
R> summary(d$sex == "Homme")
```

```
Mode FALSE TRUE  
logical 1101 899
```



# Formules

Statistiques descriptives . . . . .	929
Tableaux croisés avec xtabs . . . . .	930
Statistiques bivariées avec aggregate . . . . .	934
Panels graphiques avec lattice . . . . .	939
Visualisation bivariée . . . . .	941
Visualisation par «petits multiples» . . . . .	941
Spécifier des modèles . . . . .	943
Les formules R . . . . .	943
Pour aller plus loin . . . . .	944

Ce chapitre vise à illustrer l'utilisation de la notation «formule» de R, qui désigne l'emploi de cette notation par l'expression `formula`. Cette notation est utilisée par de très nombreuses fonctions de R : on en a notamment vu plusieurs exemples dans le [chapitre sur les graphiques bivariés](#), car l'extension `ggplot2` se sert de cette notation dans ses paramètres `facet_wrap` et `facet_grid`.

Dans ce chapitre, on verra comment se servir de la notation «formule» dans deux contextes différents. D'une part, on verra que deux fonctions basiques de R se servent de cette notation pour produire des tableaux croisés et des statistiques bivariées. D'autre part, on verra que l'extension `lattice` se sert de cette notation pour créer des graphiques «panelisés», dits graphiques à «petits multiples».

Dans plusieurs autres chapitres, les opérations décrites ci-dessus sont effectuées avec les extensions `dplyr` d'une part, et `ggplot2` d'autre part. On se servira également de ces extensions dans ce chapitre, de manière à mener une comparaison des différentes manières d'effectuer certaines opérations dans R, avec ou sans la notation «formule» :

```
R> library(dplyr)
  library(ggplot2)
```

## Statistiques descriptives

Les premiers exemples de ce chapitre montrent l'utilisation de cette notation pour produire des tableaux

croisés et des statistiques descriptives. Le jeu de données utilisé, `hdv2003`, a déjà été utilisé dans plusieurs chapitres, et font partie de l'extension `questionr`. Chargeons cette extension et le jeu de données `hdv2003` :

```
R> library(questionr)
R> data(hdv2003)
```

Pour rappel, ce jeu de données contient des individus, leur âge, leur statut professionnel, et le nombre d'heures quotidiennes passées à regarder la télévision.

```
R> glimpse(hdv2003, 75)
```

```
Rows: 2,000
Columns: 20
$ id              <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1...
$ age             <int> 28, 23, 59, 34, 71, 35, 60, 47, 20, 28, 65, 47, ...
$ sexe            <fct> Femme, Femme, Homme, Homme, Femme, Femme, Fem...
$ nivetud         <fct> "Enseignement superieur y compris technique supe...
$ poids            <dbl> 2634.3982, 9738.3958, 3994.1025, 5731.6615, 4329...
$ occup            <fct> "Exerce une profession", "Etudiant, eleve", "Exe...
$ qualif           <fct> Employe, NA, Technicien, Technicien, Employe, Em...
$ freres.soeurs   <int> 8, 2, 2, 1, 0, 5, 1, 5, 4, 2, 3, 4, 1, 5, 2, 3, ...
$ cuso             <fct> Oui, Oui, Non, Non, Oui, Non, Oui, Non, Oui, Non...
$ relig             <fct> Ni croyance ni appartenance, Ni croyance ni appa...
$ trav.imp          <fct> Peu important, NA, Aussi important que le reste, ...
$ trav.satisf      <fct> Insatisfaction, NA, Equilibre, Satisfaction, NA, ...
$ hard.rock         <fct> Non, Non, Non, Non, Non, Non, Non, Non, Non, Non...
$ lecture.bd        <fct> Non, Non, Non, Non, Non, Non, Non, Non, Non, Non...
$ peche.chasse     <fct> Non, Non, Non, Non, Non, Non, Oui, Oui, Non, Non...
$ cuisine           <fct> Oui, Non, Non, Oui, Non, Oui, Oui, Non, Non, Non...
$ bricol            <fct> Non, Non, Non, Oui, Non, Non, Non, Oui, Non, Non...
$ cinema             <fct> Non, Oui, Non, Oui, Non, Oui, Non, Non, Oui, Oui...
$ sport              <fct> Non, Oui, Oui, Oui, Non, Oui, Non, Non, Non, Oui...
$ heures.tv          <dbl> 0.0, 1.0, 0.0, 2.0, 3.0, 2.0, 2.9, 1.0, 2.0, 2.0...
```

## Tableaux croisés avec xtabs

Utilisons, pour ce premier exemple, la variable `occup` du jeu de données `hdv2003`, qui correspond au statut professionnel des individus inclus dans l'échantillon. La fonction de base pour compter les individus par statut est la fonction `table` :

```
R> table(hdv2003$occup)
```

Exerce une profession		Chomeur
	1049	134
Etudiant, eleve		Retraite
	94	392
Retire des affaires		Au foyer
	77	171
Autre inactif		
	83	

Avec la fonction `xtabs`, le même résultat est produit à partir de la notation suivante :

```
R> xtabs(~occup, data = hdv2003)
```

occup		
Exerce une profession		Chomeur
	1049	134
Etudiant, eleve		Retraite
	94	392
Retire des affaires		Au foyer
	77	171
Autre inactif		
	83	

Le premier argument est une formule, au sens où R entend cette expression. Le second argument, `data`, correspond au jeu de données auquel la formule doit être appliquée. On pourra se passer d'écrire explicitement cet argument dans les exemples suivants.

L'avantage de la fonction `xtabs` n'est pas évident dans ce premier exemple. En réalité, cette fonction devient utile lorsque l'on souhaite construire un ou plusieurs tableau(x) croisé(s). Par exemple, pour croiser la variable `occup` avec la variable `sexe`, une solution consiste à écrire :

```
R> with(hdv2003, table(occup, sexe))
```

sexe		
occup	Homme	Femme
Exerce une profession	520	529
Chomeur	54	80
Etudiant, eleve	48	46
Retraite	208	184
Retire des affaires	39	38
Au foyer	0	171

Autre inactif	30	53
---------------	----	----

Ou alors, ce qui revient au même :

```
R> table(hdv2003$occup, hdv2003$sexe)
```

Avec `xtabs`, la même opération s'écrit de la manière suivante :

```
R> xtabs(~ occup + sexe, hdv2003)
```

occup	sexe	
	Homme	Femme
Exerce une profession	520	529
Chomeur	54	80
Etudiant, élève	48	46
Retraite	208	184
Retire des affaires	39	38
Au foyer	0	171
Autre inactif	30	53

Cette écriture est plus courte que le code équivalent dans `dplyr` :

```
R> hdv2003 %>%
  group_by(occup) %>%
  summarise(
    Homme = sum(sexe == "Homme"),
    Femme = sum(sexe == "Femme")
  )
```

```
`summarise()` ungrouping output (override with `.`groups` argument)
```

Par contre, on pourra éventuellement utiliser `count` de `dplyr`. ATTENTION : le format du résultat ne sera pas le même.

```
R> hdv2003 %>%
  group_by(occup) %>%
  count(sexe)
```

Pour un tableau croisé joliment mis en forme, on pourra avoir recours à `tbl_cross` de `gtsummary`.

```
R> library(gtsummary)
hdv2003 %>%
 tbl_cross(row = "occup", col = "sexe", percent = "row")
```

Caractéristique	sexe		Total
	Homme	Femme	
<b>occup</b>			
Exerce une profession	520 (50%)	529 (50%)	1 049 (100%)
Chomeur	54 (40%)	80 (60%)	134 (100%)
Etudiant, eleve	48 (51%)	46 (49%)	94 (100%)
Retraite	208 (53%)	184 (47%)	392 (100%)
Retire des affaires	39 (51%)	38 (49%)	77 (100%)
Au foyer	0 (0%)	171 (100%)	171 (100%)
Autre inactif	30 (36%)	53 (64%)	83 (100%)
<b>Total</b>	<b>899 (45%)</b>	<b>1 101 (55%)</b>	<b>2 000 (100%)</b>

De plus, `xtabs` permet de créer plusieurs tableaux croisés en une seule formule :

```
R> xtabs(~ occup + sexe + trav.imp, hdv2003)
```

, , trav.imp = Le plus important

occup	sexe	
	Homme	Femme
Exerce une profession	13	16
Chomeur	0	0
Etudiant, eleve	0	0
Retraite	0	0
Retire des affaires	0	0
Au foyer	0	0
Autre inactif	0	0

, , trav.imp = Aussi important que le reste

occup	sexe	
	Homme	Femme
Exerce une profession	159	100

Chomeur	0	0
Etudiant, eleve	0	0
Retraite	0	0
Retire des affaires	0	0
Au foyer	0	0
Autre inactif	0	0

, , trav.imp = Moins important que le reste

occup	sexe	
	Homme	Femme
Exerce une profession	328	380
Chomeur	0	0
Etudiant, eleve	0	0
Retraite	0	0
Retire des affaires	0	0
Au foyer	0	0
Autre inactif	0	0

, , trav.imp = Peu important

occup	sexe	
	Homme	Femme
Exerce une profession	20	32
Chomeur	0	0
Etudiant, eleve	0	0
Retraite	0	0
Retire des affaires	0	0
Au foyer	0	0
Autre inactif	0	0

Cet exemple permet simplement de réaliser que la variable `trav. imp`, qui contient les réponses à une question portant sur l'importance du travail, n'a été mesurée (c'est-à-dire que la question n'a été posée) qu'aux seuls individus actifs de l'échantillon.

## Statistiques bivariées avec aggregate

```
R> aggregate(heures.tv ~ sexe, mean, data = hdv2003)
```

Ici, le premier argument est à nouveau une formule. Le second argument correspond à la statistique descriptive que l'on souhaite obtenir, et le dernier argument indique le jeu de données auquel appliquer les deux autres arguments. On peut d'ailleurs obtenir le même résultat en respectant de manière plus stricte l'ordre des arguments dans la syntaxe de la fonction `aggregate` :

```
R> aggregate(heures.tv ~ sexe, hdv2003, mean)
```

Cette écriture est, à nouveau, plus compacte que le code équivalent dans `dplyr`, qui demande de spécifier le retrait des valeurs manquantes :

```
R> hdv2003 %>
  group_by(sexe) %>%
  summarise(heures.tv = mean(heures.tv, na.rm = TRUE))
```

À nouveau, on va pouvoir combiner plusieurs variables dans la formule que l'on passe à `aggregate`, ce qui va permettre d'obtenir la moyenne des heures de télévision quotidiennes par sexe et par statut professionnel :

```
R> aggregate(heures.tv ~ sexe + occup, hdv2003, mean)
```

La même opération `dplyr` :

```
R> hdv2003 %>%
  group_by(occup, sexe) %>%
  summarise(heures.tv = mean(heures.tv, na.rm = TRUE))
```

La fonction `aggregate` permet bien sûr d'utiliser une autre fonction que la moyenne, comme dans cet exemple, suivi de son équivalent avec `dplyr` :

```
R> # âge médian par sexe et statut professionnel
  aggregate(age ~ sexe + occup, hdv2003, median)
```

```
R> # code équivalent avec l'extension 'dplyr'
  hdv2003 %>%
  group_by(occup, sexe) %>%
  summarise(age = median(age, na.rm = TRUE))
```

Si, comme dans le cas de `summarise`, on souhaite passer des arguments supplémentaires à la fonction `median`, il suffit de les lister à la suite du nom de la fonction. Par exemple, on écrirait : `aggregate(age ~ sexe + occup, hdv2003, median, na.rm = TRUE)`. Ceci étant, `aggregate` utilise par défaut l'option `na.action = na.omit`, donc il est bon de se rappeler que l'on peut désactiver cette option en utilisant l'option `na.action = na.pass`, ce qui permet éventuellement de conserver des lignes vides dans le tableau de résultat.

La fonction `aggregate` permet, par ailleurs, d'obtenir des résultats à plusieurs colonnes. Dans l'exemple ci-dessus, on illustre ce principe avec la fonction `range`, qui renvoie deux résultats (la valeur minimale et la valeur maximale de la variable, qui est toujours la variable `age`), chacun présentés dans une colonne :

```
R> aggregate(age ~ sexe + occup, hdv2003, range)
```

	sexe	occup	age.1	age.2
1	Homme	Exerce une profession	18	63
2	Femme	Exerce une profession	18	67
3	Homme	Chomeur	18	63
4	Femme	Chomeur	18	63
5	Homme	Etudiant, eleve	18	34
6	Femme	Etudiant, eleve	18	35
7	Homme	Retraite	48	92
8	Femme	Retraite	41	96
9	Homme	Retire des affaires	57	91
10	Femme	Retire des affaires	57	93
11	Femme	Au foyer	22	90
12	Homme	Autre inactif	39	71
13	Femme	Autre inactif	19	97

Cette fonction ne peut pas être facilement écrite dans `dplyr` sans réécrire chacune des colonnes, ce que le bloc de code suivant illustre. On y gagne en lisibilité dans les intitulés de colonnes :

```
R> hdv2003 %>%
  group_by(occup, sexe) %>%
  summarise(
    min = min(age, na.rm = TRUE),
    max = max(age, na.rm = TRUE)
  )
```

Depuis la version 1.0.0 de `dplyr`, `summarise` accepte maintenant des fonctions pouvant renvoyer plusieurs valeurs, créant ainsi autant de lignes (voir <https://www.tidyverse.org/blog/2020/03/dplyr-1-0-0-summarise/>).

```
R> hdv2003 %>%
  group_by(occup, sexe) %>%
  summarise(age = range(age, na.rm = TRUE), type = c("min", "max"))
```

```
`summarise()` regrouping output by 'occup', 'sexe' (override with `groups` argument)
```

On pourrait de même définir sa propre fonction et la passer à `aggregate` :

```
R> f <- function(x) c(mean = mean(x, na.rm = TRUE), sd = sd(x, na.rm = TRUE))
aggregate(age ~ sexe + occup, hdv2003, f)
```

	sexe	occup	age.mean	age.sd
1	Homme	Exerce une profession	41.461538	10.438113
2	Femme	Exerce une profession	40.710775	10.203864
3	Homme	Chomeur	38.925926	13.256329
4	Femme	Chomeur	38.012500	11.648321
5	Homme	Etudiant, eleve	20.895833	2.926326
6	Femme	Etudiant, eleve	21.586957	3.249452
7	Homme	Retraite	68.418269	8.018882
8	Femme	Retraite	69.510870	8.228957
9	Homme	Retire des affaires	71.179487	7.687556
10	Femme	Retire des affaires	73.789474	7.651737
11	Femme	Au foyer	50.730994	15.458412
12	Homme	Autre inactif	54.166667	6.597196
13	Femme	Autre inactif	59.962264	14.660206

Mais on réalisera vite une des limitations de `aggregate` dans ce cas-là : le tableau retourné ne contient pas 4 colonnes, mais 3 uniquement, ce que l'on peut vérifier à l'aide de `dim` ou `str`.

```
R> str(aggregate(age ~ sexe + occup, hdv2003, f))
```

```
'data.frame': 13 obs. of 3 variables:
$ sexe : Factor w/ 2 levels "Homme","Femme": 1 2 1 2 1 2 1 2 1 2 ...
$ occup: Factor w/ 7 levels "Exerce une profession",...: 1 1 2 2 3 3 4 4 5 5 ...
$ age  : num [1:13, 1:2] 41.5 40.7 38.9 38 20.9 ...
..- attr(*, "dimnames")=List of 2
... . $ : NULL
... . $ : chr [1:2] "mean" "sd"
```

Pour ce type d'opération, dans lequel on souhaite récupérer plusieurs variables calculées afin de travailler sur ces données agrégées soit dans le cadre d'opérations numériques soit de constructions graphiques, `dplyr` ou `Hmisc` s'avèrent plus commodes. Voici un exemple avec `summarize` de l'extension `Hmisc` :

```
R> library(Hmisc, quietly = TRUE)
```

```
Attaching package: 'Hmisc'
```

```
The following object is masked from 'package:gt':
```

```
html
```

```
The following object is masked from 'package:survey':
```

```
deff
```

```
The following object is masked from 'package:ggdendro':
```

```
label
```

```
The following objects are masked from 'package:dplyr':
```

```
src, summarize
```

```
The following objects are masked from 'package:questionr':
```

```
describe, wtd.mean, wtd.table, wtd.var
```

```
The following objects are masked from 'package:plyr':
```

```
is.discrete, summarize
```

```
The following objects are masked from 'package:base':
```

```
format.pval, units
```

```
R> with(hdv2003, summarize(age, llist(sexe, occup), f))
```

Notons que **Hmisc** offre déjà une telle fonction (`smean.sd`), ce qui nous aurait épargné d'écrire notre propre fonction, `f`, et il en existe bien d'autres. Voici un exemple avec des intervalles de confiance estimés par bootstrap :

```
R> with(hdv2003, summarize(age, llist(sexe, occup), smean.cl.boot))
```

Et un exemple avec **dplyr**.

```
R> hdv2003 %>%
  group_by(sexe, occup) %>%
  summarise(
    tibble(
      age_mean = mean(age, na.rm = TRUE),
      age_sd = sd(age, na.rm = TRUE)
    )
  )
```

`summarise()` regrouping output by 'sexe' (override with `groups` argument)

Enfin, il est également possible d'utiliser plusieurs variables numériques à gauche de l'opérateur `~`. En voici une illustration :

```
R> aggregate(cbind(age, poids) ~ sexe + occup, hdv2003, mean)
```

## Panels graphiques avec lattice

Les exemples suivants montreront ensuite comment la notation «formule» peut servir à produire des graphiques par panel avec l'extension **lattice**.

```
R> library(lattice)
```

### NOTE

L'extension **lattice** présente l'avantage d'être installée par défaut avec R. Il n'est donc pas nécessaire de l'installer préalablement.

Chargeons les mêmes données que le [chapitre sur les graphiques bivariés](#).

```
R> # charger l'extension lisant le format CSV
library(readr)

# emplacement souhaité pour le jeu de données
file <- "data/debt.csv"

# télécharger le jeu de données s'il n'existe pas
if (!file.exists(file)) {
  download.file("http://www.stat.cmu.edu/~cshalizi/uADA/13/hw/11/debt.csv",
    file,
    mode = "wb"
  )
}

# charger les données dans l'objet 'debt'
debt <- read_csv(file)
```

Warning: Missing column names filled in: 'X1' [1]

```
-- Column specification -----
cols(
  X1 = col_double(),
  Country = col_character(),
  Year = col_double(),
  growth = col_double(),
  ratio = col_double()
)
```

Rejetons rapidement un coup d'oeil à ces données, qui sont structurées par pays (variable *Country*) et par année (variable *Year*). On y trouve deux variables, *growth* (le taux de croissance du produit intérieur brut réel), et *ratio* (le ratio entre la dette publique et le produit intérieur brut), ainsi qu'une première colonne vide, ne contenant que des numéros lignes, dont on va se débarrasser :

```
R> # inspection des données
glimpse(debt, 75)
```

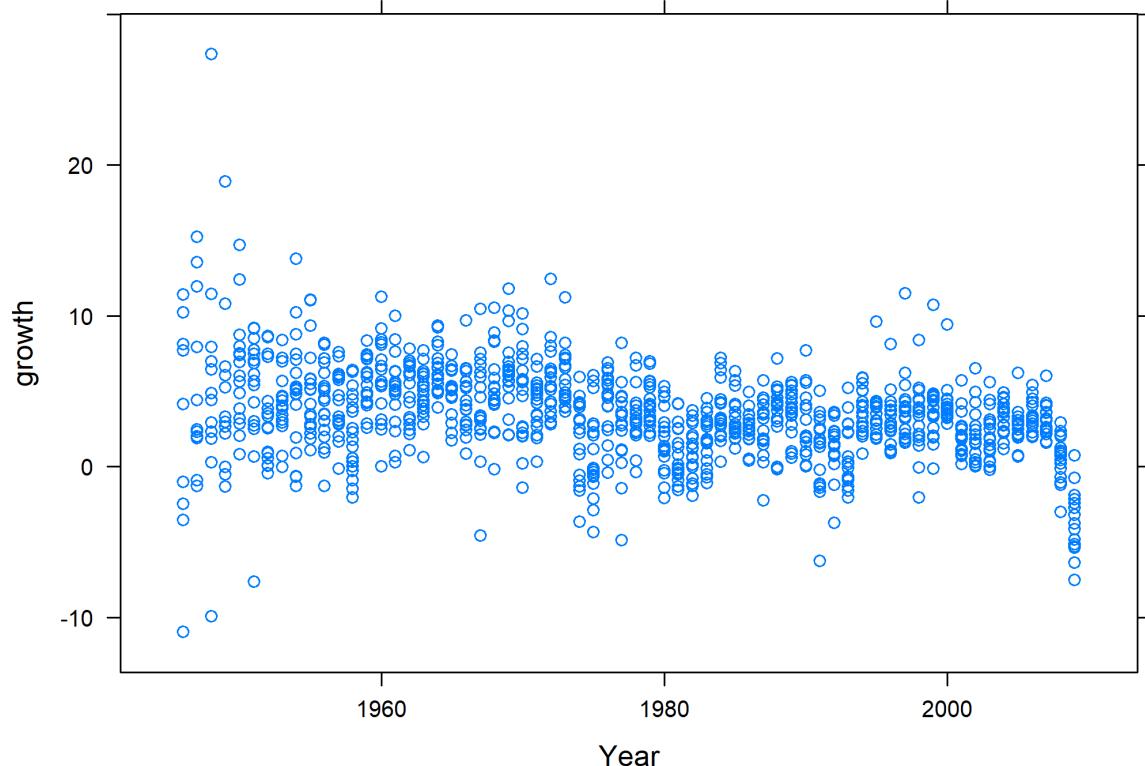
```
Rows: 1,171
Columns: 5
$ X1      <dbl> 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, ...
$ Country <chr> "Australia", "Australia", "Australia", "Australia", "A...
$ Year     <dbl> 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, ...
$ growth   <dbl> -3.5579515, 2.4594746, 6.4375341, 6.6119938, 6.9202012...
$ ratio    <dbl> 190.41908, 177.32137, 148.92981, 125.82870, 109.80940, ...
```

```
R> # suppression de la première colonne  
debt <- debt[, -1]
```

## Visualisation bivariée

Le même graphique s'écrit de la manière suivante avec l'extension **lattice** :

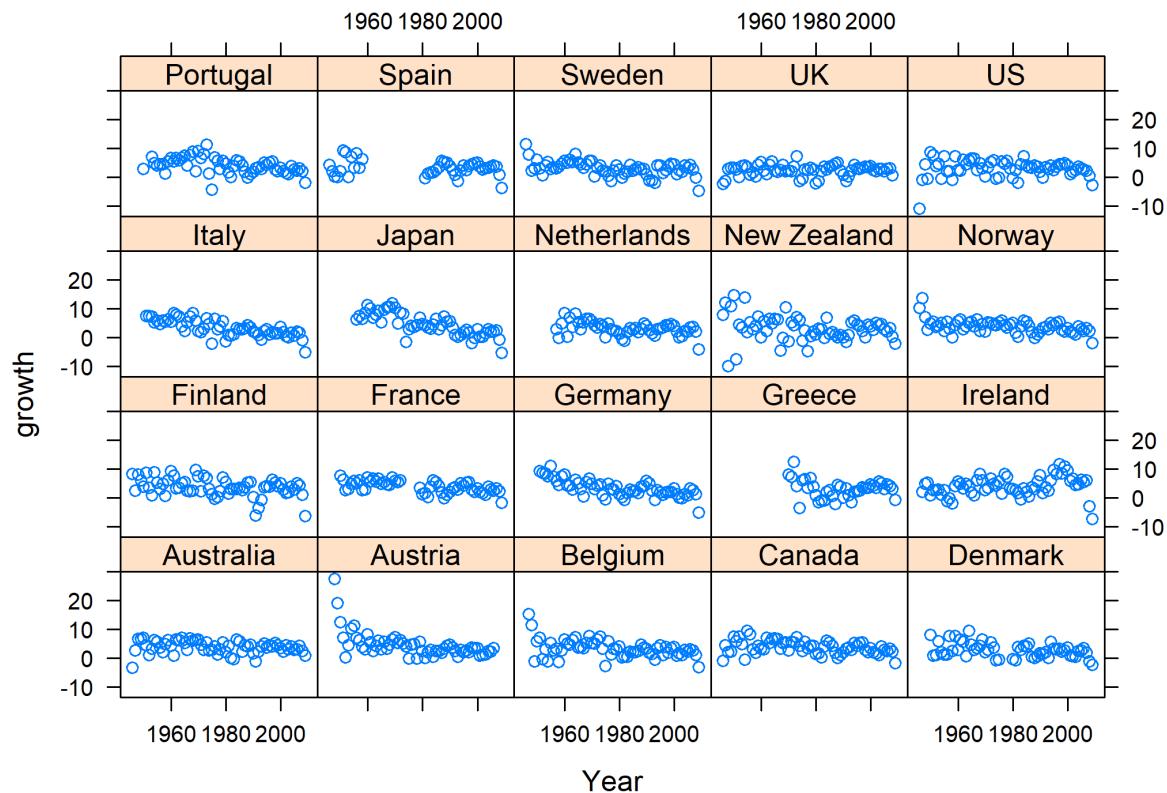
```
R> xyplot(growth ~ Year, data = debt)
```



## Visualisation par «petits multiples»

Appliquons désormais la même visualisation par «petits multiples» que vue dans le chapitre :

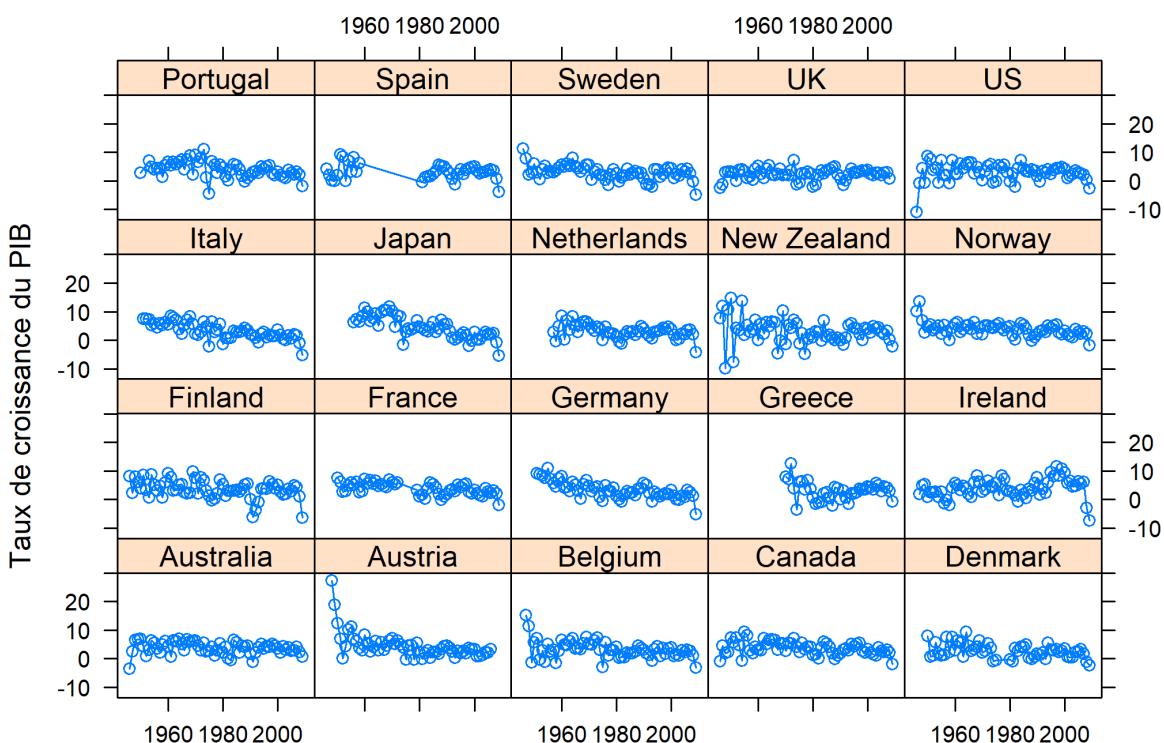
```
R> xyplot(growth ~ Year | Country, data = debt)
```



Enfin, rajoutons quelques options au graphique, afin de montrer comment l'extension **lattice** fonctionne :

```
R> xyplot(growth ~ Year | Country,
  type = c("o", "l"),
  main = "Données Reinhart et Rogoff corrigées, 1946-2009",
  ylab = "Taux de croissance du PIB",
  xlab = NULL,
  data = debt
)
```

### Données Reinhart et Rogoff corrigées, 1946-2009



## Spécifier des modèles

### Les formules R

En réalité, la notation par formule qu'utilise R est celle proposée par Wilkinson *et al.* dans les années 70 pour schématiser la relation entre plusieurs variables dans un plan d'expérience. Plus spécifiquement, l'idée revient à exprimer une relation «fonctionnelle», symbolisée par l'opérateur `~`, entre une variable réponse `y` et une ou plusieurs variables explicatives. Disons, pour simplifier, que `y` est une variable d'intérêt (numérique ou facteur selon le type de modèle), `x` une variable numérique et que `a` et `b` sont

des variables catégorielles (des facteurs dans le langage R). Voici les principales relations auxquelles on peut s'intéresser dans un modèle statistique :

- `y ~ x` : régression simple,
- `y ~ x + 0` : idem avec suppression du terme d'ordonnée à l'origine,
- `y ~ a + b` : régresse avec deux effets principaux indépendants,
- `y ~ a * b` : idem avec interaction (équivalent à `1 + a + b + a:b`),
- `y ~ a / b` : idem en considérant une relation d'emboîtement (équivalent à `1 + a + b + a %in% b`).

L'opérateur `|` est quant à lui utilisé par l'extension [lme4](#) dans le cadre de modèles mixtes avec effets aléatoires.

Voir le chapitre dédié à la régression logistique, page 507 pour des exemples de modèles multivariés et le chapitre dédié aux effets d'interaction, page 617 pour plus de détails sur cette notion.

## Pour aller plus loin

Comme vient de le voir dans ce chapitre, la notation «formule» apparaît ça et là dans les différentes fonctions de R est de ses extensions. Il est par conséquent utile d'en connaître les rudiments, et en particulier les opérateurs `~` (*tilde*) et `+`, ne serait-ce que pour pouvoir se servir des différentes fonctions présentées sur cette page. Le chapitre lattice et les formules, page 879 fournit plus de détails sur ces aspects.

La notation «formule» devient cruciale dès que l'on souhaite rédiger des modèles : la formule `y ~ x`, par exemple, qui est équivalente à la formule `y ~ 1 + x`, correspond à l'équation mathématique  $Y = a + bX$ . On trouvera de nombreux exemples d'usage de cette notation dans les chapitres consacrés, notamment, à la régression linéaire ou à la régression logistique, page 507.

De la même manière, l'opérateur `|` (*pipe*) utilisé par l'extension [lattice](#) joue aussi un rôle très important dans la rédaction de modèles multi-niveaux, où il sert à indiquer les variables à pentes ou à coefficients aléatoires. Ces modèles sont présentés dans un [chapitre dédié](#).

# Structures conditionnelles

Les boucles avec while .....	945
Les boucles avec for .....	946
Les conditions .....	947
Les instructions if ... else .....	947
La fonction switch .....	949
L'instruction repeat ... break .....	950
L'instruction next .....	951
Barre de progression .....	951

## NOTE

La version originale de ce chapitre a été écrite par Ewen Gallic dans le cadre de son support de cours d'Ewen Gallic intitulé [Logiciel R et programmation](#), chapitre 4 «Boucles et calculs vectoriels».

Il existe deux sortes de boucles dans **R**. Celles pour lesquelles les itérations continuent tant qu'une condition n'est pas invalidée (`while`), et celles pour lesquelles le nombre d'itérations est défini au moment de lancer la boucle (`for`).

Avant de présenter chacune de ces fonctions, il est nécessaire de préciser que les boucles ne sont pas le point fort de **R**. Dès que l'on souhaite appliquer une fonction à chaque élément d'un vecteur, et/ou que le résultat de chaque itération ne dépend pas de l'itération précédente, il est préférable de vectoriser les calculs (voir le chapitre sur la vectorisation, page 953).

## Les boucles avec while

Quand on souhaite répéter un calcul tant qu'une condition est satisfaite, on utilise la fonction `while`, avec la syntaxe suivante :

```
R> while (condition) {  
  instruction  
}
```

avec `condition` une valeur logique (`TRUE` ou `FALSE`), et `instruction` du code, qui peut être entouré d’accolades si on souhaite évaluer plusieurs instructions.

Le code `instruction` sera répété tant que `condition` est vrai. Prenons l’exemple d’une plante qui mesure 10 centimètres et qui va grandir de 10 % tous les ans jusqu’à atteindre 2 mètres.

```
R> taille <- 0.10  
duree <- 0  
while (taille < 2) {  
  taille <- taille * 1.1  
  duree <- duree + 1  
}  
message(glue::glue("La plante a atteint {round(taille, 1)} mètres en {duree} années."))
```

```
La plante a atteint 2.1 mètres en 32 années.
```

## Les boucles avec `for`

Quand on connaît le nombre d’itérations à l’avance, on peut utiliser la boucle `for`. La syntaxe est la suivante :

```
R> for (variable in vector) {  
  instruction  
}
```

avec `variable` le nom d’une variable locale à la boucle `for`, `vector` un vecteur à `n` éléments définissant les valeurs que prendra `variable` pour chacun des `n` tours, et `instruction` le code à exécuter à chaque itération.

On peut utiliser `for` pour remplir les éléments d’une liste, ou d’un vecteur.

```
R> # Mauvaise manière
  resultat <- NULL
  for (i in 1:3) {
    resultat[i] <- i
  }
  resultat
```

```
[1] 1 2 3
```

À chaque itération, R doit trouver le vecteur de destination en mémoire, créer un nouveau vecteur qui permettra de contenir plus de données, copier données depuis l'ancien vecteur pour les insérer dans le nouveau, et enfin supprimer l'ancien vecteur (Ross, 2014). C'est une opération coûteuse en temps. Un moyen de rendre cette allocation plus efficace est de créer a priori le vecteur ou la liste en le remplissant avec des données manquantes. Ainsi, R n'aura pas besoin de ré-allouer la mémoire à chaque itération.

```
R> # Manière plus économique
  resultat <- rep(NA, 3)
  for (i in 1:3) {
    resultat[i] <- i
  }
  resultat
```

```
[1] 1 2 3
```

## Les conditions

On peut soumettre l'exécution de codes en R à conditions que certaines conditions soient honorées.

### Les instructions if ... else

Les instructions `if` et `else` fournissent un moyen d'exécuter du code si une condition est respectée ou non. La syntaxe prend deux formes :

```
R> # Première forme (pas de code si condition == FALSE)
  if (condition) {instruction si vrai}

  # Seconde forme
  if (condition) {instruction si vrai} else {instruction si faux}
```

avec `condition` un logique, `instruction si vrai` le code à exécuter si la condition est vérifiée et `instruction si faux` le code à exécuter si la condition n'est pas remplie. À nouveau, on peut avoir recours aux accolades pour créer des regroupements.

```
R> # Simple condition
  x <- 2
  if (x == 2) print("Hello")
```

```
[1] "Hello"
```

```
R> x <- 3
  if (x == 2) print("Hello")

  # Avec des instructions dans le cas contraire
  if (x == 2) print("Hello") else print("x est différent de 2")
```

```
[1] "x est différent de 2"
```

```
R> if (x == 2) {
  print("Hello")
} else {
  x <- x - 1
  print(paste0("La nouvelle valeur de x : ", x))
}
```

```
[1] "La nouvelle valeur de x : 2"
```

### IMPORTANT

Attention, lorsque l'on fait des regroupements et qu'on utilise la structure `if` et `else`, il est nécessaire d'écrire le mot `else` sur la même ligne que la parenthèse fermante du groupe d'instructions à réaliser dans le cas où la condition du `if` est vérifiée.

## La fonction switch

Avec la fonction `switch`, on peut indiquer à R d'exécuter un code en fonction du résultat obtenu lors d'un test. La syntaxe est la suivante :

```
R> switch(valeur_test,
  cas_1 = {
    instruction_cas_1
  },
  cas_2 = {
    instruction_cas_2
  },
  ...
)
```

avec `valeur_test` un nombre ou une chaîne de caractères. Si `valeur_test` vaut `cas_1`, alors uniquement `instruction_cas_1` sera évaluée, si `valeur_test` vaut `cas_2`, alors ce sera `instruction_cas_2` qui le sera, et ainsi de suite. On peut rajouter une valeur par défaut en utilisant la syntaxe suivante :

```
R> switch(valeur_test,
  cas_1 = {
    instruction_cas_1
  },
  cas_2 = {
    instruction_cas_2
  },
  ...,
  {
    instruction_defaut
  }
)
```

Voici un exemple d'utilisation, issu de la page d'aide de la fonction.

```
R> centre <- function(x, type) {  
  switch(type,  
    mean = mean(x),  
    median = median(x),  
    trimmed = mean(x, trim = .1)  
  )  
}  
x <- rcauchy(10)  
centre(x, "mean")
```

```
[1] 2.178666
```

```
R> centre(x, "median")
```

```
[1] 0.8209592
```

```
R> centre(x, "trimmed")
```

```
[1] 1.122283
```

## L'instruction repeat ... break

L'instruction `repeat` permet de répéter une expression. Il est nécessaire d'ajouter un test d'arrêt, à l'aide de l'instruction `break`.

```
R> i <- 1  
repeat {  
  i <- i + 1  
  if (i == 3) break  
}  
i
```

```
[1] 3
```

## L'instruction next

L'instruction `next` autorise de passer immédiatement à l'itération suivante d'une boucle `for`, `while` ou `repeat`.

```
R> result <- rep(NA, 10)
  for (i in 1:10) {
    if (i == 5) next
    result[i] <- i
  }
# Le 5e élément de result n'a pas été traité
result
```

```
[1]  1  2  3  4 NA  6  7  8  9 10
```

## Barre de progression

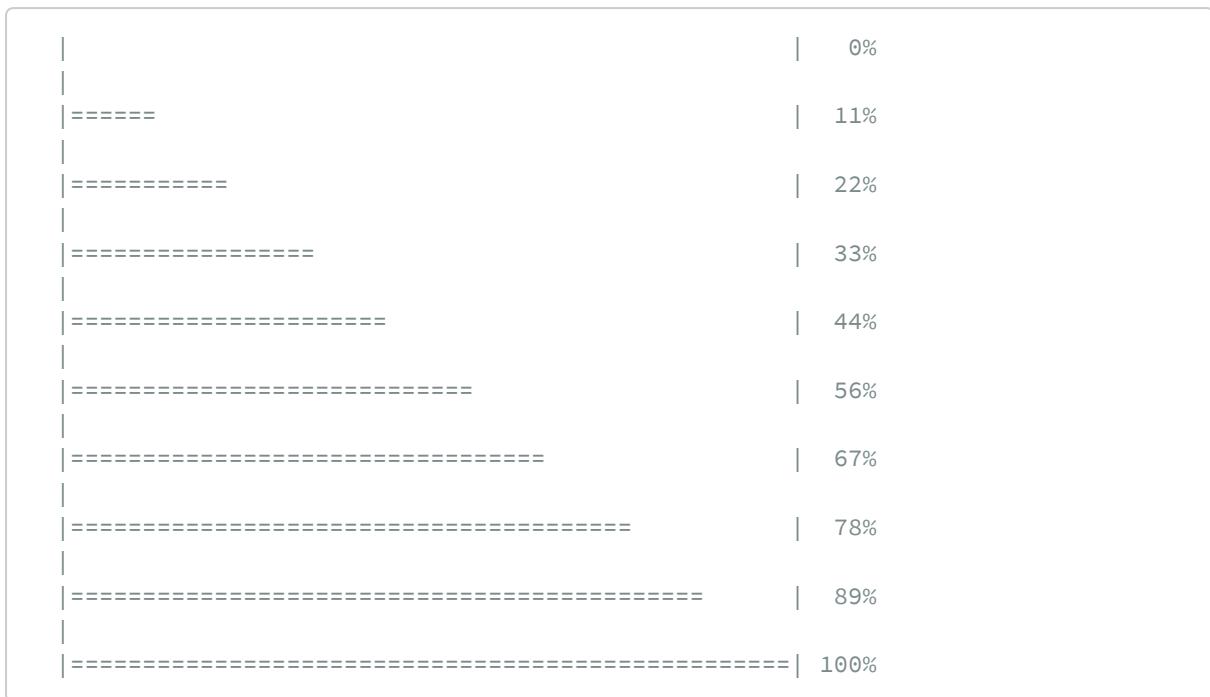
Lorsque l'exécution d'une boucle prend du temps, il peut être intéressant d'avoir une idée de l'état d'avancement des itérations. Pour cela, il est bien sûr possible d'afficher une valeur dans la console à chaque tour, chaque 10 tours, etc.

La fonction `txtProgressBar` de l'extension `utils` permet un affichage d'une barre de progression dans la console. Il suffit de lui fournir une valeur minimale et maximale, et de la mettre à jour à chaque itération. Le paramètre `style` autorise de surcroit à choisir un «style» pour la barre. Le style numéro 3 affiche un pourcentage de progression, et est utile lorsque d'autres résultats sont affichés dans la console lors de l'exécution de la boucle, dans la mesure où la barre est de nouveau affichée au complet dans la console si nécessaire.

Dans l'exemple qui suit, à chacun des dix tours, une pause de 0.1 seconde est effectuée, puis la barre de progression est mise à jour.

```
R> nb_tours <- 10
pb <- txtProgressBar(min = 1, max = nb_tours, style = 3)
for (i in 1:nb_tours) {
  Sys.sleep(0.1)
  setTxtProgressBar(pb, i)
}
```

```
|
```



Pour plus d’options, on pourra se référer à la fonction `progress_bar` de l’extension `progress`, présentée en détail sur <https://r-pkg.org/pkg/progress>.

Si l’exécution est vraiment longue, et qu’on est impatient de connaître les résultats, il existe de plus une fonction amusante dans l’extension `beepR`, qui porte le nom de `beep`. Plusieurs sons peuvent être utilisés (voir la page d’aide de la fonction).

```
R> library(beepR)
  beep("mario")
```

# Vectorisation

Les fonctions de l'extension plyr .....	954
Array en input: a*ply .....	955
Data.frame en input: d*ply .....	959
List en input: l*ply .....	963
Calcul parallèle .....	964
Les fonctions de la famille apply du package base .....	965
La fonction lapply .....	965
La fonction sapply .....	968
La fonction vapply .....	970
La fonction apply .....	971
La fonction tapply .....	972
La fonction mapply .....	973
La fonction Vectorize .....	975

## NOTE

La version originale de ce chapitre a été écrite par Ewen Gallic dans le cadre de son support de cours d'Ewen Gallic intitulé [Logiciel R et programmation](#), chapitre 4 «Boucles et calculs vectoriels».

Les boucles, page 945 sont des opérations lentes en R. Il est cependant possible, dans de nombreux cas, d'éviter de les employer, en ayant recours à la vectorisation : au lieu d'appliquer une fonction à un scalaire, on l'applique à un vecteur. En fait, nous avons déjà eu recours à maintes reprises aux calculs vectoriels. En effet, lorsque nous avons procédé à des additions, des multiplications, etc. sur des vecteurs, nous avons effectué des calculs vectoriels.

Empruntons un exemple à Burns (2011) : dans des langages comme le C, pour effectuer la somme des logarithmes naturels des n premiers entiers, voici une manière de faire :

```
R> # Somme des logarithmes des 10 premiers entiers
  somme_log <- 0
  for (i in seq_len(10)) {
    somme_log <- somme_log + log(i)
  }
  somme_log
```

```
[1] 15.10441
```

Il est possible d’obtenir le même résultat, à la fois d’une manière plus élégante, mais surtout plus efficace en vectorisant le calcul :

```
R> sum(log(seq_len(10)))
```

```
[1] 15.10441
```

Derrière ce code, la fonction `log` applique la fonction logarithme sur toutes les valeurs du vecteur donné en paramètre. La fonction `sum`, quant à elle, se charge d’additionner tous les éléments du vecteur qui lui est donné en paramètre. Ces deux fonctions utilisent la vectorisation, mais d’une manière différente : la fonction `log` applique une opération à chaque élément d’un vecteur, tandis que la fonction `sum` produit un résultat basé sur l’ensemble du vecteur. L’avantage d’utiliser des fonctions vectorielles plutôt que d’écrire une boucle pour effectuer le calcul, est que ces premières font appel à des fonctions rédigées en C ou FORTRAN, qui utilisent aussi des boucles, mais comme ce sont des langages compilés et non pas interprétés, les itérations sont réalisées dans un temps réduit.

Il existe des fonctions, rédigées en C qui effectuent des boucles `for`. On leur donne souvent le nom de “fonctions de la famille apply”. Il ne s’agit pas de la vectorisation, mais ces fonctions sont souvent mentionnées dès que l’on parle de ce sujet. Ce sont des fonctionnelles qui prennent une fonction en input et retournent un vecteur en output (Wickham, 2014). Ces fonctions sont très utilisées, mais elles souffrent d’un manque d’uniformité. En effet, elles ont été rédigées par des personnes différentes, ayant chacune leur convention. L’extension `plyr` remédie à ce problème, et ajoute par la même occasion des fonctions supplémentaires, pour couvrir plus de cas que les “fonctions de la famille apply”.

Nous allons donc présenter dans un premier temps les fonctions du package `plyr`. Les fonctions du même type du package `base` seront tout de même présentées par la suite.

## Les fonctions de l’extension `plyr`

Les fonctions que nous allons aborder dans cette section possèdent des noms faciles à se remémorer : la première lettre correspond au format d’entrée des données, la seconde au format de sortie souhaité, et la fin du nom se termine par le suffixe `ply`. Ainsi, la fonction `llply` prend en entrée une liste, effectue

une opération sur les éléments, et retourne une liste (Anderson, 2012).

Les différentes fonctions que nous allons passer en revue sont consignées dans le tableau ci-après, où les lignes correspondent aux formats d'entrée, et les lignes aux formats de sortie. Pour y avoir accès, il faut charger le package :

```
R> library(plyr)
```

		Format de sortie		
		array	data.frame	list
Format d'entrée		array	aaply	adply
data.frame		data.frame	daply	ddply
list		list	laply	ldply
				llply

Il est possible d'avoir plusieurs paramètres en input au lieu d'un seul objet. Les fonctions `mapply`, `mdply` et `maply`. Si à la place du `m`, la première lettre est un `r`, il s'agit alors de fonction de réplications. Enfin, si la seconde lettre est un trait de soulignement (`_`), alors le résultat retourné n'est pas affiché (le code utilise la fonction `invisible`).

Tous les paramètres de ces fonctions commencent par un point (`.`), afin d'éviter des incompatibilités avec la fonction à appliquer.

## Array en input : `a*ply`

Les fonctions `aaply`, `adply` et `alply` appliquent une fonction à chaque portion d'un *array* et ensuite joignent le résultat sous forme d'un *array*, d'un *data.frame* ou d'une *list* respectivement.

### NOTE

Un `array` peut être vu comme un vecteur à plusieurs dimensions. Comme pour un vecteur, toutes les valeurs doivent être du même type. Un vecteur n'est finalement qu'un *array* à une seule dimension. De même, un *array* à deux dimensions correspond à ce qu'on appelle usuellement une matrice.

Le paramètre `.margins` détermine la manière de découper le tableau. Il y en a quatre pour un tableau en deux dimensions :

1. `.margins = 1` : par lignes ;
2. `.margins = 2` : par colonnes ;

3. `.margins = c(1,2) : par cellule ;`
4. `.margins = c() : ne pas faire de découpage.`

Pour un tableau en trois dimensions, il y a trois découpages possibles en deux dimensions, trois en une dimension et une en zéro dimension (voir (Wickham, 2011)) au besoin.

```
R> tableau <- array(  
  1:24,  
  dim = c(3, 4, 2),  
  dimnames = list(  
    ligne = letters[1:3],  
    colonne = LETTERS[1:4],  
    annee = 2001:2002  
  )  
)  
tableau
```

```
, , annee = 2001
```

```
  colonne  
ligne A B C D  
a 1 4 7 10  
b 2 5 8 11  
c 3 6 9 12
```

```
, , annee = 2002
```

```
  colonne  
ligne A B C D  
a 13 16 19 22  
b 14 17 20 23  
c 15 18 21 24
```

```
R> # La moyenne des valeurs pour chaque ligne  
aapply(tableau, 1, mean) # résultat sous forme de tableau
```

```
  a      b      c  
11.5 12.5 13.5
```

```
R> adply(tableau, 1, mean) # résultat sous forme de data.frame
```

ligne	V1
a	11.5
b	12.5
c	13.5

```
R> alply(tableau, 1, mean) # résultat sous forme de liste
```

```
$`1`
[1] 11.5

$`2`
[1] 12.5

$`3`
[1] 13.5

attr(,"split_type")
[1] "array"
attr(,"split_labels")
  ligne
1     a
2     b
3     c
```

```
R> # La moyenne des valeurs pour chaque colonne
# en ne simplifiant pas le résultat
aapply(tableau, 2, mean, .drop = FALSE)
```

1	
A	8
B	11
C	14
D	17

```
R> # Par lignes et colonnes  
aapply(tableau, c(1, 2), mean)
```

	A	B	C	D
a	7	10	13	16
b	8	11	14	17
c	9	12	15	18

```
R> adply(tableau, c(1, 2), mean)
```

ligne	colonne	V1
a	A	7
b	A	8
c	A	9
a	B	10
b	B	11
c	B	12
a	C	13
b	C	14
c	C	15
a	D	16
b	D	17
c	D	18

```
R> # Avec une fonction définie par l'utilisateur
  standardise <- function(x) (x - min(x)) / (max(x) - min(x))
  # Standardiser les valeurs par colonne
  aapply(tableau, 2, standardise)
```

```
, , annee = 2001

  ligne
colonne a      b      c
A 0 0.07142857 0.1428571
B 0 0.07142857 0.1428571
C 0 0.07142857 0.1428571
D 0 0.07142857 0.1428571

, , annee = 2002

  ligne
colonne     a      b c
A 0.8571429 0.9285714 1
B 0.8571429 0.9285714 1
C 0.8571429 0.9285714 1
D 0.8571429 0.9285714 1
```

## Data.frame en input : d\*xply

Dans le cas de l'analyse d'enquêtes, on utilise principalement des tableaux de données ou *data.frame*. Aussi, la connaissance des fonctions `daply`, `ddply` et `dlply` peut être utile. En effet, elles sont très utiles pour appliquer des fonctions à des groupes basés sur des combinaisons de variables, même si dans la majorité des cas il est maintenant plus facile de passer par les extensions `dplyr` ou `data.table` avec les opérations groupées (voir la section sur `group_by` de `dplyr`, page 0 ou encore celle sur le paramètre `by` de `data.table`, page 0).

Avec les fonctions `d*xply`, il est nécessaire d'indiquer quelles variables, ou fonctions de variables on souhaite utiliser, en l'indiquant au paramètre `.variables`. Elles peuvent être contenues dans le `data.frame` fourni au paramètre `.data`, ou bien provenir de l'environnement global. R cherchera dans un premier temps si la variable est contenue dans le `data.frame` et, s'il ne trouve pas, ira chercher dans l'environnement global.

Pour indiquer que l'on désire faire le regroupement selon une variable – mettons `variable_1` – il faudra fournir l'expression `.(variable_1)` au paramètre `.variables`. Si on souhaite effectuer les regroupements selon les interactions de plusieurs variables – `variable_1`, `variable_2` et `variable_3`, il faut alors utiliser l'expression suivante : `.(variable_1, variable_2, variable_3)`.

```
R> chomage <- data.frame(
  region = rep(c(rep("Bretagne", 4), rep("Corse", 2)), 2),
  departement = rep(c(
    "Cotes-d'Armor", "Finistere",
    "Ille-et-Vilaine", "Morbihan",
    "Corse-du-Sud", "Haute-Corse"
  ), 2),
  annee = rep(c(2011, 2010), each = 6),
  ouvriers = c(
    8738, 12701, 11390, 10228, 975, 1297,
    8113, 12258, 10897, 9617, 936, 1220
  ),
  ingenieurs = c(
    1420, 2530, 3986, 2025, 259, 254,
    1334, 2401, 3776, 1979, 253, 241
  )
)
chomage
```

<b>region</b>	<b>departement</b>	<b>annee</b>	<b>ouvriers</b>	<b>ingenieurs</b>
Bretagne	Cotes-d'Armor	2011	8738	1420
Bretagne	Finistere	2011	12701	2530
Bretagne	Ille-et-Vilaine	2011	11390	3986
Bretagne	Morbihan	2011	10228	2025
Corse	Corse-du-Sud	2011	975	259
Corse	Haute-Corse	2011	1297	254
Bretagne	Cotes-d'Armor	2010	8113	1334
Bretagne	Finistere	2010	12258	2401
Bretagne	Ille-et-Vilaine	2010	10897	3776
Bretagne	Morbihan	2010	9617	1979
Corse	Corse-du-Sud	2010	936	253
Corse	Haute-Corse	2010	1220	241

```
R> # Total chomeurs en Bretagne et en Corse pour les années 2010 et 2011
# sous forme de data.frame
ddply(chomage, .(annee), summarise, total_chomeurs = sum(ouvriers + ingenieurs))
```

annee	total_chomeurs
2010	53025
2011	55803

```
R> # sous forme de array
dply(chomage, .(annee), summarise, total_chomeurs = sum(ouvriers + ingenieurs))
```

```
$`2010`
[1] 53025
```

```
$`2011`
[1] 55803
```

```
R> # sous forme de list
dply(chomage, .(annee), summarise, total_chomeurs = sum(ouvriers + ingenieurs))
```

```
$`2010`
  total_chomeurs
1      53025

$`2011`
  total_chomeurs
1      55803

attr(),"split_type")
[1] "data.frame"
attr(),"split_labels")
  annee
1  2010
2  2011
```

```
R> # Total chômeurs pour les années 2010 et 2011, par région du data frame
  ddply(chomage, .(annee, region), summarise,
        total_chomeurs = sum(ouvriers + ingenieurs)
  )
```

annee	region	total_chomeurs
2010	Bretagne	50375
2010	Corse	2650
2011	Bretagne	53018
2011	Corse	2785

```
R> # Nombre d'observations pour chaque groupe
  ddply(chomage, .(annee, region), nrow)
```

annee	region	V1
2010	Bretagne	4
2010	Corse	2
2011	Bretagne	4
2011	Corse	2

```
R> # En utilisant une fonction définie par l'utilisateur
  ddply(chomage, .(annee, region), function(x) {
    moy_ouvriers <- mean(x$ouvriers)
    moy_ingenieurs <- mean(x$ingenieurs)
    data.frame(moy_ouvriers = moy_ouvriers, moy_ingenieurs = moy_ingenieurs)
  })
```

annee	region	moy_ouvriers	moy_ingenieurs
2010	Bretagne	10221.25	2372.50
2010	Corse	1078.00	247.00
2011	Bretagne	10764.25	2490.25
2011	Corse	1136.00	256.50

## List en input : `l*ply`

Les fonctions du type `l*ply` prennent une liste en entrée. Il n'y a donc pas de paramétrage à effectuer pour choisir un découpage, il est déjà fait.

```
R> set.seed(1)
  liste <- list(normale = rnorm(10), logiques = c(TRUE, TRUE, FALSE), x = c(0, N
A, 3))
  # Obtenir la longueur de chaque élément de la liste
  laply(liste, length)
```

```
[1] 10 3 3
```

```
R> ldply(liste, length)
```

.id	V1
normale	10
logiques	3
x	3

```
R> llply(liste, length)
```

```
$normale  
[1] 10
```

```
$logiques  
[1] 3
```

```
$x  
[1] 3
```

```
R> # Calculer la moyenne pour chaque élément  
unlist(llply(liste, mean, na.rm = TRUE))
```

```
normale logiques      x  
0.1322028 0.6666667 1.5000000
```

```
R> # Appliquer une fonction définie par l'utilisateur  
llply(liste, function(x, y) x / mean(x, na.rm = TRUE) + y, y = 2)
```

```
$normale  
[1] -2.7385827 3.3891033 -4.3208096 14.0669232 4.4924421  
[6] -4.2061356 5.6869803 7.5847895 6.3552892 -0.3099997
```

```
$logiques  
[1] 3.5 3.5 2.0
```

```
$x  
[1] 2 NA 4
```

## Calcul parallèle

En utilisant plusieurs processeurs, on peut effectuer des calculs parallèles, ce qui accélère les calculs dans certains cas. En effet, quand il est possible de fractionner les opérations à effectuer en morceaux, on peut en réaliser une partie sur un processeur, une autre sur un second processeur, et ainsi de suite. Les résultats obtenus sont ensuite rassemblés avant d'être retournés. Le package **doMC** (ou **doSMP** sur **Windows**) peut être chargé pour utiliser la fonction de calcul parallèle proposé par les fonctions **\*\*ply**. Il suffit de préciser le nombre de coeurs souhaité en faisant appel à la fonction **registerDoMC**, et de fixer la valeur **TRUE** au paramètre **.parallel** de la fonction **\*\*ply**.

# Les fonctions de la famille apply du package base

Le tableau ci-après recense les fonctions principales de la famille `apply` du package `base`.

Fonction	Input	Output
<code>apply</code>	Matrice ou tableau	Vecteur ou tableau ou liste
<code>lapply</code>	Liste ou vecteur	Liste
<code>sapply</code>	Liste ou vecteur	Vecteur ou matrice ou liste
<code>vapply</code>	Liste ou vecteur	Vecteur ou matrice ou liste
<code>tapply</code>	Tableau et facteurs	Tableau ou liste
<code>mapply</code>	Listes et/ou vecteurs	Vecteur ou matrice ou liste

## La fonction lapply

La fonction `lapply` applique à chaque élément du premier paramètre qui lui est donné une fonction indiquée en second paramètre et retourne le résultat sous forme de liste. La syntaxe est la suivante :

```
R> lapply(X, FUN, ...)
```

avec `X` la liste ou le vecteur donné en paramètre sur lequel on désire appliquer la fonction `FUN`. La paramètre `...` permet de fournir des paramètres à une fonction imbriquée, en l'occurrence à celle que l'on souhaite appliquer à tous les éléments de `X`.

```
R> liste <- list(normale = rnorm(10), logiques = c(TRUE, TRUE, FALSE), x = c(0, N
A, 3))

# Obtenir la liste des longueurs de chaque élément
lapply(liste, length)
```

```
$normale
[1] 10

$logiques
[1] 3
```

```
$x  
[1] 3
```

```
R> # Calculer la moyenne pour chaque élément  
lapply(liste, mean, na.rm = TRUE)
```

```
$normale  
[1] 0.248845  
  
$logiques  
[1] 0.6666667  
  
$x  
[1] 1.5
```

On peut créer une fonction à l’intérieur de l’appel à `lapply`. Le premier paramètre est nécessairement un élément du vecteur auquel on souhaite appliquer la fonction.

```
R> lapply(liste, function(x) x / mean(x, na.rm = TRUE))
```

```
$normale  
[1] 6.07519277 1.56661087 -2.49649643 -8.89991820  
[5] 4.52060941 -0.18056868 -0.06506164 3.79286833  
[9] 3.30013177 2.38663180  
  
$logiques  
[1] 1.5 1.5 0.0  
  
$x  
[1] 0 NA 2
```

```
R> # Si la fonction doit posséder plusieurs paramètres  
lapply(liste, function(x, y) x / mean(x, na.rm = TRUE) + y, y = 2)
```

```
$normale  
[1] 8.0751928 3.5666109 -0.4964964 -6.8999182 6.5206094  
[6] 1.8194313 1.9349384 5.7928683 5.3001318 4.3866318  
  
$logiques  
[1] 3.5 3.5 2.0  
  
$x
```

```
[1] 2 NA 4
```

On peut appliquer la `lapply` sur des tableaux de données, dans la mesure où ces derniers sont des listes. Cela s'avère pratique pour réaliser des opérations pour chaque colonne d'un tableau de données. Afin de prendre moins de place dans l'affichage, l'exemple suivant utilise la fonction `unlist` pour «aplatiser» la liste obtenue.

```
R> data(cars)

# Afficher le type de chaque colonne de la data frame "cars"
unlist(lapply(cars, class))
```

```
speed      dist
"numeric" "numeric"
```

```
R> # Calculer la moyenne pour chaque colonne
  unlist(lapply(cars, mean))
```

```
speed  dist
15.40 42.98
```

**NOTE**

Attention, ce qui suit relève plus d'un tour de passe-passe que de la programmation élégante.

Si la fonction que l'on souhaite appliquer aux éléments de notre vecteur retourne un vecteur ligne de même longueur pour chaque élément, la fonction `do.call` peut devenir un outil très pratique pour créer une data frame. Voyons-le à travers un exemple.

```
R> l <- lapply(1:3, function(x) cbind(valeur = x, lettre = LETTERS[x]))
```

```
[[1]]
  valeur lettre
[1,] "1"    "A"

[[2]]
  valeur lettre
[1,] "2"    "B"

[[3]]
  valeur lettre
[1,] "3"    "C"
```

```
R> data.frame(do.call("rbind", l))
```

valeur	lettre
1	A
2	B
3	C

L'appel de `do.call("rbind", x)` revient à faire `rbind(x[[1]], x[[2]], ..., x[[n]])` avec `x` une liste de taille `n`.

## La fonction `sapply`

La fonction `sapply` applique une fonction aux éléments d'un vecteur ou d'une liste et peut retourner un vecteur, une liste ou une matrice. Elle possède la syntaxe suivante :

```
R> sapply(X, FUN, simplify, USE.NAMES)
```

où `X` est le vecteur ou la liste auquel on souhaite appliquer la fonction `FUN`. Lorsque `simplify` vaut `FALSE`, le résultat est retourné sous forme de liste, exactement comme `lapply` (la fonction `sapply` s'appuie sur la fonction `lapply`). Lorsque `simplify` vaut `TRUE` (par défaut), le résultat est retourné dans une forme simplifiée, si cela est possible. Si tous les éléments retournés par la fonction `FUN` sont des scalaires, alors `sapply` retourne un vecteur ; sinon, si les éléments retournés ont la même taille, `sapply` retourne une matrice avec une colonne pour chaque élément de `X` auquel la fonction `FUN` est appliquée. Le paramètre `USE.NAMES`, quand il vaut `TRUE` (par défaut), et si `X` est de type `character`, utilise `X` comme nom pour le résultat, à moins que le résultat possède déjà des noms.

```
R> (x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE, FALSE, FALSE, TRUE)))
```

```
$a
[1] 1 2 3 4 5 6 7 8 9 10

$beta
[1] 0.04978707 0.13533528 0.36787944 1.00000000
[5] 2.71828183 7.38905610 20.08553692

$logic
[1] TRUE FALSE FALSE TRUE
```

```
R> # Application de la fonction quantile() à chaque élément
# pour obtenir la médiane et les quartiles
# Avec lapply()
lapply(x, quantile)
```

```
$a
 0%   25%   50%   75% 100%
1.00  3.25  5.50  7.75 10.00

$beta
      0%        25%        50%        75%       100%
0.04978707 0.25160736 1.00000000 5.05366896 20.08553692

$logic
 0%   25%   50%   75% 100%
0.0  0.0  0.5  1.0  1.0
```

```
R> # Avec sapply  
sapply(x, quantile)
```

	a	beta	logic
0%	1.00	0.0497871	0.0
25%	3.25	0.2516074	0.0
50%	5.50	1.0000000	0.5
75%	7.75	5.0536690	1.0
100%	10.00	20.0855369	1.0

```
R> # Exemple avec USE.NAMES  
sapply(LETTERS[1:3], nchar)
```

```
A B C  
1 1 1
```

```
R> sapply(LETTERS[1:3], nchar, USE.NAMES = FALSE)
```

```
[1] 1 1 1
```

## La fonction vapply

La fonction `vapply` est similaire à `sapply`, mais elle possède un type de valeurs spécifié, ce qui peut rendre l’utilisation plus sûre (et parfois plus rapide). Lorsqu’on lui fournit un `data.frame`, `vapply` retourne le même résultat que `sapply`. Cependant, quand on lui fournit une liste vide, `vapply` retourne un vecteur logique de longueur nulle (ce qui est plus sensé que la liste vide que retourne `sapply`).

```
R> vapply(X, FUN, FUN.VALUE, ..., USE.NAMES)
```

avec `X`, `FUN`, `...` et `USE.NAMES` les mêmes paramètres que pour `sapply`. Le paramètre `FUN.VALUE` doit être un vecteur, un masque pour la valeur retornée par la fonction de `FUN`.

```
R> # Retourner le vecteur
  sapply(cars, is.numeric)
```

```
speed  dist
TRUE   TRUE
```

```
R> vapply(cars, is.numeric, FUN.VALUE = logical(1))
```

```
speed  dist
TRUE   TRUE
```

```
R> # Avec la liste vide
  sapply(list(), is.numeric)
```

```
list()
```

```
R> vapply(list(), is.numeric, FUN.VALUE = logical(1))
```

```
logical(0)
```

## La fonction apply

La fonction `apply` possède la syntaxe suivante :

```
R> apply(X, MARGIN, FUN, ...)
```

avec `X` une matrice ou un tableau, `MARGIN` indiquant si on souhaite appliquer la fonction `FUN` aux lignes (`MARGIN = 1`) ou aux colonnes (`MARGIN = 2`), et `...` des paramètres supplémentaires éventuels à passer à la fonction `FUN`.

```
R> (X <- matrix(1:9, ncol = 3))
```

1	4	7
2	5	8
3	6	9

```
R> # Somme par ligne  
apply(X, MARGIN = 1, sum)
```

```
[1] 12 15 18
```

```
R> # Somme par colonne  
apply(X, MARGIN = 2, sum)
```

```
[1] 6 15 24
```

```
R> # Fonction définie par l'utilisateur  
apply(X, MARGIN = 1, function(x) sum(x) / sum(X))
```

```
[1] 0.2666667 0.3333333 0.4000000
```

## La fonction tapply

La fonction `tapply` s'applique à chaque cellule d'un tableau, sur des regroupements définis par les variables catégorielles fournies. La syntaxe est la suivante :

```
R> tapply(X, INDEX, FUN, ..., simplify)
```

avec `X` le tableau de données, `INDEX` une liste d'un ou plusieurs facteurs, chacun de même taille que `X`. Le paramètre `FUN` renseigne la fonction que l'on souhaite appliquer. Si `simplify` vaut `FALSE`, le résultat est un tableau de mode *list*. Sinon (par défaut), le résultat est un tableau de scalaires.

```
R> data(iris)
head(iris)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

```
R> # Moyenne de la longueur des sépales par espèce
tapply(iris$Sepal.Length, iris$Species, mean)
```

```
setosa versicolor virginica
5.006      5.936      6.588
```

```
R> # Pour retourner le résultat sous forme de liste
tapply(iris$Sepal.Length, iris$Species, mean, simplify = FALSE)
```

```
$setosa
[1] 5.006

$versicolor
[1] 5.936

$virginica
[1] 6.588
```

## La fonction mapply

La fonction `mapply` applique une fonction à plusieurs listes ou vecteurs. La syntaxe est la suivante :

```
R> mapply(FUN, ..., MoreArgs, SIMPLIFY, USE.NAMES)
```

avec `FUN` la fonction à appliquer aux vecteurs ou listes fournies (grâce à `...`), `MoreArgs` une liste de paramètres supplémentaires à fournir à la fonction à appliquer. Les paramètres `SIMPLIFY` et `USE.NAMES` ont le même usage que pour la fonction `sapply`.

```
R> (l1 <- list(a = c(1:5), b = c(6:10)))
```

```
$a  
[1] 1 2 3 4 5  
  
$b  
[1] 6 7 8 9 10
```

```
R> (l2 <- list(c = c(11:15), d = c(16:20)))
```

```
$c  
[1] 11 12 13 14 15  
  
$d  
[1] 16 17 18 19 20
```

```
R> # La somme des éléments correspondants de l1 et l2  
mapply(sum, l1$a, l1$b, l2$c, l2$d)
```

```
[1] 34 38 42 46 50
```

```
R> # Attention au recyclage silencieux !  
(l1 <- list(a = c(1:5), b = c(6:20)))
```

```
$a  
[1] 1 2 3 4 5  
  
$b  
[1] 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

```
R> mapply(sum, l1$a, l1$b, l2$c, l2$d)
```

```
[1] 34 38 42 46 50 39 43 47 51 55 44 48 52 56 60
```

## La fonction Vectorize

La fonction `Vectorize` permet de convertir une fonction scalaire en une fonction vectorielle. Attention, cela ne permet pas d'améliorer la rapidité d'exécution du code. Par contre, son utilisation assez intuitive permet de gagner du temps. Il s'agit donc de faire l'arbitrage entre le temps passé à trouver un moyen élégant et efficace pour effectuer une opération en passant par de réels calculs vectoriels et le gain d'exécution que ce calcul vectoriel apporte vis-à-vis d'une boucle. La syntaxe de la `Vectorize` est la suivante :

```
R> Vectorize(FUN, vectorize.args, SIMPLIFY, USE.NAMES)
```

avec `FUN` une fonction à appliquer, `vectorize.args` un vecteur de paramètres (de type caractère) qui devraient être vectorisés (par défaut, tous les paramètres de `FUN`). Les paramètres `SIMPLIFY` et `USE.NAMES` ont le même emploi que dans la fonction `sapply`.

```
R> f <- function(x = 1:3, y) c(x, y)
# On "vectorise" la fonction f
vf <- Vectorize(f, SIMPLIFY = FALSE)
f(1:3, 1:3)
```

```
[1] 1 2 3 1 2 3
```

```
R> vf(1:3, 1:3)
```

```
[[1]]
[1] 1 1
```

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

```
[[3]]
[1] 3 3
```

```
R> # Vectorise seulement y, pas x  
vf(y = 1:3)
```

```
[[1]]  
[1] 1 2 3 1
```

```
[[2]]  
[1] 1 2 3 2
```

```
[[3]]  
[1] 1 2 3 3
```

# Expressions régulières

Les expressions régulières sont un outils pour rechercher / remplacer dans des chaînes de texte. Il est préférable d'avoir lu au préalable le chapitre dédié à la manipulation de texte, page 289.

Pour une introduction (en anglais) aux expressions régulières, on pourra se référer au chapitre «Strings» de l'ouvrage *R for Data Science* de Garrett Grolemund et Hadley Wickham (<http://r4ds.had.co.nz/strings.html>).

Pour aller plus loin, le site de l'extension `stringr` propose une présentation détaillée (en anglais) de la syntaxe des expressions régulières (<http://stringr.tidyverse.org/articles/regular-expressions.html>).

Pour des besoins plus pointus, on pourra aussi utiliser l'extension `stringi` sur laquelle est elle-même basée `stringr`.



# R Markdown : les rapports automatisés

Rapport rapide à partir d'un script R . . . . .	980
Un premier exemple . . . . .	981
Créer un nouveau document . . . . .	986
Éléments d'un document R Markdown . . . . .	987
En-tête (préambule) . . . . .	987
Texte du document . . . . .	988
Blocs de code R . . . . .	989
Compiler un document ( <i>Knit</i> ) . . . . .	991
Personnaliser le document généré . . . . .	992
Options des blocs de code R . . . . .	994
Nom du bloc . . . . .	995
Options . . . . .	995
Modifier les options . . . . .	996
Options globales . . . . .	997
Mise en cache des résultats . . . . .	998
Rendu des tableaux . . . . .	999
Tableaux croisés . . . . .	999
Tableaux de données et tris à plat . . . . .	1001
Autres extensions pour présenter des tableaux . . . . .	1002
Éditeur avancé . . . . .	1010
Modèles de documents . . . . .	1011
Slides . . . . .	1011
Templates . . . . .	1013
Ressources . . . . .	1017

**NOTE**

La version originale de ce chapitre a été écrite par Julien Barnier dans le cadre de son [Introduction à R et au tidyverse](#).

L’extension **rmarkdown** permet de générer des documents de manière dynamique en mélangeant texte mis en forme et résultats produits par du code **R**. Les documents générés peuvent être au format **HTML**, **PDF**, **Word**, et bien d’autres<sup>1</sup>. C’est donc un outil très pratique pour l’exportation, la communication et la diffusion de résultats d’analyse.

Le présent document a lui-même été généré à partir de fichiers **R Markdown**.

**rmarkdown** ne fait pas partie du **tidyverse**, mais elle est installée et chargée par défaut par **RStudio**<sup>2</sup>.

## Rapport rapide à partir d’un script R

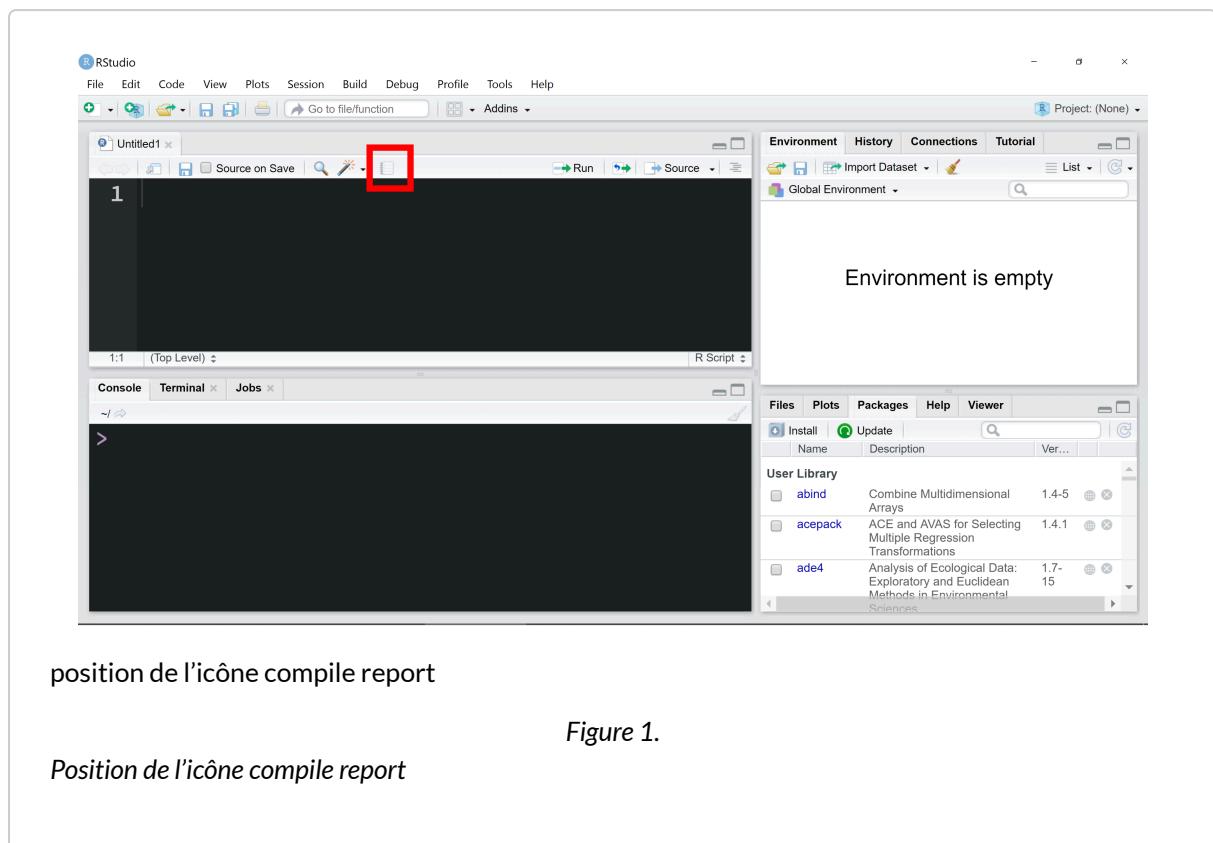
Si vos analyses sont présentes dans un script **R** et que ce script contient tout le nécessaire pour la réalisation de votre analyse (i.e. chargement des données et des packages requis), vous pouvez très facilement réaliser un rapport rapide au format **HTML**, **Word** ou **PDF**, contenant à la fois votre code et les sorties associées.

Il suffit de cliquer sur l’icône *compile report* dans le quadrant supérieur gauche.

---

1. On peut citer les formats **odt**, **rtf**, **Markdown**, etc.

2. Si vous n’utilisez pas ce dernier, l’extension peut être installée à part avec `install.packages("rmarkdown")` et chargée explicitement avec `library(rmarkdown)`.



position de l'icône compile report

*Figure 1.*

*Position de l'icône compile report*

De manière alternative, si votre script s'intitule `analyses.R`, vous pouvez exécuter la commande suivante :

```
R> rmarkdown::render("analyses.R")
```

Pour plus d'informations sur les options disponibles, vous pouvez également consulter [https://rmarkdown.rstudio.com/articles\\_report\\_from\\_r\\_script.html](https://rmarkdown.rstudio.com/articles_report_from_r_script.html).

## Un premier exemple

Voici un exemple de document **R Markdown** minimal :

```
---
title: "Test R Markdown"
---

*R Markdown* permet de mélanger :
```

- du texte libre mis en forme
- des blocs de code R

Les blocs de code sont exécutés et leur résultat affiché, par exemple :

```
```{r}
mean(mtcars$mpg)
```

## Graphiques
```

On peut également inclure des graphiques :

```
```{r}
plot(mtcars$hp, mtcars$mpg)
```
```

Ce document peut être «compilé» sous différents formats. Lors de cette étape, le texte est mis en forme, les blocs de code sont exécutés, leur résultat ajouté au document, et le tout est transformé dans un des différents formats possibles.

Voici le rendu du document précédent au format **HTML** :

# Test R Markdown

R Markdown permet de mélanger :

- du texte libre mis en forme
- des blocs de code R

Les blocs de code sont exécutés et leur résultat affiché, par exemple :

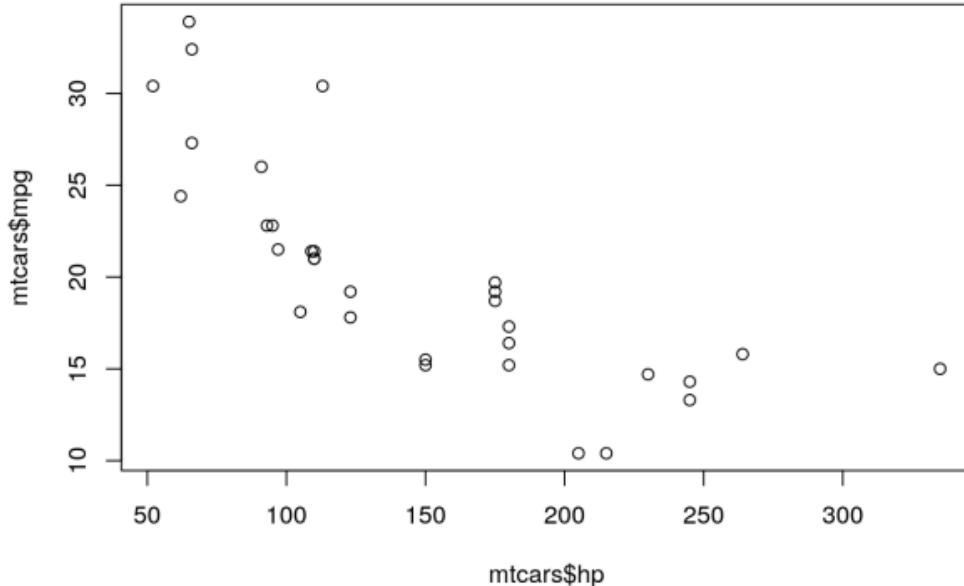
```
mean(mtcars$mpg)
```

```
## [1] 20.09062
```

## Graphiques

On peut également inclure des graphiques :

```
plot(mtcars$hp, mtcars$mpg)
```



Rendu HTML

Figure 2.

Rendu HTML

Le rendu du même document au format **PDF** :

## Test R Markdown

*R Markdown* permet de mélanger :

- du texte libre mis en forme
- des blocs de code R

Les blocs de code sont exécutés et leur résultat affiché, par exemple :

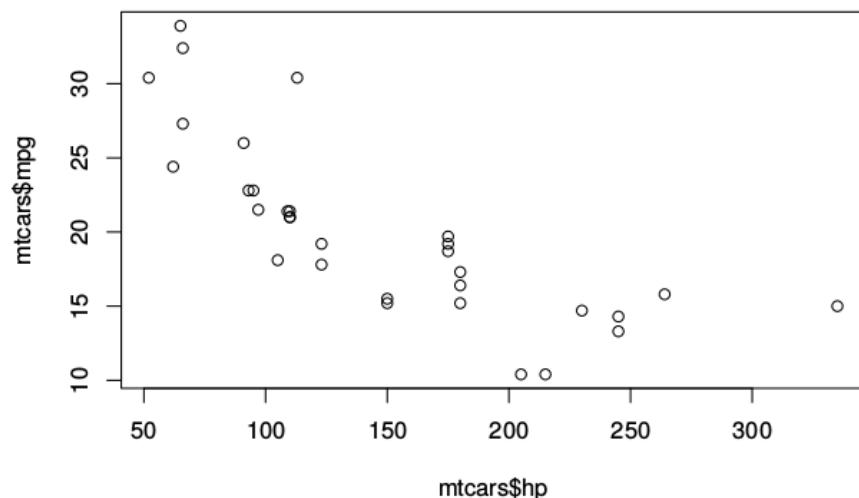
```
mean(mtcars$mpg)
```

```
## [1] 20.09062
```

### Graphiques

On peut également inclure des graphiques :

```
plot(mtcars$hp, mtcars$mpg)
```



Rendu PDF

*Figure 3.*

Rendu PDF

Et le rendu au format **Word** :

## Test R Markdown

R Markdown permet de mélanger :

- du texte libre mis en forme
- des blocs de code R

Les blocs de code sont exécutés et leur résultat affiché, par exemple :

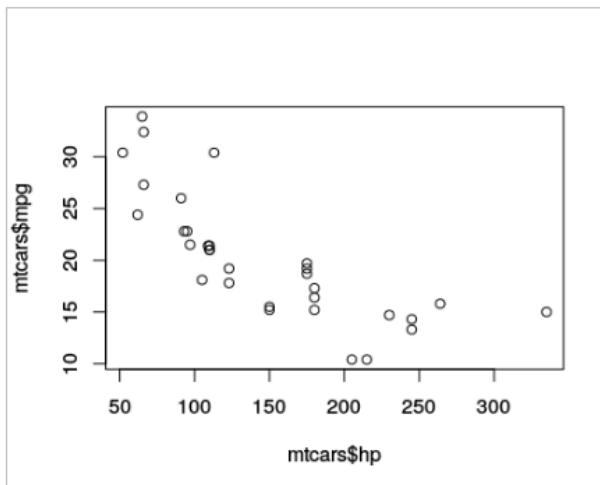
```
mean(mtcars$mpg)
```

```
## [1] 20.09062
```

### Graphiques

On peut également inclure des graphiques :

```
plot(mtcars$hp, mtcars$mpg)
```



Rendu docx

Figure 4.

Rendu docx

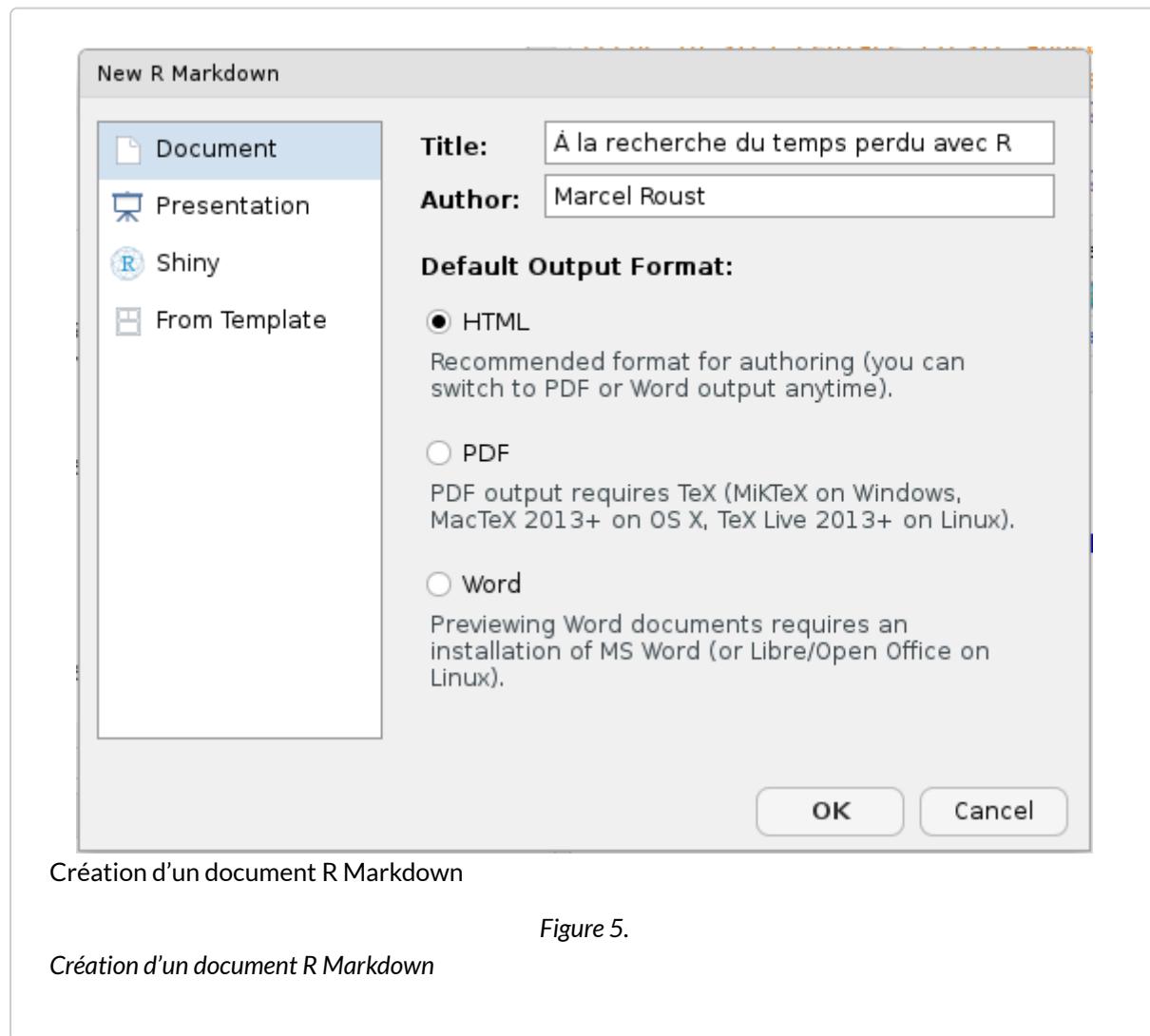
Les avantages de ce système sont nombreux :

- le code et ses résultats ne sont pas séparés des commentaires qui leur sont associés
- le document final est reproductible
- le document peut être très facilement régénéré et mis à jour, par exemple si les données source ont été modifiées.

## Créer un nouveau document

Un document **R Markdown** est un simple fichier texte enregistré avec l’extension `.Rmd`.

Sous **RStudio**, on peut créer un nouveau document en allant dans le menu *File* puis en choisissant *New file* puis *R Markdown*.... La boîte de dialogue suivante s’affiche :



On peut indiquer le titre, l'auteur du document ainsi que le format de sortie par défaut (il est possible de modifier facilement ses éléments par la suite). Plutôt qu'un document classique, on verra plus loin qu'on peut aussi choisir de créer une présentation sous forme de slides (entrée *Presentation*) ou de créer un document à partir d'un modèle (Entrée *From Template*).

Un fichier comportant un contenu d'exemple s'affiche alors. Vous pouvez l'enregistrer où vous le souhaitez

avec une extension `.Rmd`.

## Éléments d'un document R Markdown

Un document **R Markdown** est donc un fichier texte qui ressemble à quelque chose comme ça :

```
---
```

```
title: "Titre"
author: "Prénom Nom"
date: "10 avril 2017"
output: html_document
```

```
---
```

```
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## Introduction
```

Ceci est un document RMarkdown, qui mélange :

- du texte balisé selon la syntaxe Markdown
- des bouts de code R qui seront exécutés

Le code R se présente de la manière suivante :

```
```{r}
summary(cars)
```

## Graphiques
```

On peut aussi inclure des graphiques, par exemple :

```
```{r}
plot(pressure)
```
```

## En-tête (préambule)

La première partie du document est son *en-tête*. Il se situe en tout début de document, et est délimité par trois tirets (---) avant et après :

```
---
```

```
title: "Titre"
author: "Prénom Nom"
date: "10 avril 2017"
output: html_document
---
```

Cet en-tête contient les métadonnées du document, comme son titre, son auteur, sa date, plus tout un tas d'options possibles qui vont permettre de configurer ou personnaliser l'ensemble du document et son rendu. Ici, par exemple, la ligne `output: html_document` indique que le document généré doit être au format HTML.

## Texte du document

Le corps du document est constitué de texte qui suit la syntaxe **Markdown**. Un fichier Markdown est un fichier texte contenant un balisage léger qui permet de définir des niveaux de titres ou de mettre en forme le texte. Par exemple, le texte suivant :

```
Ceci est du texte avec *de l'italique* et **du gras**.
```

```
On peut définir des listes à puces :
```

- premier élément
- deuxième élément

Génèrera le texte mis en forme suivant :

*Ceci est du texte avec de l'italique et du gras.*

*On peut définir des listes à puces :*

- premier élément
- deuxième élément

On voit que des mots placés entre des astérisques sont mis en italique, des lignes qui commencent par un tiret sont transformés en liste à puce, etc.

On peut définir des titres de différents niveaux en faisant débuter une ligne par un ou plusieurs caractères `#` :

```
# Titre de niveau 1
```

```
## Titre de niveau 2

### Titre de niveau 3
```

Quand des titres ont été définis, si vous cliquez sur l'icône *Show document outline* totalement à droite de la barre d'outils associée au fichier R Markdown, une table des matières dynamique générée automatiquement à partir des titres présents dans le document s'affiche et vous permet de naviguer facilement dans celui-ci :

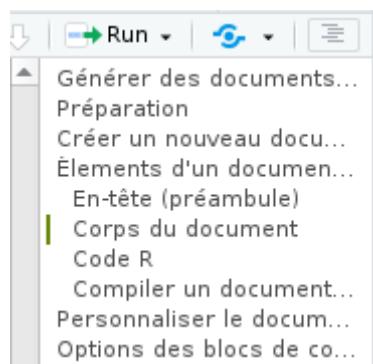


Table des matières dynamique

*Figure 6.*

*Table des matières dynamique*

La syntaxe **Markdown** permet d'autres mises en forme, comme la possibilité d'insérer des liens ou des images. Par exemple, le code suivant :

```
[Exemple de lien] (https://example.com)
```

Donnera le lien suivant :

*Exemple de lien*

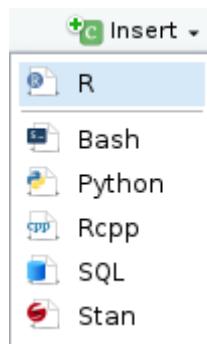
Dans **RStudio**, le menu *Help* puis *Markdown quick reference* donne un aperçu plus complet de la syntaxe.

## Blocs de code R

En plus du texte libre au format Markdown, un document **R Markdown** contient, comme son nom l'indique, du code **R**. Celui-ci est inclus dans des blocs (*chunks*) délimités par la syntaxe suivante :

```
```{r}
x <- 1:5
```
```

Comme cette suite de caractères n'est pas très simple à saisir, vous pouvez utiliser le menu *Insert* de **RStudio** et choisir **R**<sup>3</sup>, ou utiliser le raccourci clavier **Ctrl+Alt+i**.



Menu d'insertion d'un bloc de code

Figure 7.

Menu d'insertion d'un bloc de code

Dans **RStudio** les blocs de code **R** sont en général affichés avec une couleur de fond légèrement différente pour les distinguer du reste du document.

Quand votre curseur se trouve dans un bloc, vous pouvez saisir le code **R** que vous souhaitez, l'exécuter, utiliser l'autocomplétion, exactement comme si vous vous trouviez dans un script **R**. Vous pouvez également exécuter l'ensemble du code contenu dans un bloc à l'aide du raccourci clavier **Ctrl+Shift+Entrée**.

Dans **RStudio**, par défaut, les résultats d'un bloc de code (texte, tableau ou graphique) s'affichent directement *dans la fenêtre d'édition du document*, permettant de les visualiser facilement et de les conserver le temps de la session<sup>4</sup>.

Lorsque le document est «compilé» au format **HTML**, **PDF** ou **docx**, chaque bloc est exécuté tour à tour, et le résultat inclus dans le document final, qu'il s'agisse de texte, d'un tableau ou d'un graphique. Les blocs sont liés entre eux, dans le sens où les données importées ou calculées dans un bloc sont accessibles aux blocs suivants. On peut donc aussi voir un document **R Markdown** comme un script **R** dans lequel on aurait intercalé du texte libre au format **Markdown**.

3. Il est possible d'inclure dans un document **R Markdown** des blocs de code d'autres langages

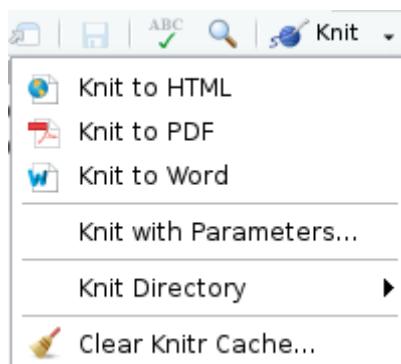
4. Ce comportement peut être modifié en cliquant sur l'icône d'engrenage de la barre d'outils et en choisissant **Chunk Output in Console**

**NOTE**

À noter qu'avant chaque compilation, une nouvelle session R est lancée, ne contenant aucun objet. Les premiers blocs de code d'un document sont donc souvent utilisés pour importer des données, exécuter des recodages, etc.

## Compiler un document (*Knit*)

On peut à tout moment compiler, ou plutôt «tricoter» (*Knit*), un document R Markdown pour obtenir et visualiser le document généré. Pour cela, il suffit de cliquer sur le bouton *Knit* et de choisir le format de sortie voulu :



Menu Knit

Figure 8.

Menu Knit

Vous pouvez aussi utiliser le raccourci `Ctrl+Shift+K` pour compiler le document dans le dernier format utilisé.

**IMPORTANT**

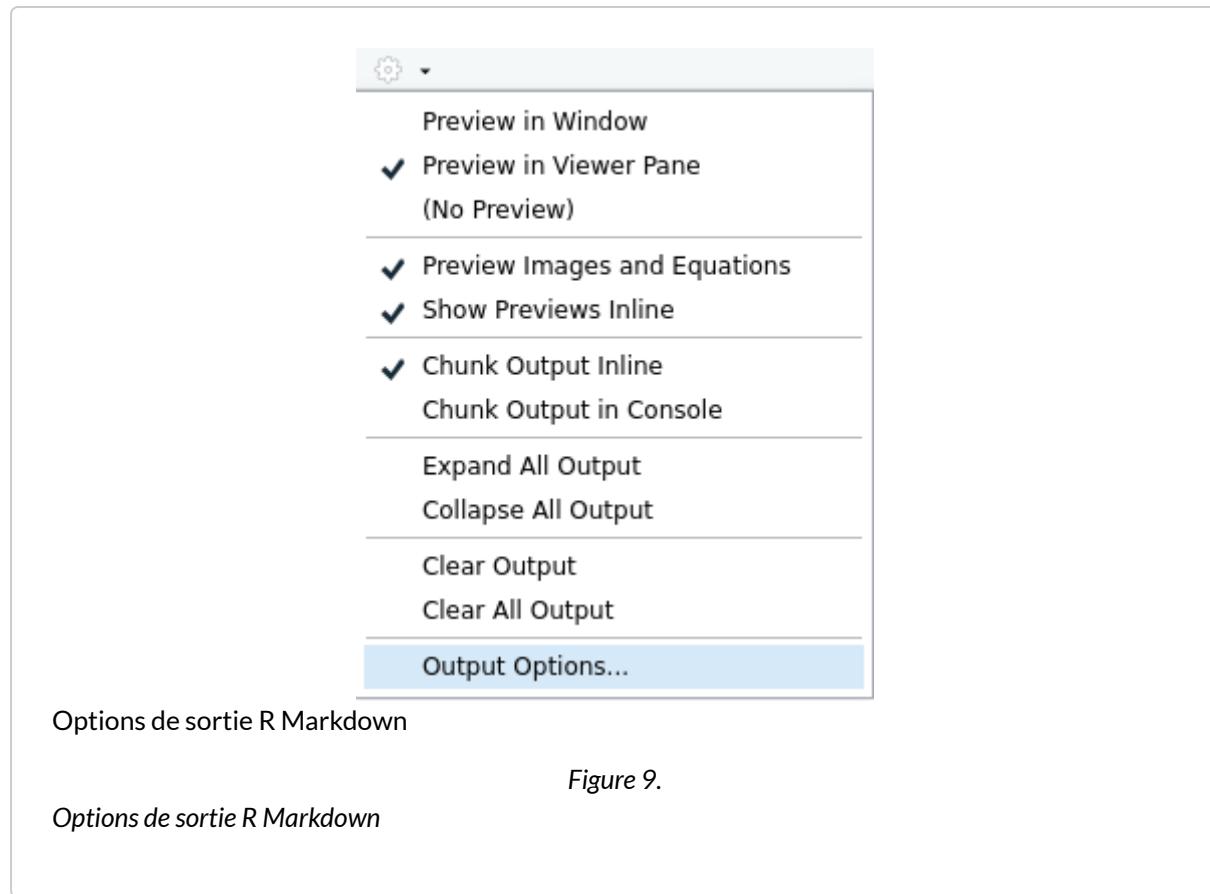
Pour la génération du format PDF, vous devez avoir une installation fonctionnelle de `LaTeX` sur votre système. C'est en général le cas pour des ordinateurs Mac ou Linux, mais pas sous Windows : dans ce cas vous devrez installer une distribution comme `MiKTeX`.

Un onglet *R Markdown* s'ouvre dans la même zone que l'onglet *Console* et indique la progression de la

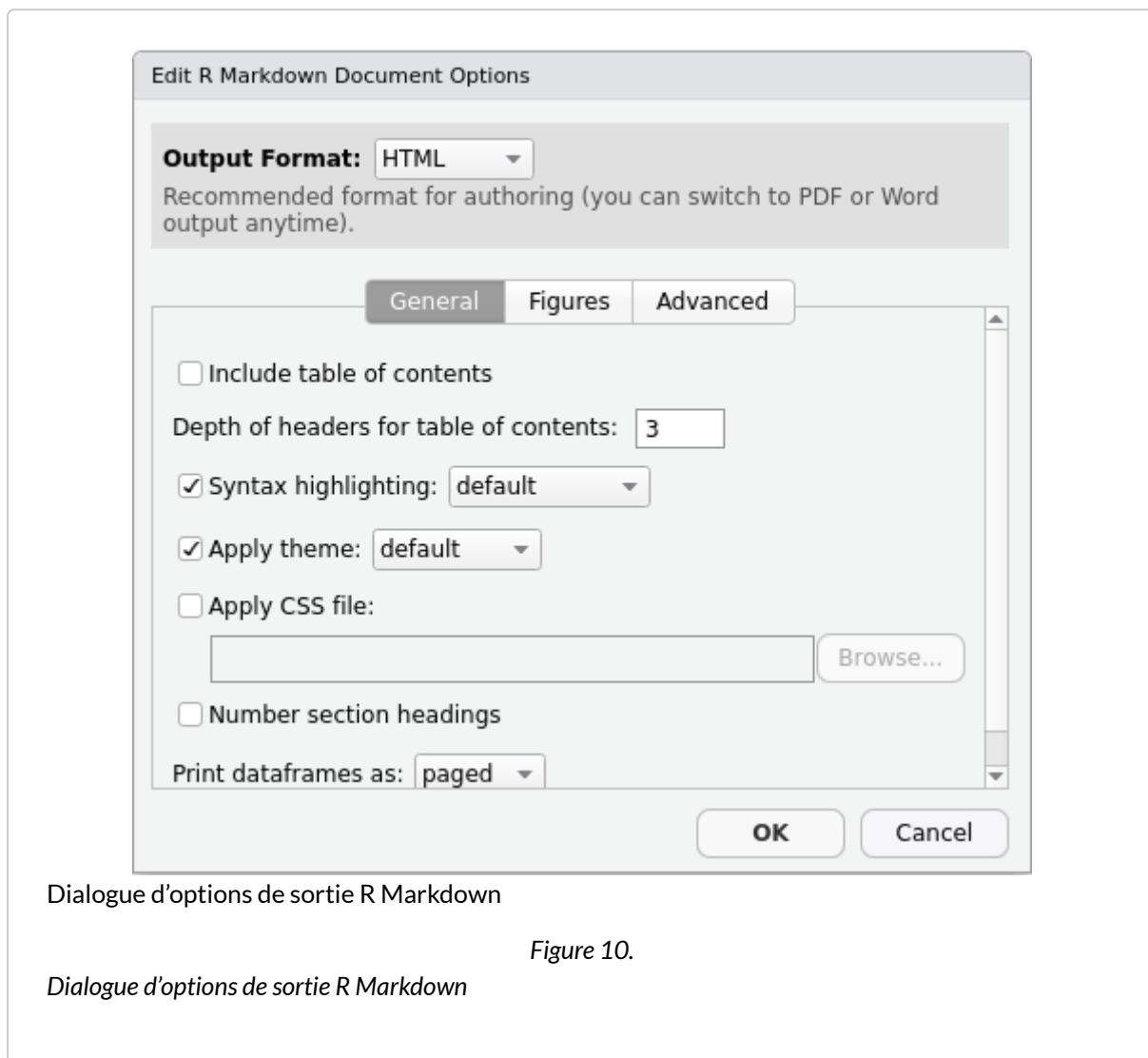
compilation, ainsi que les messages d’erreur éventuels. Si tout se passe bien, Le document devrait s’afficher soit dans une fenêtre Viewer de RStudio (pour la sortie HTML), soit dans le logiciel par défaut de votre ordinateur.

## Personnaliser le document généré

La personnalisation du document généré se fait en modifiant des options dans le préambule du document. RStudio propose néanmoins une petite interface graphique permettant de changer ces options plus facilement. Pour cela, cliquez sur l’icône en forme d’engrenage à droite du bouton Knit et choisissez Output Options...



Une boîte de dialogue s’affiche vous permettant de sélectionner le format de sortie souhaité et, selon le format, différentes options :



Pour le format **HTML** par exemple, l'onglet *General* vous permet de spécifier si vous voulez une table des matières, sa profondeur, les thèmes à appliquer pour le document et la coloration syntaxique des blocs **R**, etc. L'onglet *Figures* vous permet de changer les dimensions par défaut des graphiques générés.

#### NOTE

Une option très intéressante pour les fichiers **HTML**, accessible via l'onglet *Advanced*, est l'entrée *Create standalone HTML document*. Si elle est cochée (ce qui est le cas par défaut), le document **HTML** généré contiendra en un seul fichier le code **HTML** mais aussi les images et toutes les autres ressources nécessaires à son affichage. Ceci permet de générer des fichiers (parfois assez volumineux) que vous pouvez transférer très facilement à quelqu'un par mail ou en le mettant en ligne quelque part. Si la case n'est pas cochée, les images et autres ressources sont placées dans un dossier à part.

Lorsque vous changez des options, **RStudio** va en fait modifier le préambule de votre document. Ainsi, si vous choisissez d'afficher une table des matières et de modifier le thème de coloration syntaxique, votre en-tête va devenir quelque chose comme :

```
---
```

```
title: "Test R Markdown"
```

```
output:
```

```
  html_document:
```

```
    highlight: kate
```

```
    toc: yes
```

```
---
```

Vous pouvez modifier les options directement en éditant le préambule.

À noter qu'il est possible de spécifier des options différentes selon les formats, par exemple :

```
---
```

```
title: "Test R Markdown"
```

```
output:
```

```
  html_document:
```

```
    highlight: kate
```

```
    toc: yes
```

```
  pdf_document:
```

```
    fig_caption: yes
```

```
    highlight: kate
```

```
---
```

La liste complète des options possibles est présente sur [le site de la documentation officielle](#) (très complet et bien fait) et sur l'antisèche et le guide de référence, accessibles depuis **RStudio** via le menu *Help* puis *Cheatsheets*.

## Options des blocs de code R

Il est également possible de passer des options à chaque bloc de code R pour modifier son comportement.

On rappelle qu'un bloc de code se présente de la manière suivante :

```
```{r}
```

```
x <- 1:5
```

```
```
```

Les options d'un bloc de code sont à placer à l'intérieur des accolades `{r}`.

## Nom du bloc

La première possibilité est de donner un *nom* au bloc. Celui-ci est indiqué directement après le `r` :

```
{r nom_du_bloc}
```

Il n'est pas obligatoire de nommer un bloc, mais cela peut être utile en cas d'erreur à la compilation, pour identifier le bloc ayant causé le problème. Attention, on ne peut pas avoir deux blocs avec le même nom.

## Options

En plus d'un nom, on peut passer à un bloc une série d'options sous la forme `option = valeur`. Voici un exemple de bloc avec un nom et des options :

```
```{r mon_bloc, echo = FALSE, warning = TRUE}
x <- 1:5
```

```

Et un exemple de bloc non nommé avec des options :

```
```{r echo = FALSE, warning = FALSE}
x <- 1:5
```

```

Une des options la plus utile est l'option `echo`. Par défaut `echo` vaut `TRUE`, et le bloc de code R est inséré dans le document généré, de cette manière :

```
R> x <- 1:5
print(x)
```

```
[1] 1 2 3 4 5
```

Mais si on positionne l'option `echo=FALSE`, alors le code R n'est plus inséré dans le document, et seul le résultat est visible :

```
[1] 1 2 3 4 5
```

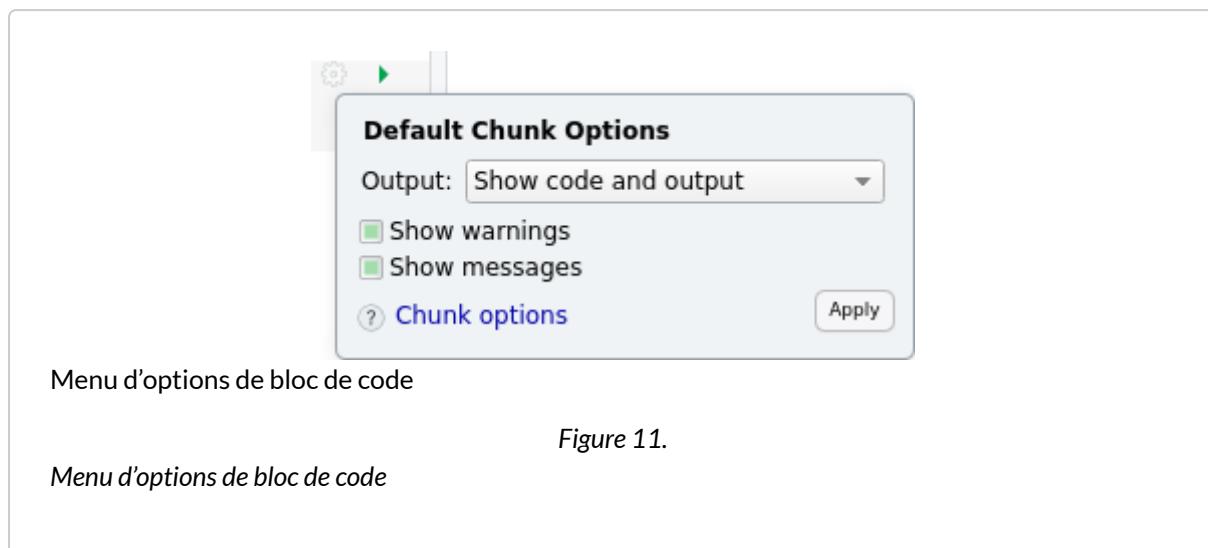
Voici une liste de quelques unes des options disponibles :

| Option  | Valeurs                       | Description  |
|---------|-------------------------------|--|
| echo    | TRUE/FALSE                    | Afficher ou non le code R dans le document                 |
| eval    | TRUE/FALSE                    | Exécuter ou non le code R à la compilation                 |
| include | TRUE/FALSE                    | Inclure ou non le code R et ses résultats dans le document |
| results | “hide”/“asis”/“markup”/“hold” | Type de résultats renvoyés par le bloc de code             |
| warning | TRUE/FALSE                    | Afficher ou non les avertissements générés par le bloc     |
| message | TRUE/FALSE                    | Afficher ou non les messages générés par le bloc           |

Il existe de nombreuses autres options décrites notamment dans [guide de référence R Markdown](#) (PDF en anglais).

## Modifier les options

Il est possible de modifier les options manuellement en éditant l'en-tête du bloc de code, mais on peut aussi utiliser une petite interface graphique proposée par **RStudio**. Pour cela, il suffit de cliquer sur l'icône d'engrenage située à droite sur la ligne de l'en-tête de chaque bloc :



Vous pouvez ensuite modifier les options les plus courantes, et cliquer sur **Apply** pour les appliquer.

## Options globales

On peut vouloir appliquer une option à l'ensemble des blocs d'un document. Par exemple, on peut souhaiter par défaut ne pas afficher le code R de chaque bloc dans le document final.

On peut positionner une option globalement en utilisant la fonction `knitr::opts_chunk$set()`. Par exemple, insérer `knitr::opts_chunk$set(echo = FALSE)` dans un bloc de code positionnera l'option `echo = FALSE` par défaut pour tous les blocs suivants.

En général, on place toutes ces modifications globales dans un bloc spécial nommé `setup` et qui est le premier bloc du document :

```
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
...```

```

### NOTE

Par défaut RStudio exécute systématiquement le contenu du bloc `setup` avant d'exécuter celui d'un autre bloc.

## Mise en cache des résultats

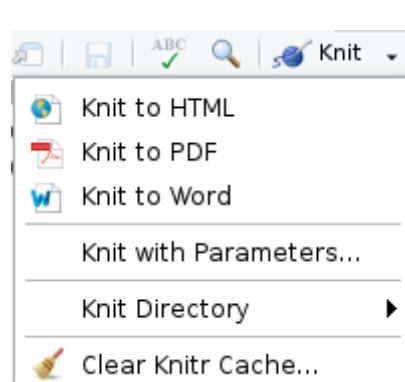
Compiler un document **R Markdown** peut être long, car il faut à chaque fois exécuter l’ensemble des blocs de code R qui le constituent.

Pour accélérer cette opération, **R Markdown** utilise un système de *mise en cache* : les résultats de chaque bloc sont enregistrés dans un fichier et à la prochaine compilation, si le code et les options du bloc n’ont pas été modifiés, c’est le contenu du fichier de cache qui est utilisé, ce qui évite d’exécuter le code R.

### NOTE

On peut activer ou désactiver la mise en cache des résultats pour chaque bloc de code avec l’option `cache = TRUE` ou `cache = FALSE`, et on peut aussi désactiver totalement la mise en cache pour le document en ajoutant `knitr::opts_chunk$set(echo = FALSE)` dans le premier bloc *setup*.

Ce système de cache peut poser problème par exemple si les données source changent : dans ce cas les résultats de certains blocs peuvent ne pas être mis à jour s’ils sont présents en cache. Dans ce cas, on peut vider le cache du document, ce qui forcera un recalcul de tous les blocs de code à la prochaine compilation. Pour cela, vous pouvez ouvrir le menu *Knit* et choisir *Clear Knitr Cache...* :



Menu Knit

Figure 12.

Menu Knit

# Rendu des tableaux

## Tableaux croisés

Par défaut, les tableaux issus de la fonction `table` sont affichés comme ils apparaissent dans la console de R, en texte brut :

```
R> library(questionr)
data(hdv2003)
tab <- lprop(table(hdv2003$qualif, hdv2003$sexe))
tab
```

	Homme	Femme	Total
Ouvrier specialise	47.3	52.7	100.0
Ouvrier qualifie	78.4	21.6	100.0
Technicien	76.7	23.3	100.0
Profession intermediaire	55.0	45.0	100.0
Cadre	55.8	44.2	100.0
Employe	16.2	83.8	100.0
Autre	36.2	63.8	100.0
Ensemble	44.8	55.2	100.0

On peut améliorer leur présentation en utilisant la fonction `kable` de l'extension `knitr`. Celle-ci fournit un formatage adapté en fonction du format de sortie. On aura donc des tableaux «propres» que ce soit en **HTML**, **PDF** ou aux formats traitements de texte :

```
R> library(knitr)
kable(tab)
```

	Homme	Femme	Total
Ouvrier specialise	47.29064	52.70936	100
Ouvrier qualifie	78.42466	21.57534	100
Technicien	76.74419	23.25581	100
Profession intermediaire	55.00000	45.00000	100
Cadre	55.76923	44.23077	100
Employe	16.16162	83.83838	100
Autre	36.20690	63.79310	100
Ensemble	44.82759	55.17241	100

Différents arguments permettent de modifier la sortie de `kable`. `digits`, par exemple, permet de spécifier le nombre de chiffres significatifs à afficher dans les colonnes de nombres :

```
R> kable(tab, digits = 1)
```

	Homme	Femme	Total
Ouvrier specialise	47.3	52.7	100
Ouvrier qualifie	78.4	21.6	100
Technicien	76.7	23.3	100
Profession intermediaire	55.0	45.0	100
Cadre	55.8	44.2	100
Employe	16.2	83.8	100
Autre	36.2	63.8	100
Ensemble	44.8	55.2	100

Pour une présentation plus avancée, on pourra également avoir recours à l'extension `gtsummary` et aux fonctions `tbl_summary`, `tbl_svysummary` et `tbl_regression`.

```
R> library(gtsummary)
  theme_gtsummary_language("fr", decimal.mark = ",",
                           big.mark = " ")
```

Setting `language: fr` theme

```
R> hdv2003 %>%
  dplyr::select(sexe, qualif) %>%
 tbl_summary(by = "sexe")
```

Caractéristique	Homme, N = 899 <sup>1</sup>	Femme, N = 1 101 <sup>1</sup>
qualif		
Ouvrier specialise	96 (13%)	107 (12%)
Ouvrier qualifie	229 (31%)	63 (6,9%)
Technicien	66 (8,9%)	20 (2,2%)
Profession intermediaire	88 (12%)	72 (7,9%)
Cadre	145 (20%)	115 (13%)
Employe	96 (13%)	498 (55%)
Autre	21 (2,8%)	37 (4,1%)
Manquant	158	189

<sup>1</sup> n (%)

## Tableaux de données et tris à plat

En ce qui concerne les tableaux de données (*tibble* ou *data frame*), l'affichage **HTML** par défaut se contente d'un affichage texte comme dans la console, très peu lisible dès que le tableau dépasse une certaine dimension.

Une alternative est d'utiliser la fonction `paged_table`, qui affiche une représentation HTML paginée du tableau :

```
R> rmarkdown::paged_table(hdv2003)
```

Une autre alternative est d'utiliser `kable`, ou encore la fonction `datatable` de l'extension `DT`, qui

propose encore davantage d’interactivité :

```
R> DT::datatable(hdv2003)
```

Dans tous les cas il est déconseillé d’afficher de cette manière un tableau de données de très grandes dimensions, car le fichier HTML résultant contiendrait l’ensemble des données et serait donc très volumineux.

#### NOTE

On peut définir un mode d’affichage par défaut pour tous les tableaux de données en modifiant les *Output options* du format **HTML** (onglet *General*, *Print dataframes as*), ou en modifiant manuellement l’option `df_print` de l’entrée `html_document` dans le préambule.

À noter que les tableaux issus de la fonction `freq` de **questionr** s’affichent comme des tableaux de données (et non comme des tableaux croisés).

## Autres extensions pour présenter des tableaux

Il existe de nombreuses extensions offrant des fonctionnalités de présentation enrichie des tableaux et autres objets R.

### kableExtra

L’extension **kableExtra** a pour objectif d’étendre la fonction `kable` de **knitr** avec des options comme la possibilité de regrouper des colonnes, ajouter des notes de tableau, coloriser certaines cellules...

```
R> library(kableExtra)
dt <- mtcars[1:5, 1:4]

kable(dt, format = "html", caption = "Demo Table") %>%
  kable_styling(
    bootstrap_options = "striped",
    full_width = F
  ) %>%
  add_header_above(c(" ", "Group 1" = 2, "Group 2[note]" = 2)) %>%
  add_footnote(c("table footnote"))
```

Demo Table

	Group 1	Group 2 <sup>a</sup>		
	mpg	cyl	disp	hp
Mazda RX4	21.0	6	160	110
Mazda RX4 Wag	21.0	6	160	110
Datsun 710	22.8	4	108	93
Hornet 4 Drive	21.4	6	258	110
Hornet Sportabout	18.7	8	360	175

<sup>a</sup>table footnote

Pour une présentation détaillée, vous pouvez vous référer aux [vignettes disponibles avec l'extension](#).

Notamment, il est possible d'utiliser [kableExtra](#) en conjonction avec l'extension [formattable](#) pour des rendus colorés et encore plus personnalisé de vos tableaux (voir la [vignette dédiée](#)).

```
R> library(knitr)
library(kableExtra)
library(formattable)
library(dplyr, quietly = TRUE)
```

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

```
The following object is masked from 'package:kableExtra':
```

```
group_rows
```

The following objects are masked from 'package:stats':

filter, lag

The following objects are masked from 'package:base':

intersect, setdiff, setequal, union

```
R> mtcars[1:5, 1:4] %>%
  mutate(
    car = row.names.,
    mpg = color_tile("white", "orange")(mpg),
    cyl = cell_spec(cyl, "html",
      angle = (1:5) * 60,
      background = "red", color = "white", align = "center"
    ),
    disp = ifelse(disp > 200,
      cell_spec(disp, "html", color = "red", bold = T),
      cell_spec(disp, "html", color = "green", italic = T)
    ),
    hp = color_bar("lightgreen")(hp)
  ) %>%
  relocate(car) %>%
  kable("html", escape = F) %>%
  kable_styling("hover", full_width = F) %>%
  column_spec(5, width = "3cm") %>%
  add_header_above(c(" ", "Hello" = 2, "World" = 2))
```

	Hello		World	
car	mpg	cyl	disp	hp
Mazda RX4	21.0	σ	160	110
Mazda RX4 Wag	21.0	9	160	110
Datsun 710	22.8	4	108	93
Hornet 4 Drive	21.4	♂	258	110
Hornet Sportabout	18.7	♀	360	175

## gt

L'extension **gt** en cours de développement par l'équipe de **RStudio** vise, un peu sur le modèle de **ggplot2**, à proposer une grammaire des tableaux. L'extension n'est pour le moment compatible qu'avec un rendu **HTML** mais la prise en compte du format **PDF** est prévue. C'est une extension encore assez nouvelle mais qui risque de se développer assez rapidement vu les équipes en charge de son développement.

```
R> library(gt)
```

```
Attaching package: 'gt'
```

```
The following object is masked from 'package:formattable':
```

```
currency
```

```
R> exibble %>%
  gt(
    rowname_col = "row",
    groupname_col = "group"
  ) %>%
  tab_spinner(
    label = "Date et/ou Heure",
    columns = vars(date, time, datetime)
  ) %>%
  fmt_number(
    columns = vars(num),
    decimals = 2,
    locale = "fr"
  ) %>%
  fmt_date(
    columns = vars(date),
    date_style = 7
  ) %>%
  fmt_time(
    columns = vars(time),
    time_style = 4
  ) %>%
  fmt_datetime(
    columns = vars(datetime),
    date_style = 6,
    time_style = 4
  ) %>%
  fmt_currency(
    columns = vars(currency),
    currency = "EUR",
    locale = "fr"
  ) %>%
  tab_options(
    column_labels.font.size = "small",
    table.font.size = "small",
    row_group.font.size = "small",
    data_row.padding = px(3)
  ) %>%
  summary_rows(
    groups = TRUE,
    columns = vars(num),
    fns = list(
      moyenne = ~ mean(., na.rm = TRUE),
      total = ~ sum(., na.rm = TRUE)
    )
  ) %>%
  grand_summary_rows(
```

```
columns = vars(currency),  
fns = list("grand total" = ~ sum(., na.rm = TRUE)),  
formatter = fmt_currency,  
currency = "EUR"  
)
```

	num	char	fctr	Date et/ou Heure			currency
				date	time	datetime	
<b>grp_a</b>							
row_1	0,11	apricot	one	15 janv. 2015	1:35	janv. 1, 2018 2:22	€49,95
row_2	2,22	banana	two	15 févr. 2015	2:40	févr. 2, 2018 2:33	€17,95
row_3	33,33	coconut	three	15 mars 2015	3:45	mars 3, 2018 3:44	€1,39
row_4	444,40	durian	four	15 avr. 2015	4:50	avr. 4, 2018 3:55	€65 100,00
moyenne	120.02	–	–	–	–	–	–
total	480.06	–	–	–	–	–	–
<b>grp_b</b>							
row_5	5 550,00	NA	five	15 mai 2015	5:55	mai 5, 2018 4:00	€1 325,81
row_6	NA	fig	six	15 juin 2015	NA	juin 6, 2018 4:11	€13,26
row_7	777 000,00	grapefruit	seven	NA	7:10	juil. 7, 2018 5:22	NA
row_8	8 880 000,00	honeydew	eight	15 août 2015	8:20	NA	€0,44
moyenne	3,220,850.00	–	–	–	–	–	–
total	9,662,550.00	–	–	–	–	–	–

	num	char	fctr	Date et/ou Heure			currency
				date	time	datetime	
grand total	-	-	-	-	-	-	€66,508.79

Pour en savoir plus : <https://gt.rstudio.com/>

## printr

L'extension **printr** développée par le même auteur que **knitr** étent le fonctionnement par défaut de **knitr**. Une fois chargée, le rendu automatique de certains objets (tel qu'un tableau croisé à trois variables) sera amélioré.

```
R> library(printr)
```

```
Registered S3 method overwritten by 'printr':
method           from
knit_print.data.frame rmarkdown
```

```
R> x1 <- sample(letters[1:2], 1000, TRUE)
x2 <- sample(letters[1:2], 1000, TRUE)
x3 <- sample(letters[1:2], 1000, TRUE)
table(x1, x2, x3)
```

x1	x2	x3	Freq
a	a	a	136
		b	124
b	a	a	132
		b	131
b	a	a	129
		b	102
b	a	a	123
		b	123

Pour d'autres exemples, voir la documentation de l'extension sur <https://yihui.name/printr/>.

## Autres extensions

On peut également citer les extensions suivantes :

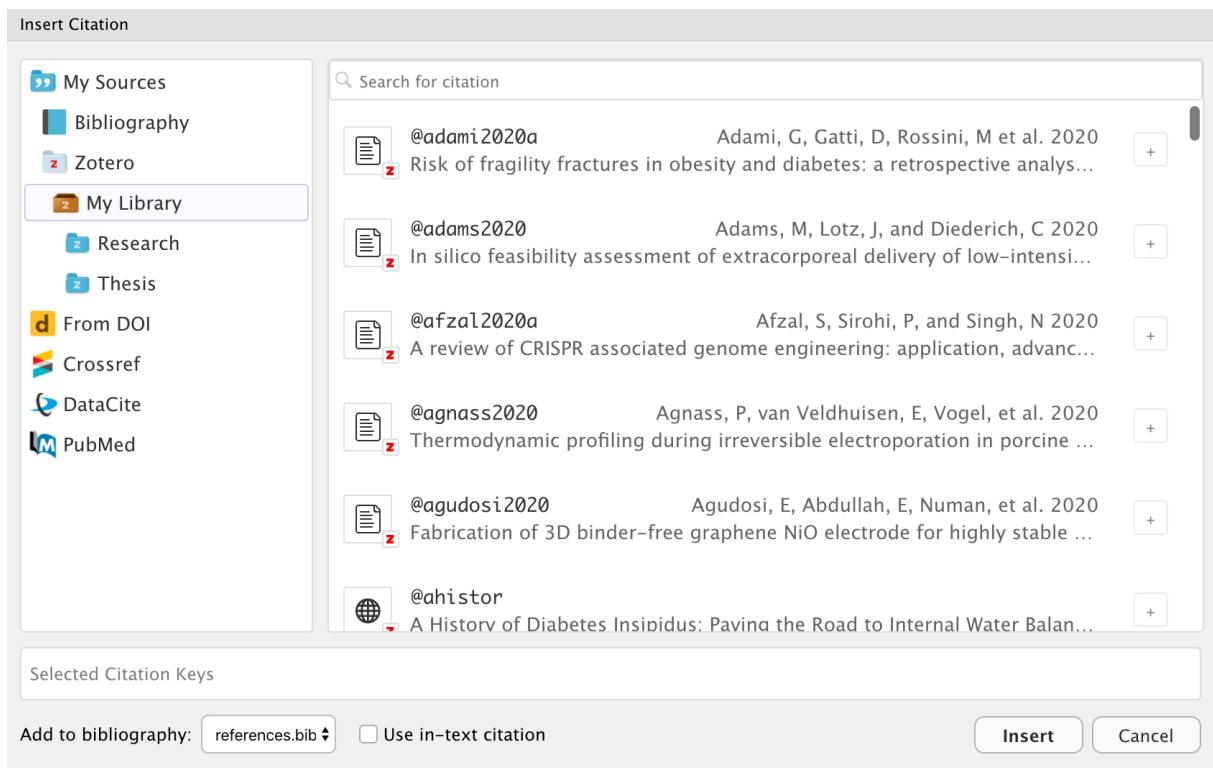
- **flextable**, assez complet (<https://davidgohel.github.io/flextable>)
- **pander**
- **xtable**
- **tables**
- **sjPlot** et ses fonctions `sjt*`

Pour chacune, vous trouverez une documentation sur leur page **CRAN**, notamment sous forme de vignettes.

## Éditeur avancé

La version 1.4 de **RStudio** inclue un éditeur **RMarkdown** avancé, avec plusieurs fonctions facilitant la mise en forme. Ces différentes fonctionnalités sont présentées sur ce billet de blog : <https://blog.rstudio.com/2020/09/30/rstudio-v1-4-preview-visual-markdown-editing/>

Une fonctionnalité très pratique lorsqu’on écrit des documents scientifiques est la gestion des références bibliographiques. En particulier, la version 1.4 de **RStudio** peut se connecter au logiciel bibliographique **Zotero**. Plus d’informations sur <https://blog.rstudio.com/2020/11/09/rstudio-1-4-preview-citations/>.



## Modèles de documents

On a vu ici la production de documents «classiques», mais **R Markdown** permet de créer bien d'autres choses.

Le site de documentation de l'extension propose [une galerie](#) des différentes sorties possibles. On peut ainsi créer des slides, des sites Web ou même des livres entiers, comme le présent document.

## Slides

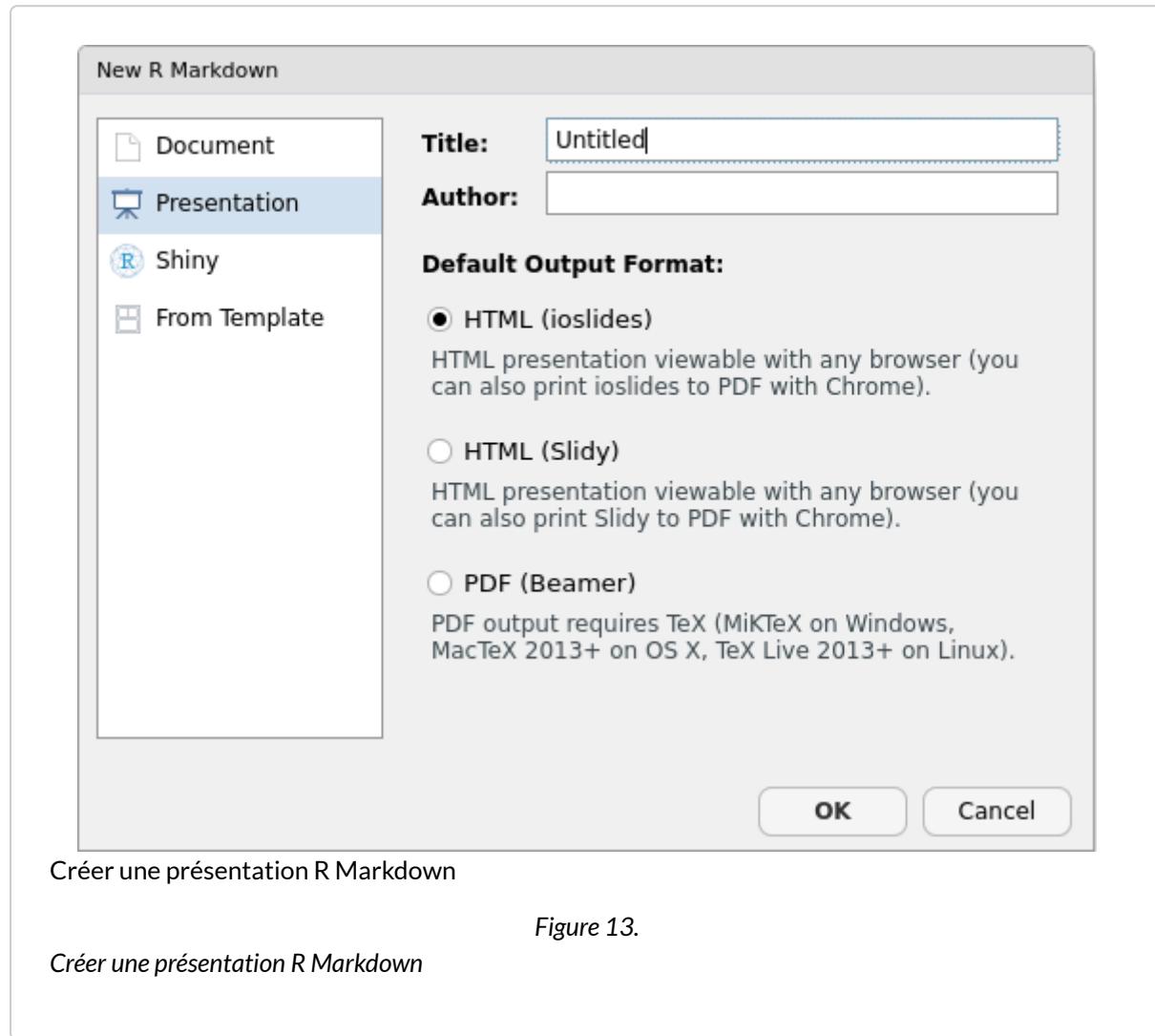
Un usage intéressant est la création de diaporamas pour des présentations sous forme de slides. Le principe reste toujours le même : on mélange texte au format **Markdown** et code **R**, et **R Markdown** transforme le tout en présentations au format **HTML** ou **PDF**. En général les différents slides sont séparés au niveau de certains niveaux de titre.

Certains modèles de slides sont inclus avec **R Markdown**, notamment :

- `ioslides` et `Slidy` pour des présentations HTML
- `beamer` pour des présentations en PDF via `LaTeX`

Quand vous créez un nouveau document dans RStudio, ces modèles sont accessibles via l'entrée

Presentation :



Créer une présentation R Markdown

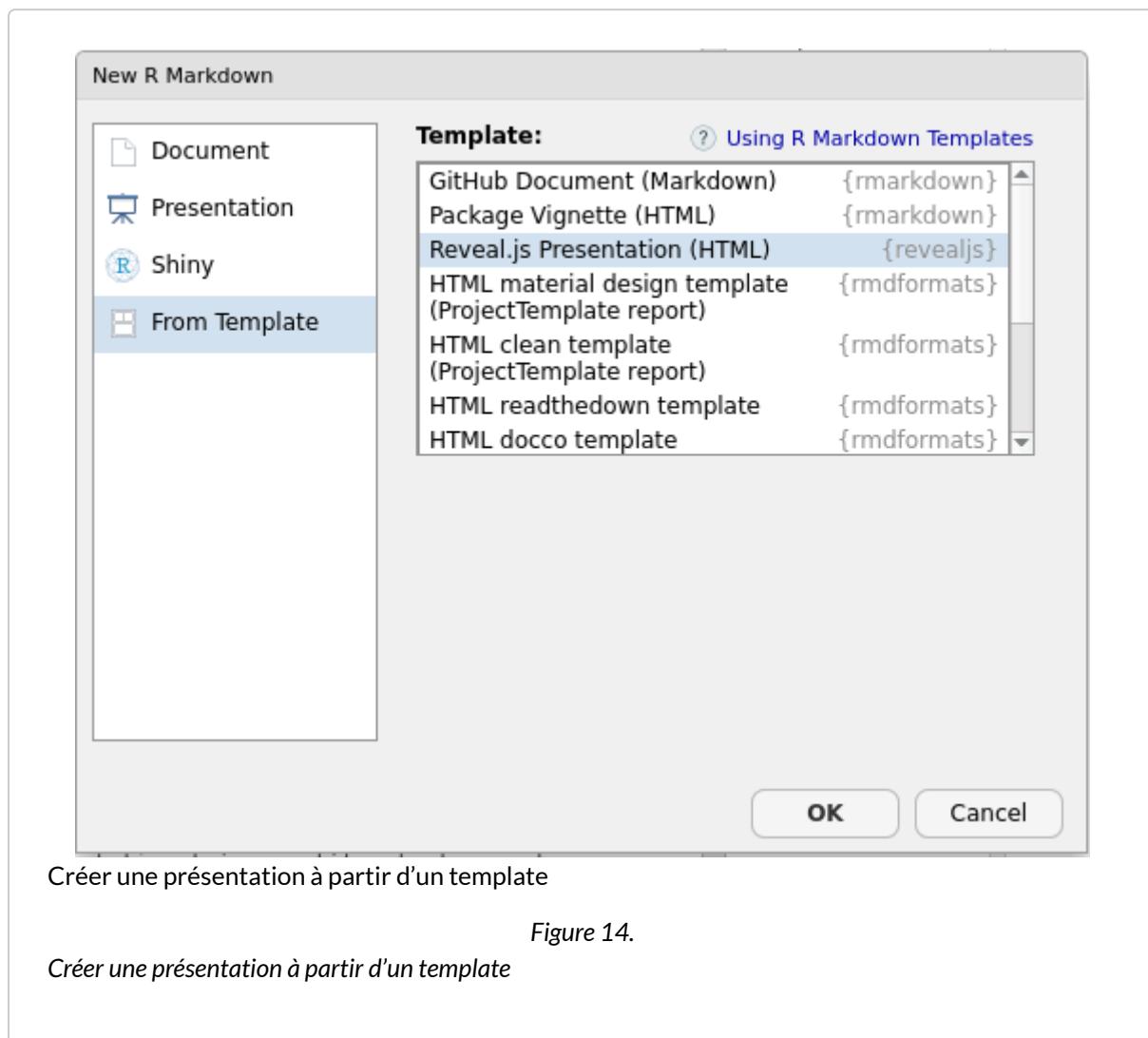
Figure 13.

Créer une présentation R Markdown

D'autres extensions, qui doivent être installées séparément, permettent aussi des diaporamas dans des formats variés. On citera notamment :

- **revealjs** (<https://github.com/rstudio/revealjs>) pour des présentations HTML basées sur le framework reveal.js
- **rmdshower** (<https://github.com/mangothecat/rmdshower>) pour des diaporamas HTML basés sur shower

Une fois l'extension installée, elle propose en général un *template* de départ lorsqu'on crée un nouveau document dans RStudio. Ceux-ci sont accessibles depuis l'entrée *From Template*.



## Templates

Il existe également différents *templates* permettant de changer le format et la présentation des documents générés. Une liste de ces formats et leur documentation associée est accessible depuis la page [formats](#) de la documentation.

On notera notamment :

- le template **distill** comportant de nombreuses options pour des documents scientifiques et techniques au format **HTML** (plus d'infos sur <https://rstudio.github.io/distill/>) ;
- des formats d'article correspondant à des publications dans différentes revues : `jss_article`, `elsevier_article`, etc. ;
- le format **Tufte Handouts** qui permet de produire des documents **PDF** ou **HTML** dans un format

proche de celui utilisé par Edward Tufte pour certaines de ses publications.

Enfin, l’extension **rmdformats** (<https://github.com/juba/rmdformats>) de Julien Barnier propose plusieurs modèles **HTML** adaptés notamment pour des documents longs :

The screenshot shows a web-based interface for the 'readthedown template example'. On the left, a sidebar lists 'Code and tables', 'Figures', and 'MathJax'. The main content area has a title 'readthedown template example' and two sections: 'Code and tables' and 'Syntax highlighting'. Under 'Code and tables', there is a code chunk for R:library(ggplot2)
library(dplyr)

not\_null <- function (v) {
 if (!is.null(v)) return(paste0(v, "**not null**"))
}

data(iris)
tab <- iris %>%
 group\_by(Species) %>%
 summarise(Sepal.Length = mean(Sepal.Length),
 Sepal.Width = mean(Sepal.Width),
 Petal.Length = mean(Petal.Length),
 Petal.Width = mean(Petal.Width))

Under 'Syntax highlighting', it says 'Here is a sample code chunk, just to show that syntax highlighting works as expected.'

Below these, there is a section titled 'Verbatim' with the heading 'Verbatim' and the text 'Here is the structure of the `iris` dataset.' followed by the output of the R command `str(iris)`:

```
'data.frame': 150 obs. of 5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",... 1 1 1 1 1 1 1 1 1 1 ...'
```

Modèle readthedown

Figure 15.

Modèle readthedown

# html\_clean template example

## Code and tables

### Syntax highlighting

Here is a sample code chunk, just to show that syntax highlighting works as expected.

```
library(ggplot2)
library(dplyr)

not.null <- function (v) {
  if (!is.null(v)) return(paste(v, "not null"))
}

data(iris)
tbl <- iris %>%
  group_by(Species) %>%
  summarise(Sepal.Length = mean(Sepal.Length),
            Sepal.Width = mean(Sepal.Width),
            Petal.Length = mean(Petal.Length),
            Petal.Width = mean(Petal.Width))
```

Code and tables  
Syntax highlighting  
Verbatim  
Table  
Figures  
Mathjax

### Verbatim

Here is the structure of the `iris` dataset.

```
str(iris)
```

```
'data.frame': 150 obs. of 5 variables:
$ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
$ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
$ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
$ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
$ Species     : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

### Table

Here is a sample table output.

Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
setosa	5.005	3.428	1.462	1.462
versicolor	5.935	2.770	4.260	4.260

Modèle html\_clean

Figure 16.

Modèle html\_clean

The screenshot shows a sample R Markdown document titled "Sample document". The left sidebar contains tabs for "Code and verbatim", "Tables", "Figures", and "Tabbed sections". The main content area is titled "Code and verbatim" and includes a section titled "Syntax highlighting". It displays a sample code chunk:not\_null <- function (v) {  
 if (!is.null(v)) return(paste(v, "not null"))  
}This code is highlighted in red and green. Below this is a section titled "Verbatim" which shows the structure of the `iris` dataset:str(iris)

```
'data.frame': 150 obs. of 5 variables:  
 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...  
 $ Sepal.Width : num 3.5 2 3.2 3.1 3.6 3.9 3.4 3.4 2.0 3.1 ...  
 $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...  
 $ Petal.Width : num 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 0.1 ...  
 $ Species    : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...'
```

Modèle material

*Figure 17.*

Modèle material

Là encore, la plupart du temps, ces modèles de documents proposent un *template* de départ lorsqu'on crée un nouveau document dans **RStudio** (entrée *From Template*) :

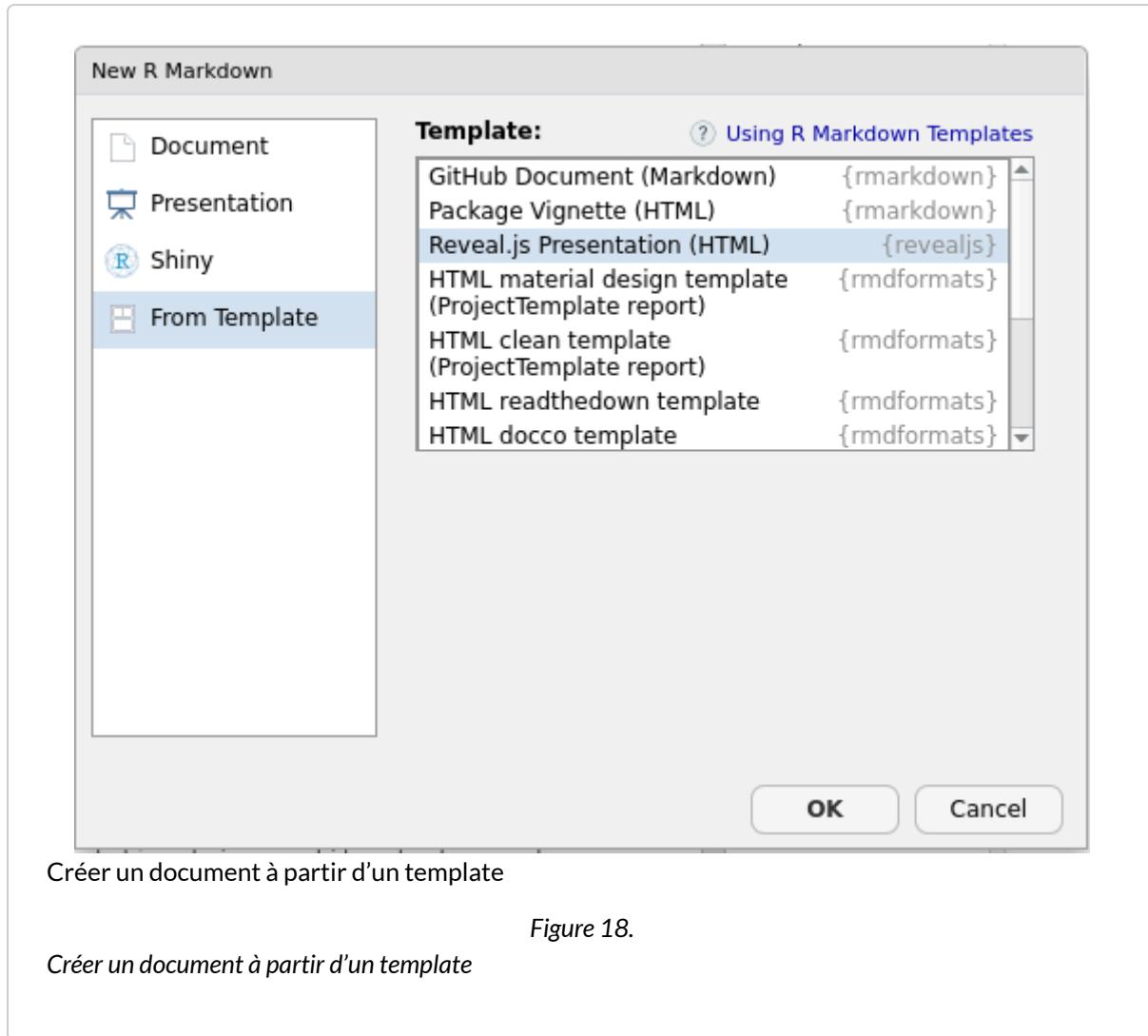


Figure 18.

*Créer un document à partir d'un template*

## Ressources

Les ressources suivantes sont toutes en anglais...

Yihui Xie, J. J. Allaire et Garrett Grolemund ont écrit un livre dédié intitulé *R Markdown: The Definitive Guide*. Ce livre est intégralement accessible à <https://bookdown.org/yihui/rmarkdown/>.

L'ouvrage *R for data science*, accessible en ligne, contient [un chapitre dédié à R Markdown](#).

Le [site officiel de l'extension](#) contient une documentation très complète, tant pour les débutants que pour un usage avancé.

Enfin, l'aide de RStudio (menu *Help* puis *Cheatsheets*) permet d'accéder à deux documents de synthèse : une "antisèche" synthétique (*R Markdown Cheat Sheet*) et un "guide de référence" plus complet (*R*

*Markdown Reference Guide).*

# Mettre en forme des nombres

## IMPORTANT

Ce chapitre est en cours d'écriture.

Le plus simple est d'avoir recours à l'extension `scales` et aux fonctions `number`, `comma`, `percent`, `unit`, `ordinal` et `pvalue`.



# Couleurs et Palettes

Noms de couleur .....	1021
Couleurs RVB .....	1021
Palettes de couleurs .....	1022
Palettes natives .....	1022
Color Brewer .....	1023
Viridis .....	1024
ggsci .....	1025
hrbrthemes .....	1025
ggthemes .....	1027
khroma .....	1031
paletteer .....	1037

## Noms de couleur

Lorsque l'on doit indiquer à R une couleur, notamment dans les fonctions graphiques, on peut mentionner certaines couleurs en toutes lettres (en anglais) comme "red" ou "blue". La liste des couleurs reconnues par R est disponible sur <http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf>.

## Couleurs RVB

En informatique, les couleurs sont usuellement codées en Rouge/Vert/Bleu (voir [https://fr.wikipedia.org/wiki/Rouge\\_vert\\_bleu](https://fr.wikipedia.org/wiki/Rouge_vert_bleu)) et représentées par un code hexadécimal à 6 caractères, précédés du symbole #. Ce code est reconnu par R et on pourra par exemple indiquer "#FF0000" pour la couleur rouge. Le code hexadécimal des différentes couleurs peut s'obtenir aisément sur internet, de nombreux sites étant consacrés aux palettes de couleurs.

Parfois, au lieu du code hexadécimal, les couleurs RVB sont indiquées avec trois chiffres entiers compris entre 0 et 255. La conversion en hexadécimal se fait avec la fonction `rgb`.

```
R> rgb(255, 0, 0, maxColorValue = 255)
```

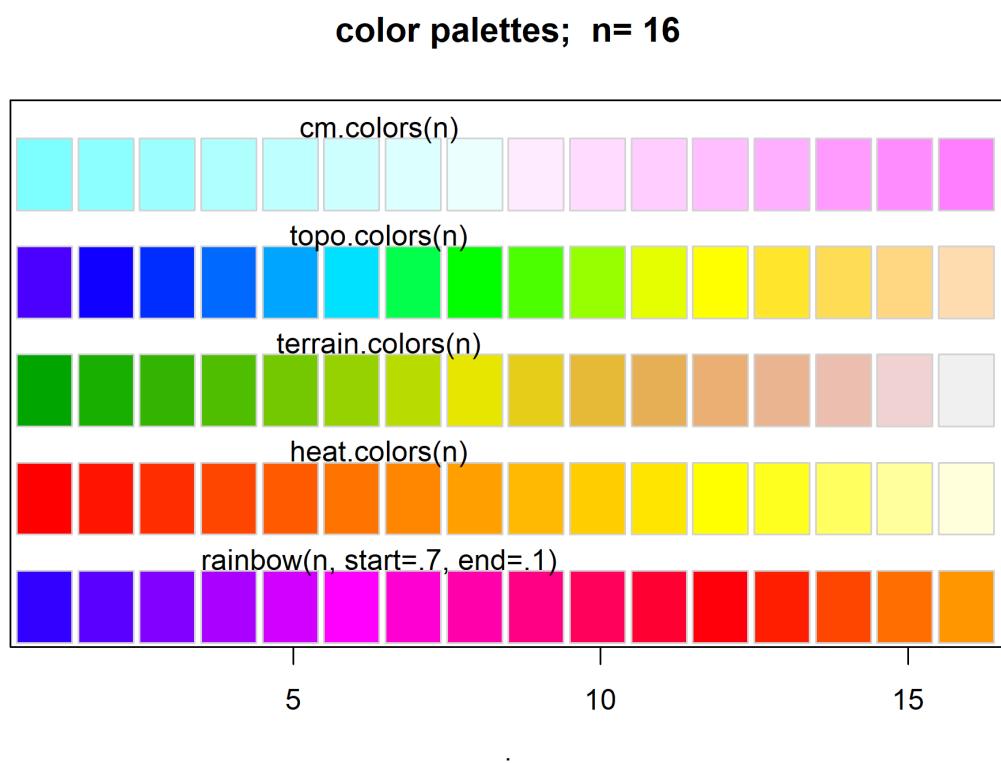
```
[1] "#FF0000"
```

## Palettes de couleurs

### Palettes natives

R fournit nativement quelques palettes de couleurs continues telles que `rainbow`, `heat.colors`, `terrain.colors`, `topo.colors` ou encore `cm.colors`.

Voici un aperçu de ces palettes de couleurs :

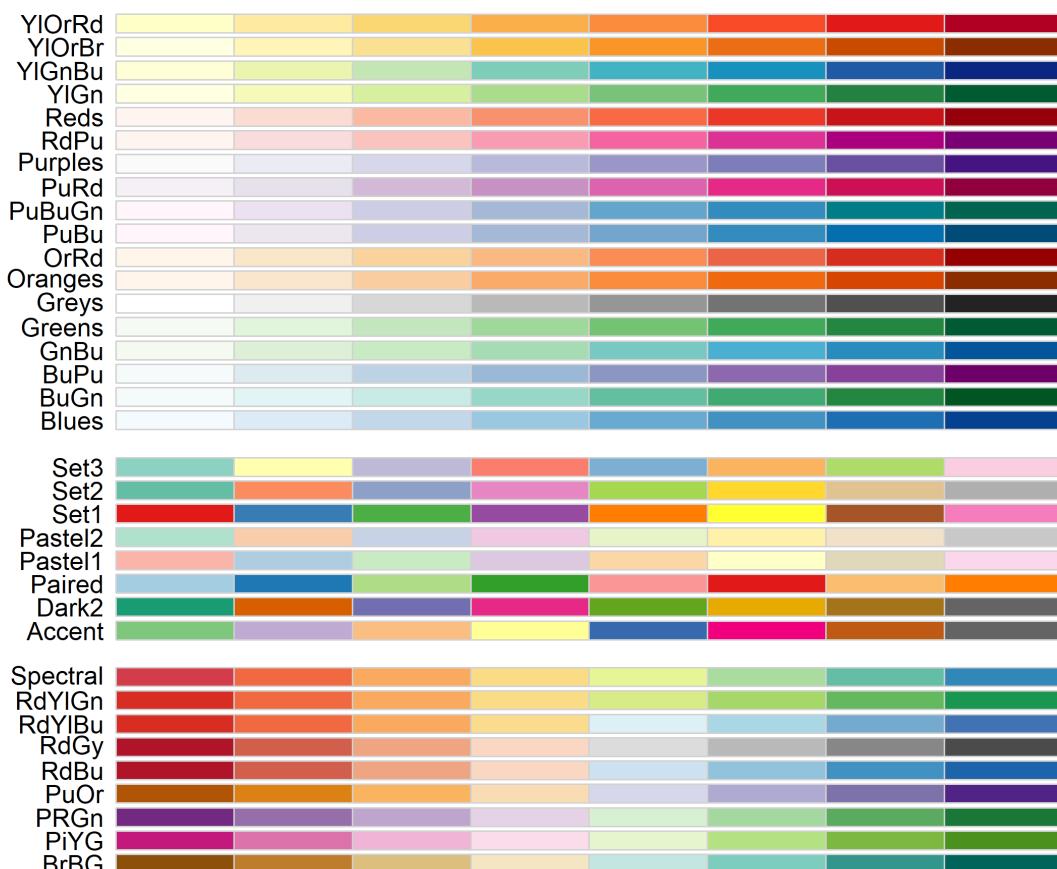


i

## Color Brewer

Le projet **Color Brewer** a développé des palettes cartographiques, à la fois séquentielles, divergentes et catégorielles, présentées en détail sur <http://colorbrewer2.org/>. Pour chaque type de palette, et en fonction du nombre de classes, est indiqué sur ce site si la palette est adaptée aux personnes souffrant de daltonisme, si elle est rendue correctement sur écran, en cas d'impression couleur et en cas d'impression en noir et blanc.

Voici un aperçu des différentes palettes disponibles :



L'extension **RColorBrewer** permet d'accéder à ces palettes sous R.

Si on utilise **ggplot2**, les palettes Color Brewer sont directement disponibles via les fonctions `scale_fill_brewer` et `scale_colour_brewer`.

ATTENTION : les palettes Color Brewer sont seulement implémentées pour des variables catégorielles. Il est cependant possible de les utiliser avec des variables continues en les combinants avec `scale_fill_gradientn` ou `scale_coulour_gradientn` (en remplaçant "Set1" par le nom de la palette désirée) :

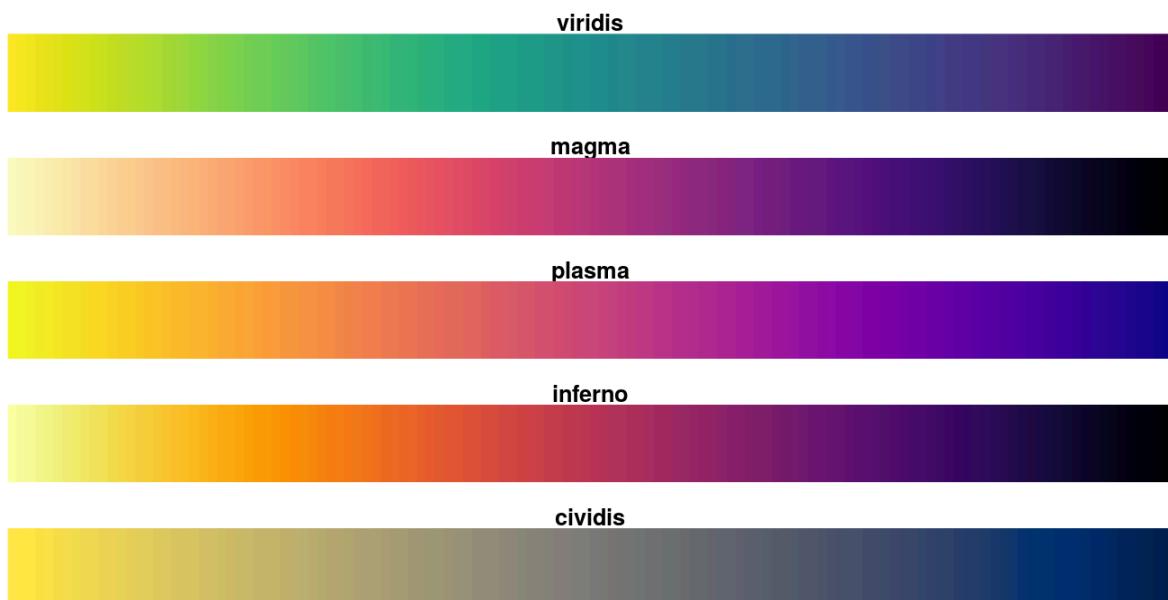
```
R> scale_fill_gradientn(values = RColorBrewer::brewer.pal(6, "Set1"))
```

## Viridis

Les palettes de couleurs de la famille **Viridis** ont initialement été créées pour **Matplotlib** de telles manières que :

- les couleurs sont perçues de manière uniformes, même lorsqu'elles sont imprimées en noir et blanc ;
- les couleurs sont distinguées par les formes les plus courantes de daltonisme.

Voici un aperçu des différentes palettes de cette famille :



Ces palettes sont accessibles de manière générale via l'extension **viridis**.

Elles sont également implémentées dans **ggplot2** via les fonctions `scale_fill_viridis_c` et `scale_colour_viridis_c` pour des variables continues et `scale_fill_viridis_d` et `scale_colour_viridis_d` pour des variables catégorielles.

`scale_colour_viridis_d` pour des variables discrètes.

## ggsci

L'extension `ggsci` fournit plusieurs palettes de couleurs inspirées de journaux scientifiques (comme le *Lancet* ou le *New England Journal of Medicine*) mais aussi de séries/films (comme les *Simpsons* ou *Star Trek*).

On y trouve également des palettes continues basées sur *Material Design* de **Google**.

Toutes ces palettes sont directement utilisables avec `ggplot2`.

Plus d'informations sur <https://cran.r-project.org/web/packages/ggsci/vignettes/ggsci.html>.

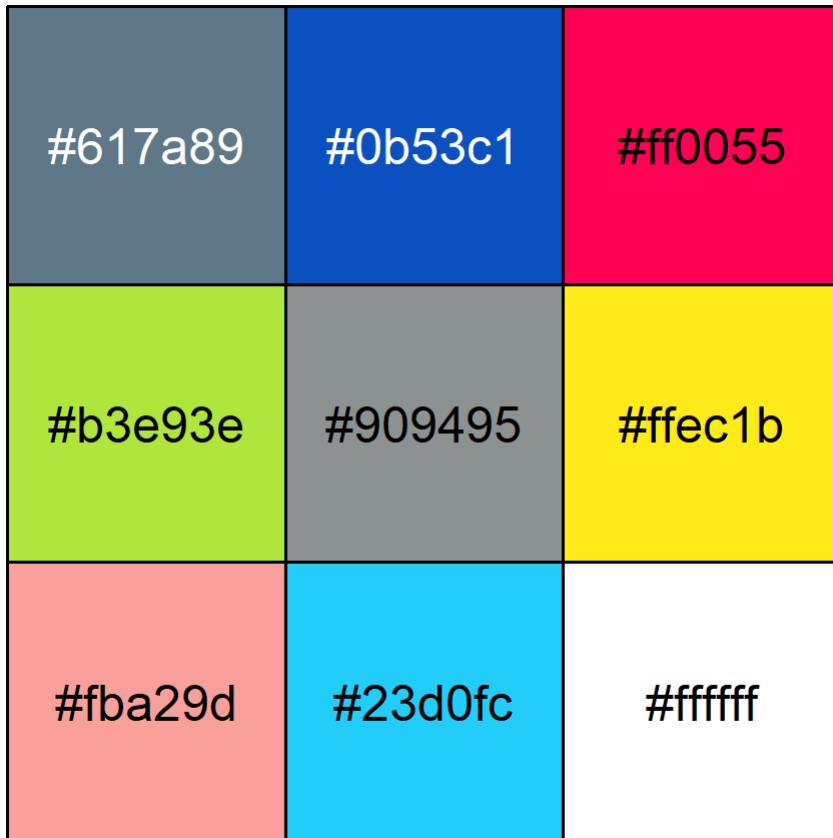
## hrbrthemes

L'extension `hrbrthemes` propose deux palettes discrètes : `ipsum_pal` et `ft_pal` ainsi que les fonctions correspondantes pour `ggplot2`.

```
R> library(hrbrthemes)
scales::show_col(ipsum_pal()(9))
```



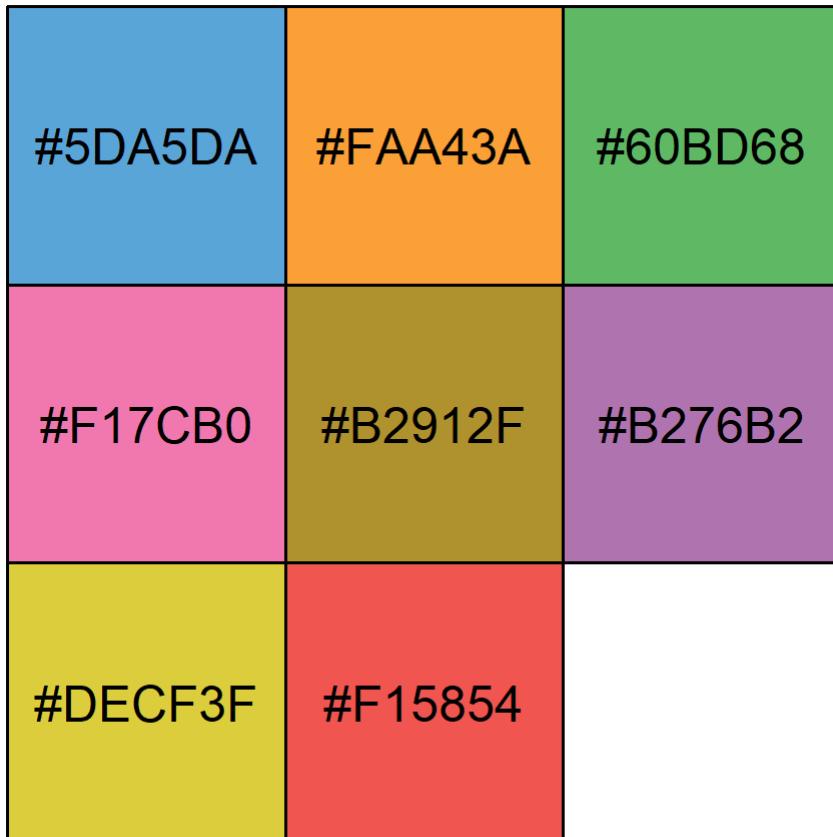
```
R> scales::show_col(ft_pal()(9))
```



## ggthemes

L'extension `ggthemes` fournit plusieurs palettes de couleurs, en particulier `few_pal` issue de l'ouvrage *Show Me the Numbers* (2012) de Stephen Few ; `colorblind_pal` adaptées aux personnes daltoniennes ; `ptol_pal` inspirée de l'ouvrage *Colour Schemes* (2012) de Paul Tol ; et diverses autres (voir <https://jrnlold.github.io/ggthemes/>).

```
R> library(ggthemes)
scales::show_col(few_pal()(8))
```



```
R> scales::show_col(few_pal("Dark")(8))
```



```
R> scales::show_col(colorblind_pal()(8))
```



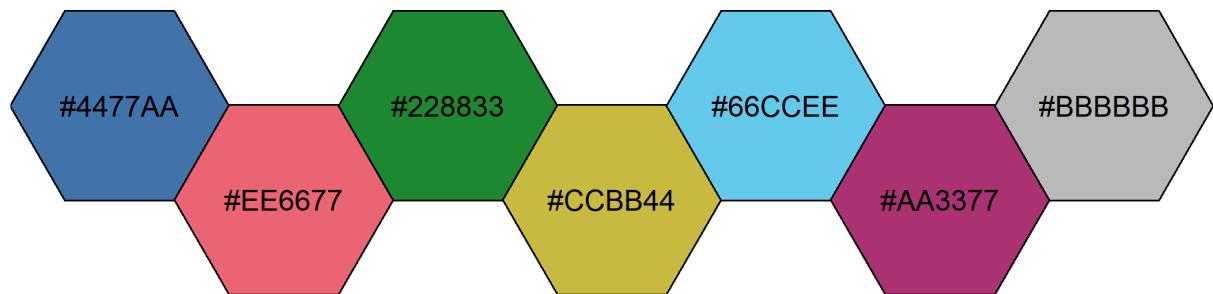
```
R> scales::show_col(ptol_pal()(12))
```



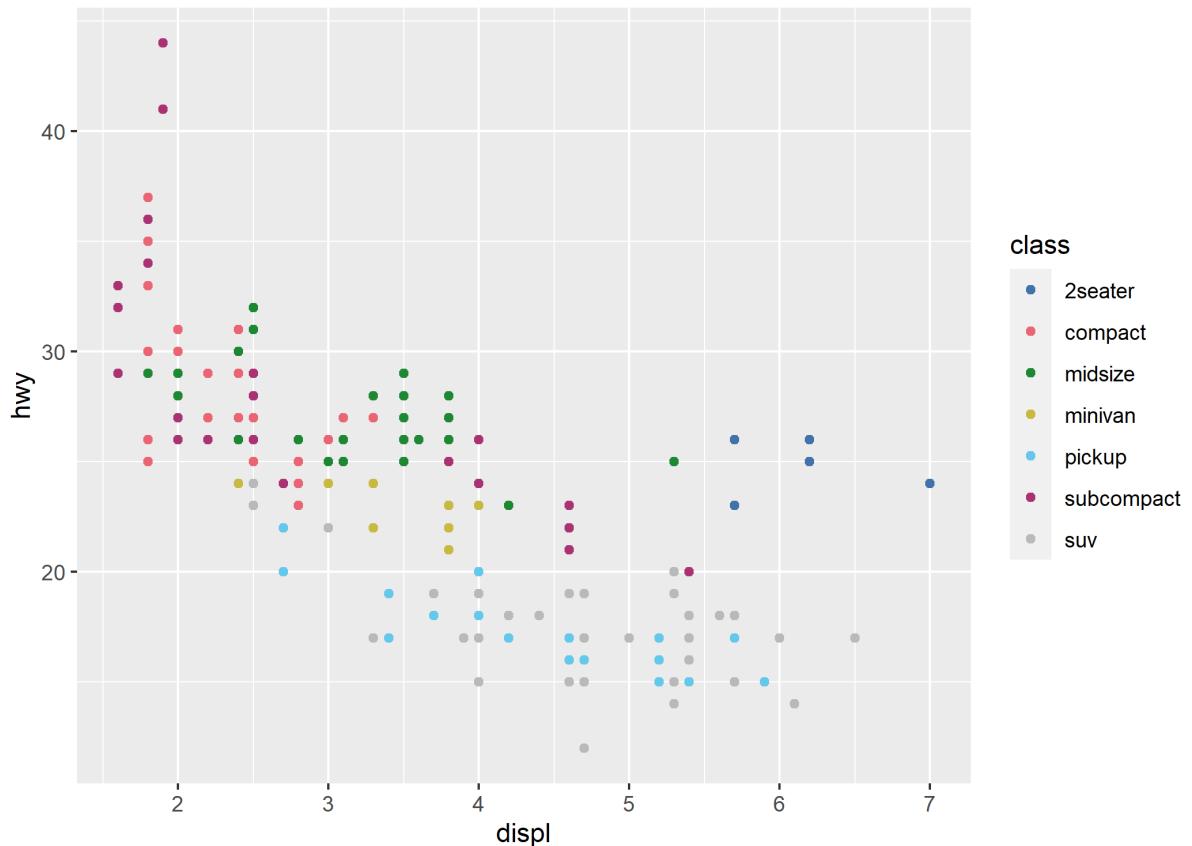
## khroma

L'extension [khroma](#) implémente elle aussi les palettes de couleurs proposées par Paul Tol dans *Colour Schemes* (2012) mais de manière plus complète que [ggthemes](#).

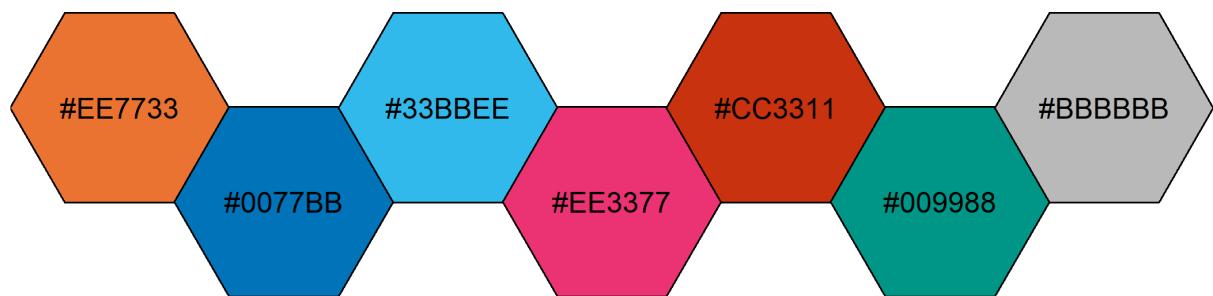
```
R> library(khroma)
  plot_scheme(colour("bright")(7), colours = TRUE)
```



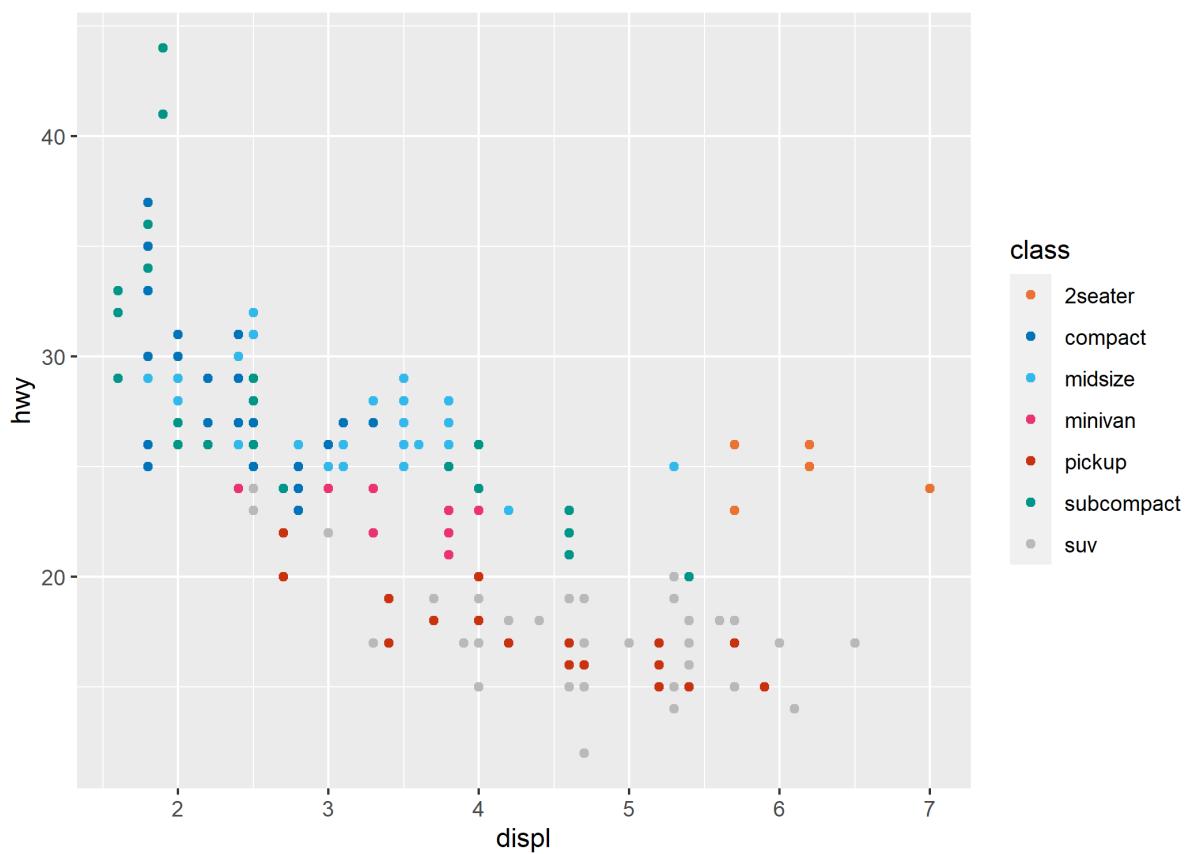
```
R> library(ggplot2)
p <- ggplot(mpg) +
  aes(x = displ, y = hwy, colour = class) +
  geom_point()
p + scale_colour_bright()
```



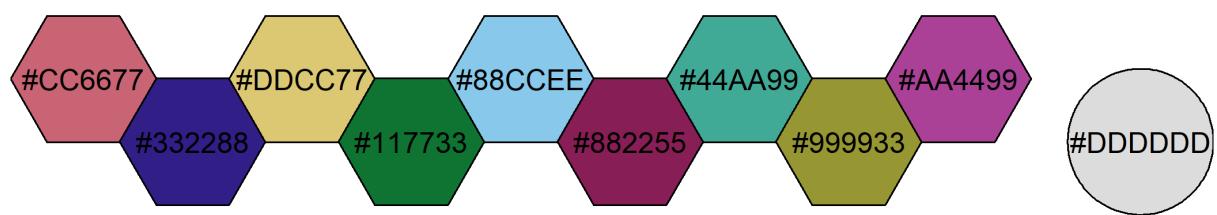
```
R> plot_scheme(colour("vibrant")(7), colours = TRUE)
```



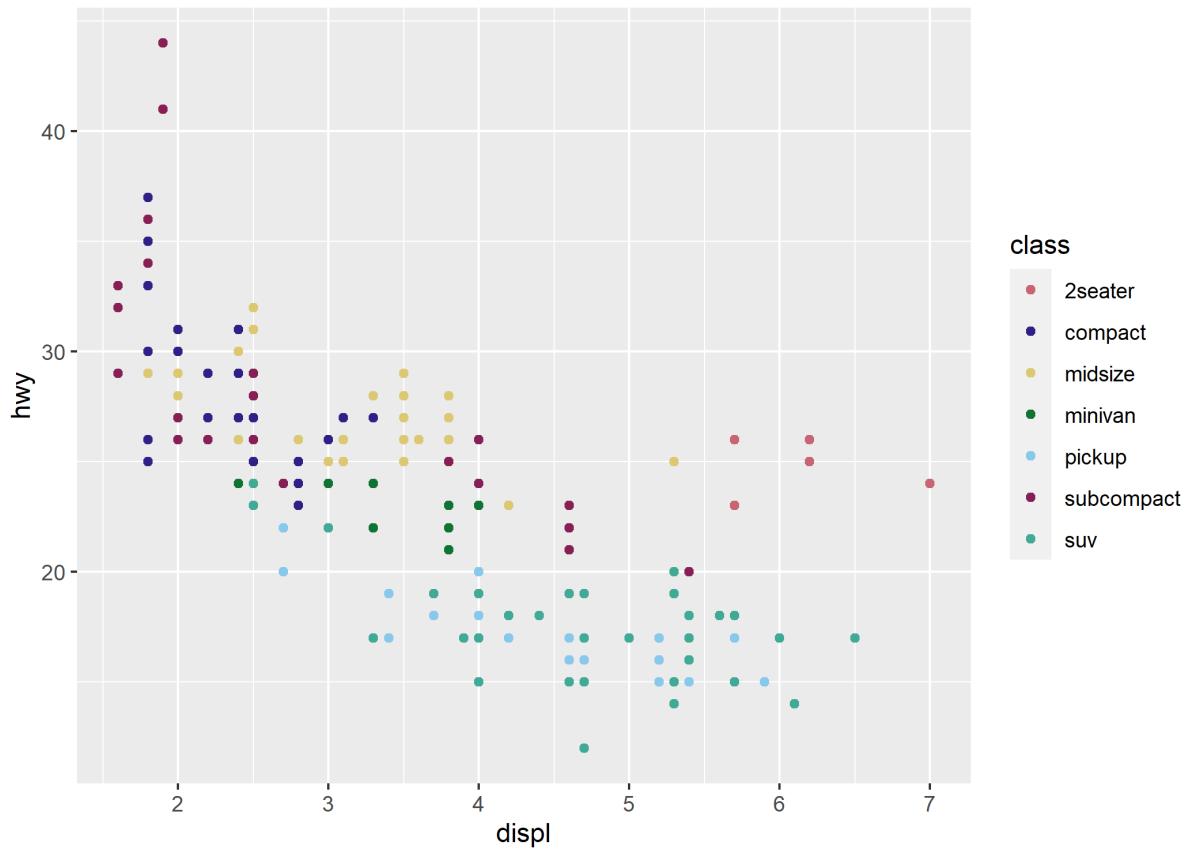
```
R> p + scale_colour_vibrant()
```



```
R> plot_scheme(colour("muted")(9), colours = TRUE)
```



```
R> p + scale_colour_muted()
```



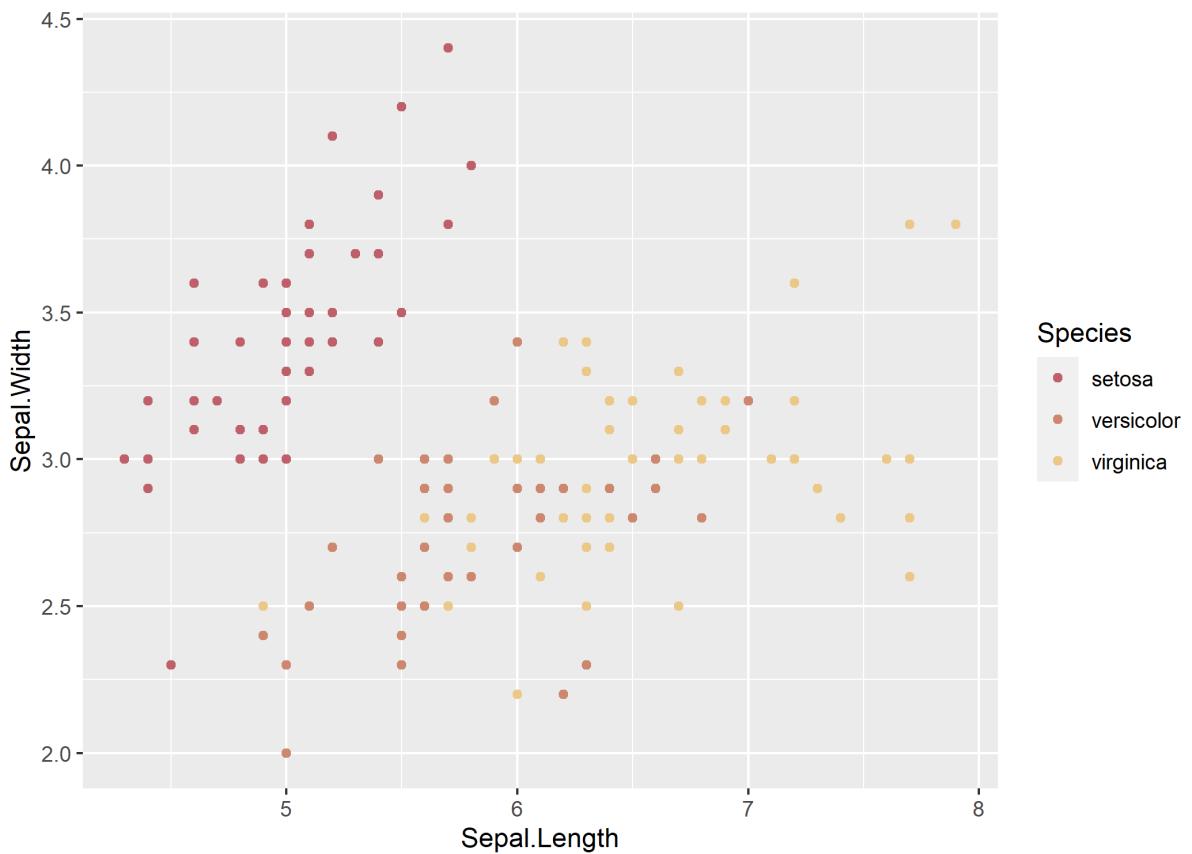
Pour plus de détails, voir <https://khroma.archaeo.science/articles/tol.html>

## paletteer

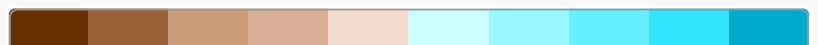
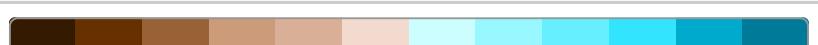
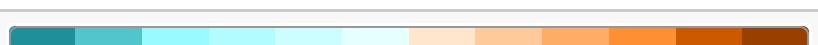
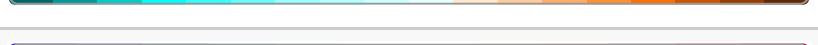
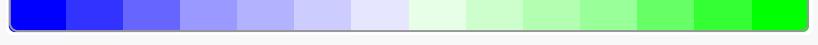
L'extension `paletteer` vise à proposer une interface unifiée pour l'utilisation de palettes de couleurs fournies par d'autres extensions.

`scale_color_paletteer_d` et `scale_color_paletteer_c` permettent de sélectionner une des palettes disponibles avec un usage avec `ggplot2`. Il existe bien entendu leur équivalent pour l'esthétique `fill`.

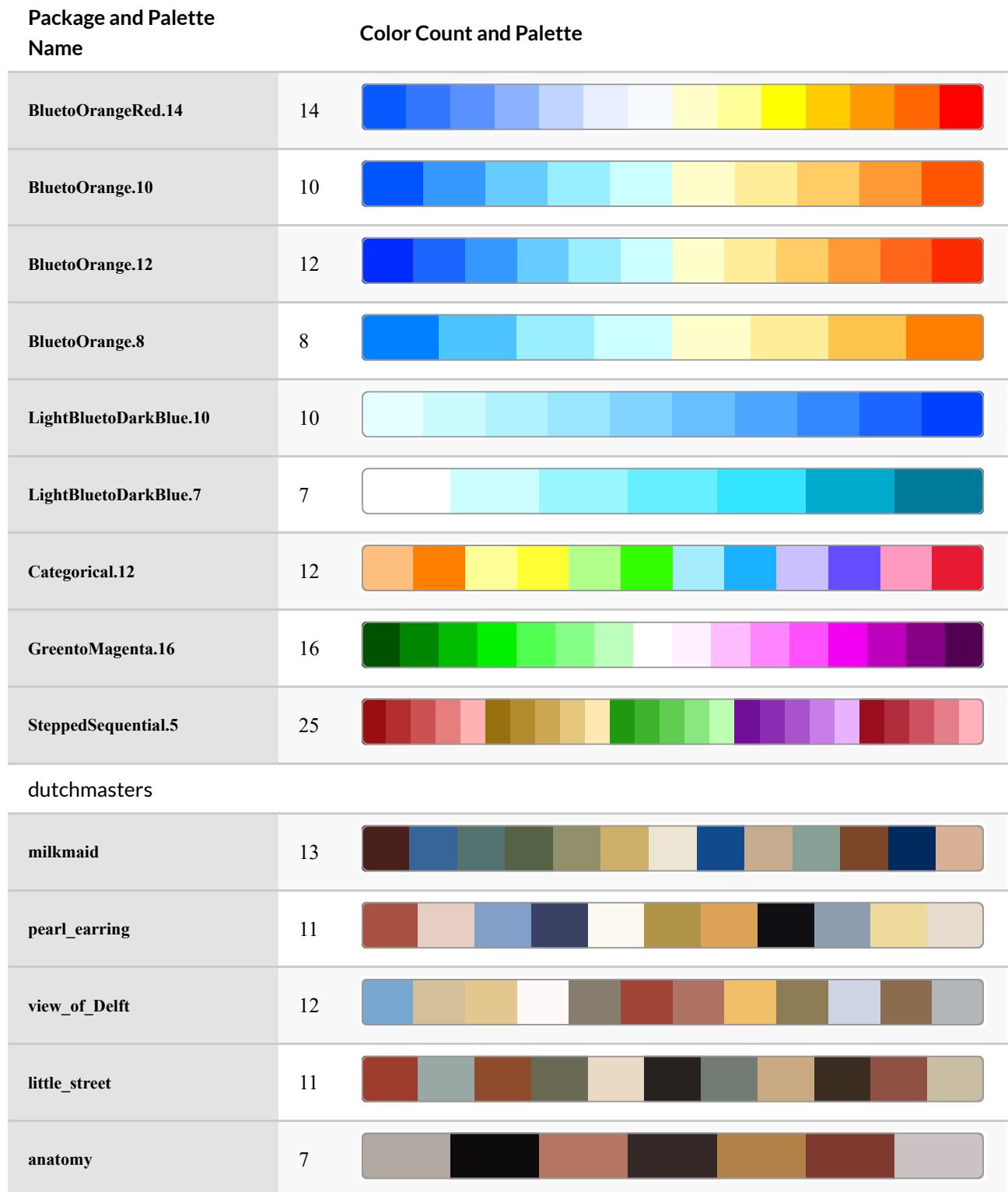
```
R> library(paletter)
R> ggplot(iris) +
  aes(Sepal.Length, Sepal.Width, color = Species) +
  geom_point() +
  scale_color_paletter("nord::aurora")
```



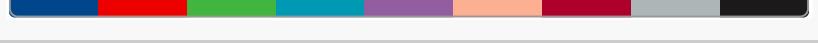
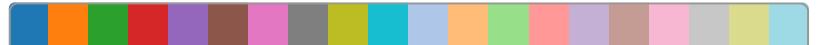
Voici une liste de palettes disponibles via [paletter](#) :

Package and Palette Name	Color Count and Palette
awtools	
a_palette	8 
bpalette	16 
gpalette	4 
mpalette	9 
spalette	6 
dichromat	
BrownToBlue.10	10 
BrownToBlue.12	12 
BluetoDarkOrange.12	12 
BluetoDarkOrange.18	18 
DarkRedtoBlue.12	12 
DarkRedtoBlue.18	18 
BluetoGreen.14	14 
BluetoGray.8	8 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
staalmeesters	7 
ggsci	
nrc_npg	10 
default_aaas	10 
default_nejm	8 
lanonc_lancet	9 
default_jama	7 
default_jco	10 
default_ucscgb	26 
category10_d3	10 
category20_d3	20 
category20b_d3	20 
category20c_d3	20 
default_igv	51 
alternating_jgv	2 

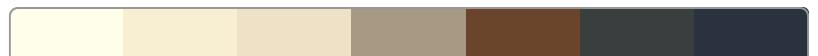
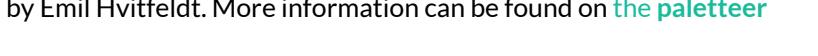
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



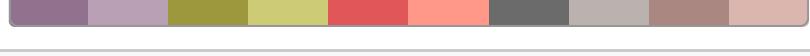
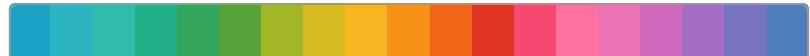
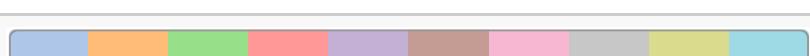
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
purple_material	10 
deep-purple_material	10 
indigo_material	10 
blue_material	10 
light-blue_material	10 
cyan_material	10 
teal_material	10 
green_material	10 
light-green_material	10 
lime_material	10 
yellow_material	10 
amber_material	10 
orange_material	10 
deep-orange_material	10 
brown_material	10 

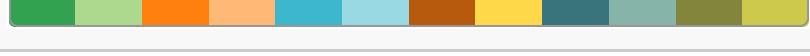
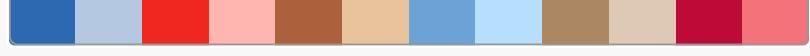
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
grey_material	10 
blue-grey_material	10 
ggpomological	
pomological_base	7 
pomological_palette	9 
ggthemes	
calc	12 
manyeyes	19 
gdoc	10 
fivethirtyeight	6 
colorblind	8 
Tableau 10	10 
Tableau 20	20 
Color Blind	10 
Seattle Grays	5 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
Traffic	9 
Miller Stone	11 
Superfishel Stone	10 
Nuriel Stone	9 
Jewel Bright	9 
Summer	8 
Winter	10 
Green-Orange-Teal	12 
Red-Blue-Brown	12 
Purple-Pink-Gray	12 
Hue Circle	19 
Classic 10	10 
Classic 10 Medium	10 
Classic 10 Light	10 
Classic 20	20 

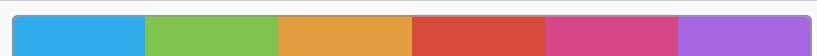
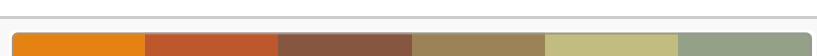
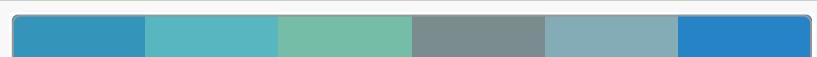
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
Classic Gray 5	5 
Classic Color Blind	10 
Classic Traffic Light	9 
Classic Purple-Gray 6	6 
Classic Purple-Gray 12	12 
Classic Green-Orange 6	6 
Classic Green-Orange 12	12 
Classic Blue-Red 6	6 
Classic Blue-Red 12	12 
Classic Cyclic	13 
few_Light	9 
few_Medium	9 
few_Dark	9 
excel_Atlas	6 
excel_Badge	6 

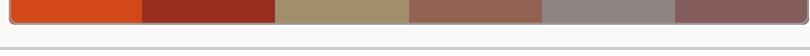
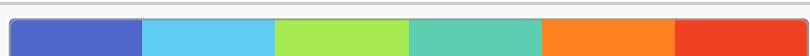
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
excel_Berlin	6 
excel_Celestial	6 
excel_Crop	6 
excel_Depth	6 
excel_Droplet	6 
excel_Facet	6 
excel_Feathered	6 
excel_Gallery	6 
excel_Headlines	6 
excel_Integral	6 
excel_Ion_Boardroom	6 
excel_Ion	6 
excel_Madison	6 
excel_Main_Event	6 
excel_Mesh	6 

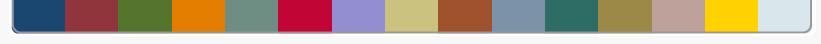
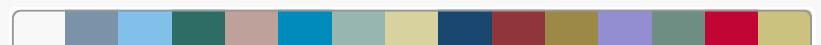
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
excel_Office_Theme	6 
excel_Organic	6 
excel_Parallax	6 
excel_Parcel	6 
excel_Retrospect	6 
excel_Savon	6 
excel_Slice	6 
excel_Vapor_Trail	6 
excel_View	6 
excel_Wisp	6 
excel_Wood_Type	6 
excel_Aspect	6 
excel_Blue_Green	6 
excel_Blue_II	6 
excel_Blue_Warm	6 

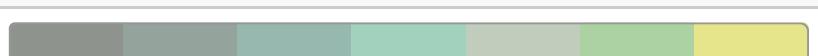
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
excel_Blue	6 
excel_Gayscale	6 
excel_Green_Yellow	6 
excel_Green	6 
excel_Marquee	6 
excel_Median	6 
excel_Office_2007-2010	6 
excel_Orange_Red	6 
excel_Orange	6 
excel_Paper	6 
excel_Red_Orange	6 
excel_Red_Violet	6 
excel_Red	6 
excel_Slipstream	6 
excel_Violet_II	6 

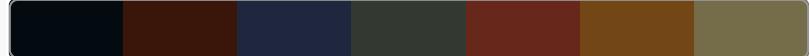
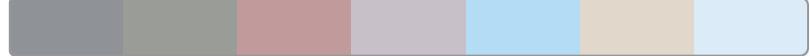
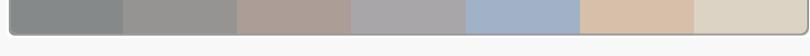
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
excel_Violet	6 
excel_Yellow_Orange	6 
excel_Yellow	6 
wsj_rgbv	4 
wsj_red_green	2 
wsj_black_green	4 
wsj_dem_rep	3 
wsj_colors6	6 
stata_s2color	15 
stata_s1rcolor	15 
stata_s1color	15 
stata_mono	15 
stata_economist	15 
hc_default	10 
hc_darkunica	11 

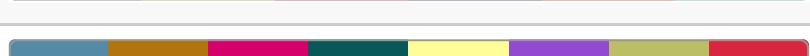
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
hc_bg	5 
hc_fg	12 
<b>ghibli</b>	
MarnieLight1	7 
MarnieMedium1	7 
MarnieDark1	7 
MarnieLight2	7 
MarnieMedium2	7 
MarnieDark2	7 
PonyoLight	7 
PonyoMedium	7 
PonyoDark	7 
LaputaLight	7 
LaputaMedium	7 
LaputaDark	7 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
MononokeLight	7 
MononokeMedium	7 
MononokeDark	7 
SpiritedLight	7 
SpiritedMedium	7 
SpiritedDark	7 
YesterdayLight	7 
YesterdayMedium	7 
YesterdayDark	7 
KikiLight	7 
KikiMedium	7 
KikiDark	7 
TotoroLight	7 
TotoroMedium	7 
TotoroDark	7 

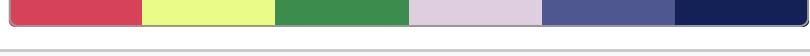
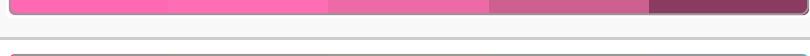
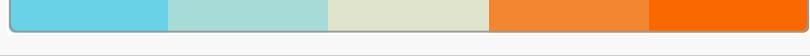
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
grDevices	
blues9	9 
jcolors	
default	5 
pal2	5 
pal3	5 
pal4	6 
pal5	6 
pal6	8 
pal7	8 
pal8	12 
pal9	6 
pal10	10 
pal11	12 
pal12	13 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



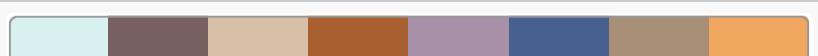
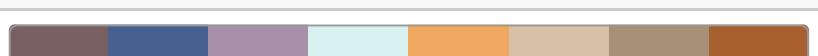
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
KeyLime	5 
PommeBaya	5 
CeriseLimon	6 
PinaFraise	6 
KiwiSandia	7 
MelonPomelo	6 
MurePepino	6 
paired	14 
<b>NineteenEightyR</b>	
cobra	6 
electronic_night	5 
hotpink	5 
malibu	5 
miami1	5 
miami2	5 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



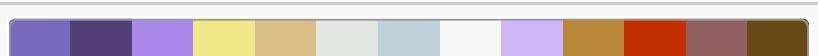
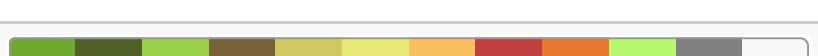
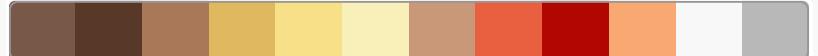
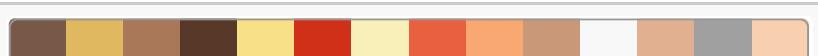
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
victory_bonds	5 
halifax_harbor	6 
moose_pond	8 
algoma_forest	7 
rocky_mountain	6 
red_mountain	8 
baie_mouton	7 
afternoon_prairie	9 
<hr/>	
ochRe	
namatjira_qual	8 
namatjira_div	8 
mccrea	11 
parliament	8 
tasmania	7 
nolan_ned	5 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



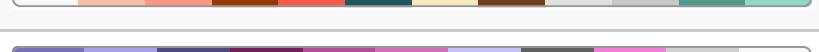
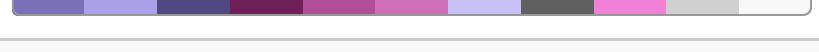
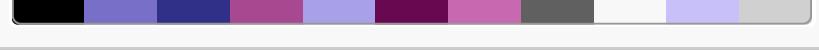
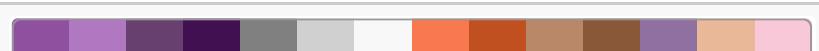
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
charmeleon	12 
charizard	14 
squirtle	14 
wartortle	13 
blastoise	14 
caterpie	12 
metapod	5 
butterfree	13 
weedle	11 
kakuna	8 
beedrill	14 
pidgey	12 
pidgeotto	13 
pigeot	14 
rattata	13 

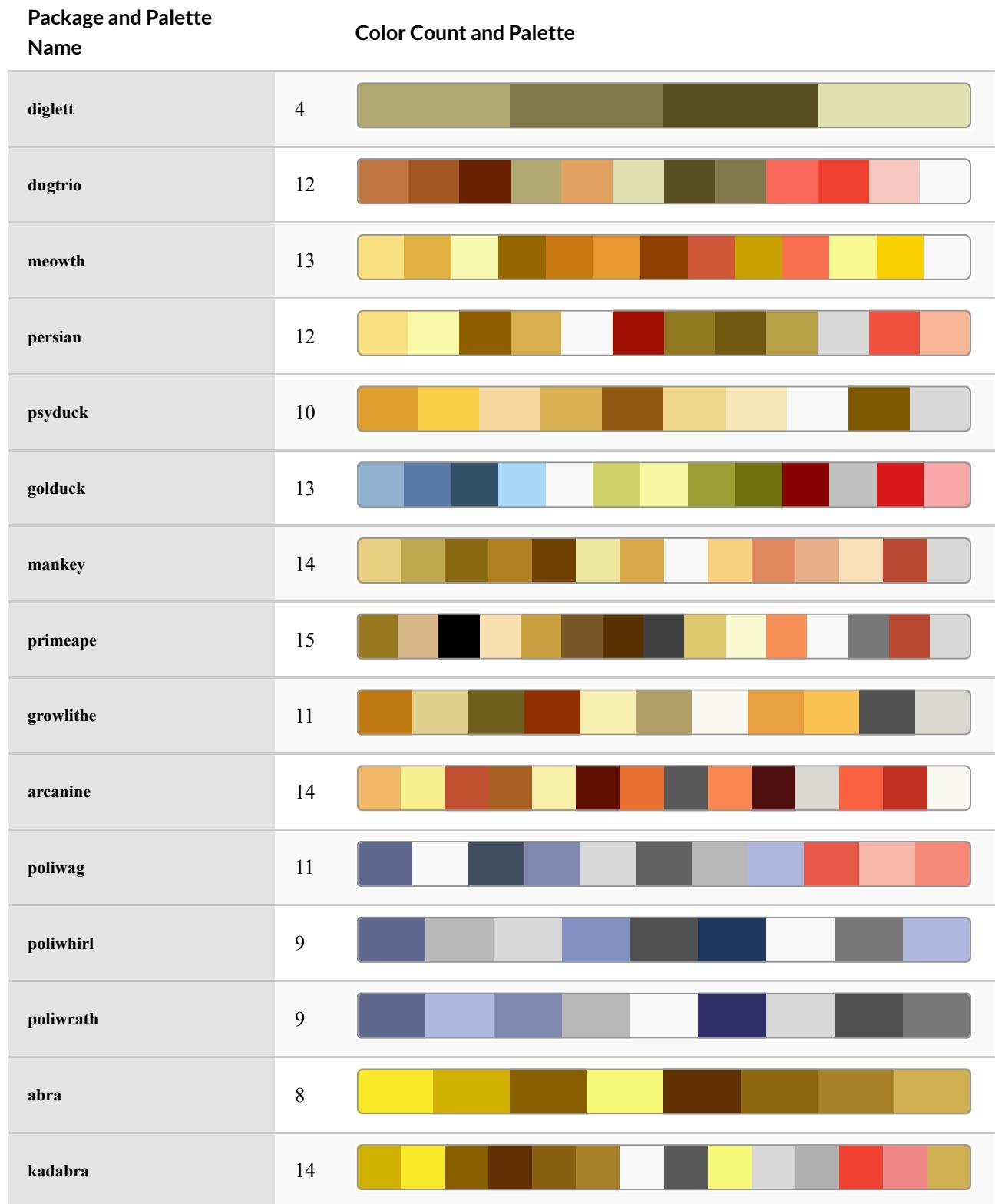
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



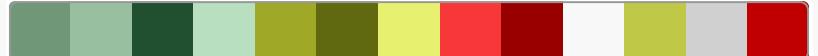
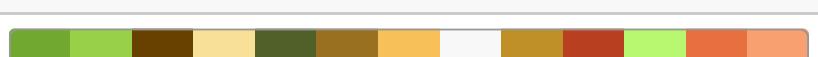
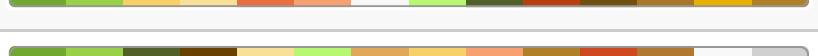
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
clefairy	13 
clefable	12 
vulpix	13 
ninetales	7 
jigglypuff	11 
wigglytuff	12 
zubat	11 
golbat	11 
oddish	10 
gloom	14 
vileplume	13 
paras	12 
parasect	11 
venonat	14 
venomoth	11 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



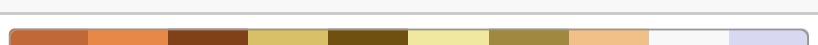
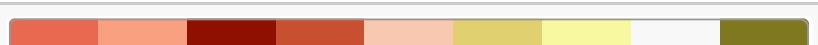
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
alakazam	12 
machop	13 
machoke	13 
machamp	13 
bellsprout	13 
weepinbell	14 
victreebel	14 
tentacool	14 
tentacruel	14 
geodude	6 
graveler	6 
golem	12 
ponyta	11 
rapidash	11 
slowpoke	12 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



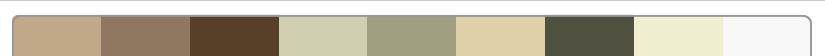
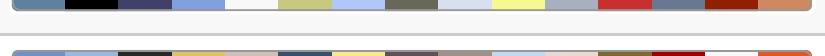
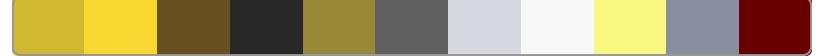
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
onix	6 
drowzee	9 
hypno	9 
krabby	10 
kingler	10 
voltorb	9 
electrode	10 
exeggute	9 
exeggcutor	14 
cubone	13 
marowak	14 
hitmonlee	12 
hitmonchan	13 
lickitung	9 
koffing	12 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
electabuzz	10 
magmar	12 
pinsir	9 
tauros	13 
magikarp	13 
gyarados	15 
lapras	15 
ditto	6 
eevee	9 
vaporeon	15 
jolteon	11 
flareon	10 
polygon	11 
omanyte	12 
omastar	12 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



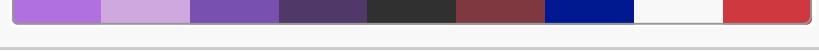
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
cyndaquil	13 
quilava	14 
typhlosion	14 
totodile	11 
croconaw	13 
feraligatr	14 
sentret	12 
furret	11 
hoothoot	13 
noctowl	13 
ledyba	12 
ledian	14 
spinarak	14 
ariados	14 
crobat	9 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



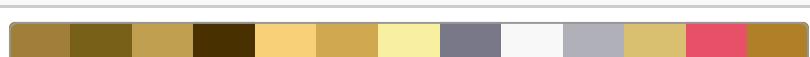
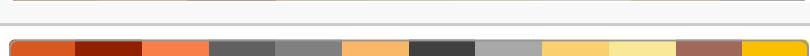
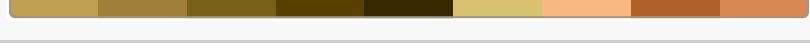
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
sudowoodo	11 
politoed	14 
hoppip	10 
skiploom	11 
jumpluff	13 
aipom	12 
sunkern	13 
sunflora	12 
yanma	11 
wooper	9 
quagsire	9 
espeon	9 
umbreon	10 
murkrow	12 
slowking	13 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



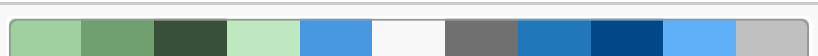
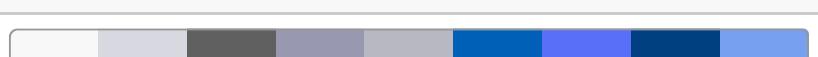
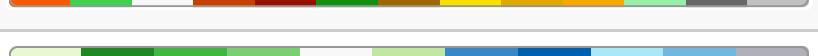
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
sneasel	14 
teddiursa	12 
ursaring	13 
slugma	9 
magcargo	12 
swinub	9 
piloswine	12 
corsola	9 
remoraid	9 
octillery	12 
delibird	13 
mantine	13 
skarmory	12 
houndour	11 
houndoom	12 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



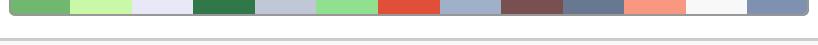
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
suicune	14 
larvitar	9 
pupitar	9 
tyranitar	11 
lugia	9 
ho-oh	13 
celebi	11 
treecko	14 
grovyle	14 
sceptile	13 
torchic	12 
combusken	14 
blaziken	14 
mudkip	14 
marshomp	14 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



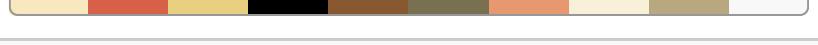
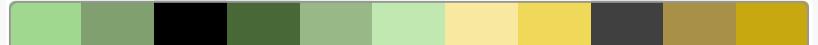
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
shiftry	13 
taillow	15 
swellow	15 
wingull	14 
pelipper	15 
ralts	11 
kirlia	13 
gardevoir	12 
surskit	12 
masquerain	14 
shroomish	11 
breloom	15 
slakoth	14 
vigoroth	15 
slaking	14 

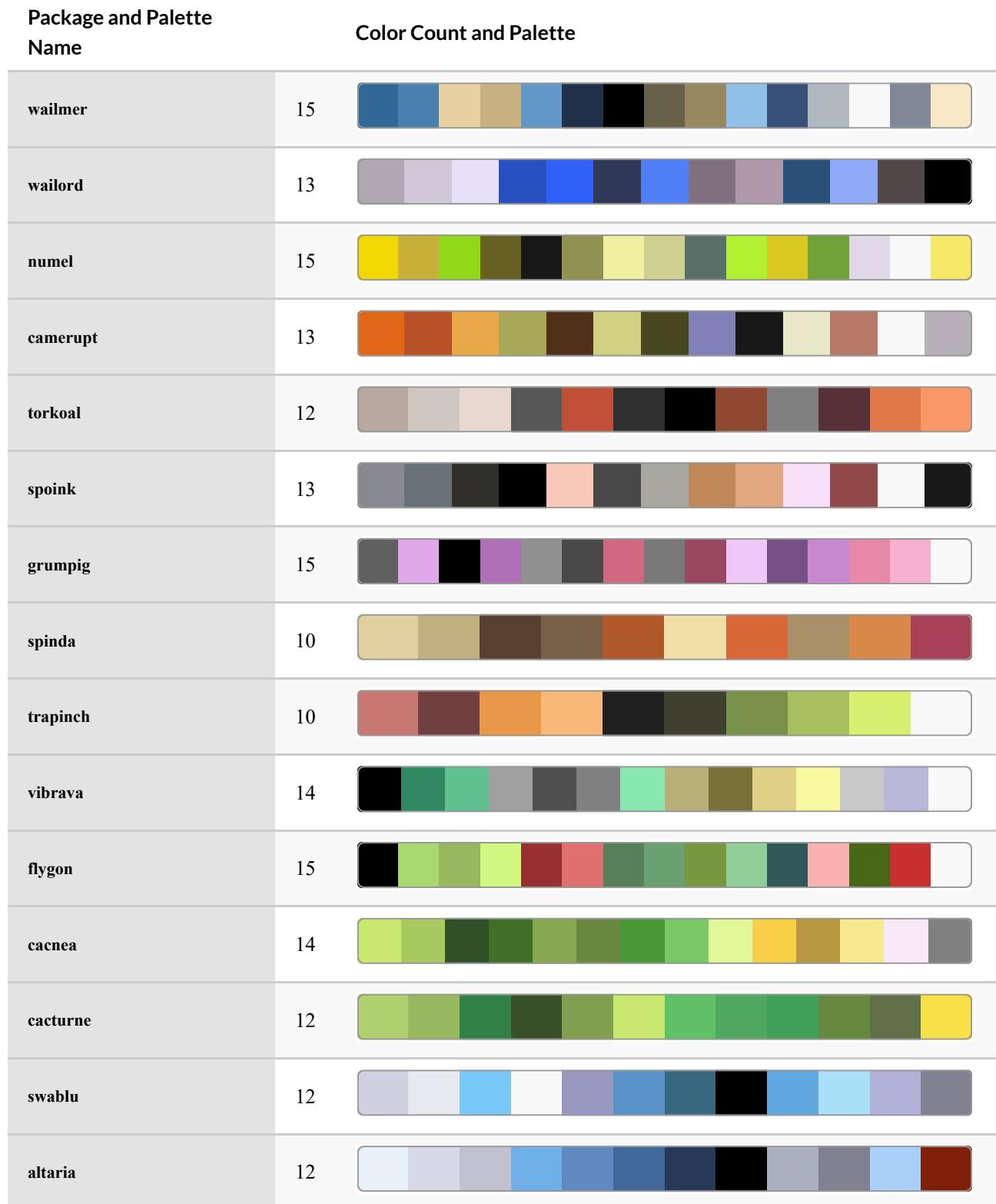
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
lairon	12 
aggron	12 
meditite	13 
medicham	13 
electrike	10 
manectric	13 
plusle	10 
minun	13 
volbeat	15 
illumise	15 
roselia	15 
gulpin	11 
swalot	14 
carvanha	13 
sharpedo	14 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



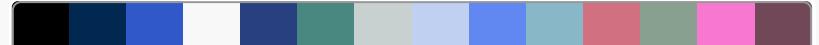
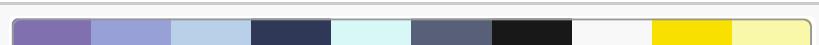
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
zangoose	15 
seviper	14 
lunatone	14 
solrock	14 
barboach	12 
whiscash	15 
corphish	13 
crawdaunt	15 
baltoy	7 
claydol	12 
lileep	12 
cradily	12 
anorith	14 
armaldo	14 
feebas	13 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



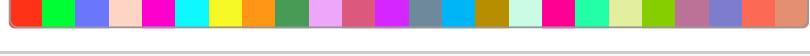
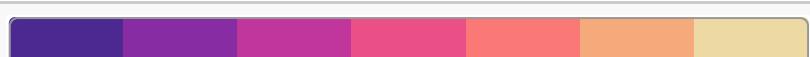
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
walrein	12 
clamperl	14 
huntail	14 
gorebyss	12 
relicanth	13 
luvdisc	9 
bagon	11 
shelgon	12 
salamence	13 
beldum	10 
metang	13 
metagross	13 
regirock	13 
regice	10 
registeele	13 

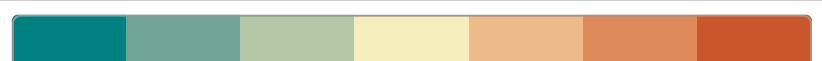
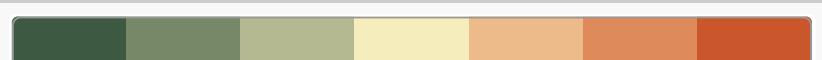
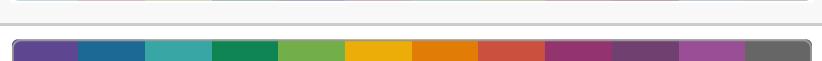
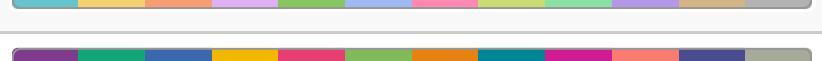
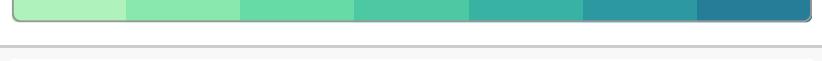
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
latias	14 
latios	14 
kyogre	14 
groudon	14 
rayquaza	12 
jirachi	12 
deoxys	14 
teamrocket	6 
starterspairs	6 
startersDark	6 
pals	
alphabet	26 
alphabet2	26 
glasbey	32 
kelly	22 

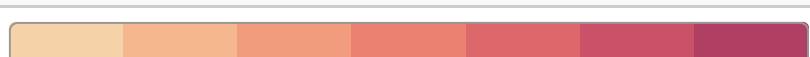
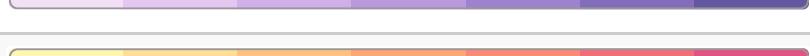
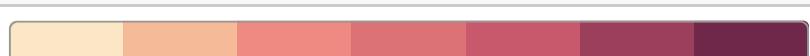
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
polychrome	36 
stepped	24 
tol	12 
watlington	16 
<b>Polychrome</b>	
alphabet	26 
dark	24 
glasbey	32 
green.armytague	26 
kelly	22 
light	24 
palette36	36 
<b>rcartocolor</b>	
ag_Sunset	7 
ag_GrnYl	7 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
Tropic	7 
Temps	7 
TealRose	7 
Geyser	7 
Fall	7 
Earth	7 
ArmyRose	7 
Vivid	12 
Safe	12 
Prism	12 
Pastel	12 
Bold	12 
Antique	12 
TealGrn	7 
Teal	7 

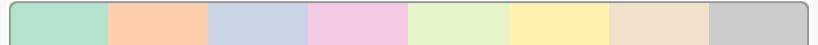
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
SunsetDark	7 
Sunset	7 
RedOr	7 
PurpOr	7 
Purp	7 
PinkYl	7 
Peach	7 
OrYel	7 
Mint	7 
Magenta	7 
Emrlid	7 
DarkMint	7 
BurgYl	7 
Burg	7 
BrwnYl	7 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



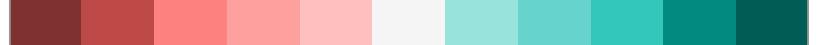
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
Pastell	9 
Pastel2	8 
Set1	9 
Set2	8 
Set3	12 
Blues	9 
BuGn	9 
BuPu	9 
GnBu	9 
Greens	9 
Greys	9 
Oranges	9 
OrRd	9 
PuBu	9 
PuBuGn	9 

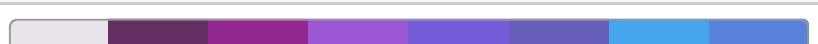
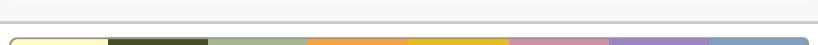
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
dPBIRdGn	11 
qMSOStd	10 
qMSO12	8 
qMSO15	8 
qMSOBuWarm	8 
qMSOBu	8 
qMSOBu2	8 
qMSOBuGn	8 
qMSOGn	8 
qMSOGnYl	8 
qMSOYl	8 
qMSOYlOr	8 
qMSOOrr	8 
qMSOOrrRd	8 
qMSORdOr	8 

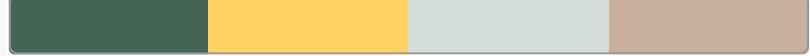
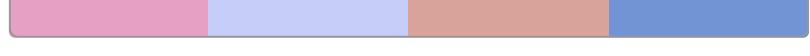
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
qMSORd	8 
qMSORdPu	8 
qMSOPu	8 
qMSOPu2	8 
qMSOMed	8 
qMSOPap	8 
qMSOMrq	8 
qMSOSlp	8 
qMSOAsp	8 
qPBI	8 
sPBIGn	9 
sPBIGy1	9 
sPBIRd	9 
sPBIVI	9 
sPBIGy2	9 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



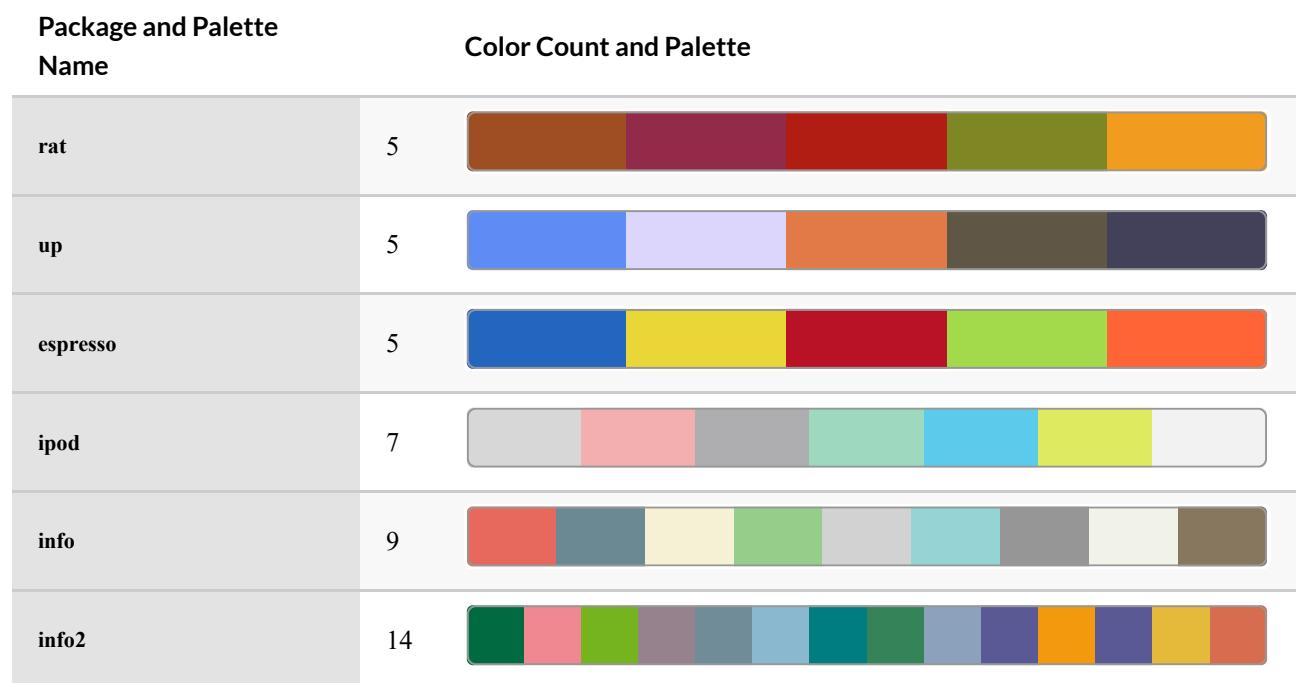
The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
Royal2	5 
Zissou1	5 
Darjeeling1	5 
Darjeeling2	5 
Chevalier1	4 
FantasticFox1	5 
Moonrise1	4 
Moonrise2	4 
Moonrise3	5 
Cavalcanti1	5 
GrandBudapest1	4 
GrandBudapest2	4 
IsleofDogs1	6 
IsleofDogs2	5 
yarr	

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

Package and Palette Name	Color Count and Palette
basel	10 
pony	9 
xmen	8 
decision	6 
southpark	6 
google	4 
eternal	7 
evildead	6 
usualsuspects	7 
ohbrother	7 
appletv	6 
brave	5 
bugs	5 
cars	5 
nemo	5 

The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).



The **paletteer** package is maintained by Emil Hvitfeldt. More information can be found on [the paletteer site](#) and on the [CRAN info page](#).

# Annotations mathématiques

Combiner texte et expression . . . . .	1097
Intégrer une valeur calculée dans une expression . . . . .	1098
Syntaxe de <code>plotmath</code> . . . . .	1100

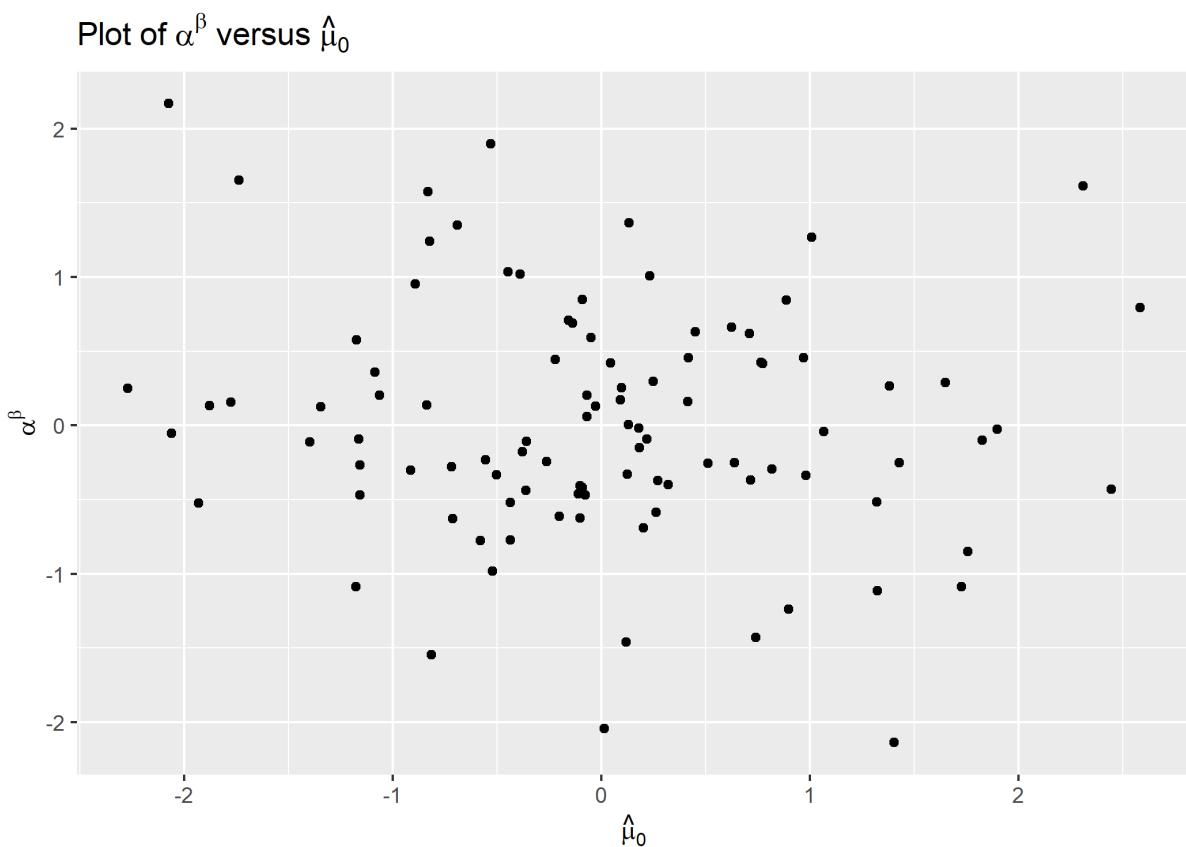
Pour ajouter des annotation mathématiques à un graphique, comme une équation, on aura recours à la fonction `expression`. Les expressions qui peuvent être utilisées sont présentées en détail dans l'aide en ligne de `plotmath`, visible également sur <http://www.rdocumentation.org/packages/grDevices/functions/plotmath>.

## Combiner texte et expression

On aura recours à la fonction `paste` à l'intérieur de l'appel à `expression`. Un exemple :

```
R> # données aléatoires
df <- data.frame(x = rnorm(100), y = rnorm(100))

library(ggplot2)
ggplot(df) +
  aes(x = x, y = y) +
  geom_point() +
  xlab(expression(hat(mu)[0])) +
  ylab(expression(alpha^beta)) +
  ggtitle(expression(paste("Plot of ", alpha^beta, " versus ", hat(mu)[0])))
```

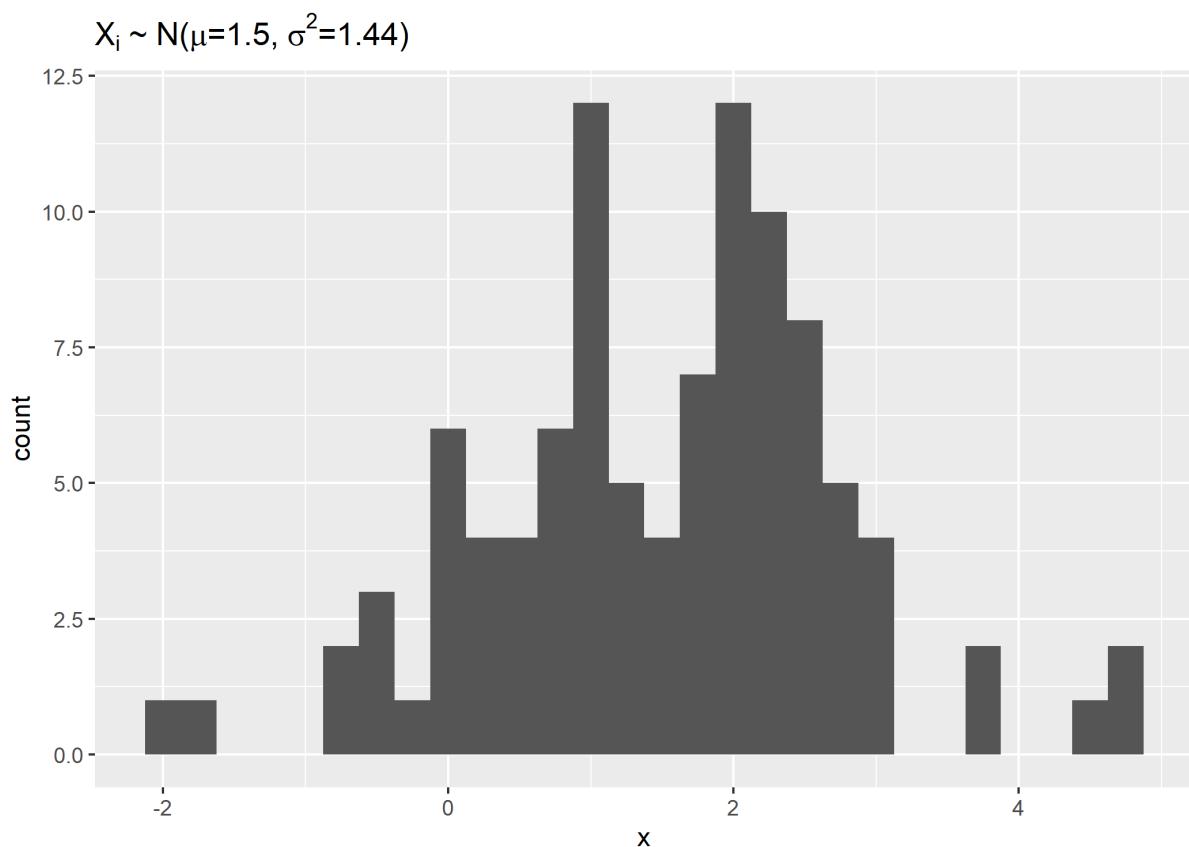


## Intégrer une valeur calculée dans une expression

Pour intégrer une valeur pré-calculée, et donc stockée dans un objet R, dans une expression, on aura recours à la fonction [substitute](#).

```
R> x_mean <- 1.5
  x_sd <- 1.2
  df <- data.frame(x = rnorm(100, x_mean, x_sd))

  ggplot(df) +
    aes(x = x) +
    geom_histogram(binwidth = .25) +
    ggtitle(
      substitute(
        paste(X[i], " ~ N(", mu, "=", m, ", ", sigma^2, "=", s2, ")"),
        list(m = x_mean, s2 = x_sd^2)
      )
    )
```



# Syntaxe de plotmath

```
R> demo(plotmath)
```

Arithmetic Operators		Lists	
$x + y$	$x+y$	<code>list(x, y, z)</code>	$x, y, z$
Relations			
$x - y$	$x-y$	$x == y$	$x = y$
$x * y$	$xy$	$x != y$	$x \neq y$
$x/y$	$x/y$	$x < y$	$x < y$
$x \%+ \% y$	$x \pm y$	$x <= y$	$x \leq y$
$x\%/\%y$	$x \div y$	$x > y$	$x > y$
$x \% \% y$	$x \times y$	$x >= y$	$x \geq y$
$x \%.\% y$	$x \cdot y$	$x \%{\sim}\% y$	$x \approx y$
$-x$	$-x$	$x \%{=}\% y$	$x \equiv y$
$+x$	$+x$	$x \%{==}\% y$	$x \equiv y$
Sub/Superscripts		$x \equiv y$	
$x[i]$	$x_i$	$x \%{\text{prop}} \% y$	$x \propto y$
$x^2$	$x^2$	$x \%{\sim}\% y$	$x \sim y$
Juxtaposition		Typeface	
$x * y$	$xy$	<code>plain(x)</code>	$x$
<code>paste(x, y, z)</code>	$xyz$	<code>italic(x)</code>	$x$
Radicals		$x$	
$\sqrt{x}$	$\sqrt{x}$	<code>bolditalic(x)</code>	$x$
$\sqrt[x]{y}$	$\sqrt[x]{y}$	<code>underline(x)</code>	$x$

Ellipsis		Arrows	
<code>list(x[1], ..., x[n])</code>	$x_1, \dots, x_n$	$x \%{<->}\% y$	$x \leftrightarrow y$
$x[1] + \dots + x[n]$	$x_1 + \dots + x_n$	$x \%{->}\% y$	$x \rightarrow y$
<code>list(x[1], cdots, x[n])</code>	$x_1, \dots, x_n$	$x \%{<-}\% y$	$x \leftarrow y$
$x[1] + ldots + x[n]$	$x_1 + \dots + x_n$	$x \%{up}\% y$	$x \uparrow y$
Set Relations		$x \downarrow y$	
$x \%subset\% y$	$x \subset y$	$x \%{<=}\% y$	$x \leftrightarrow y$
$x \%subseq\% y$	$x \subseteq y$	$x \%{=>}\% y$	$x \Rightarrow y$
$x \%supset\% y$	$x \supset y$	$x \%{<=}\% y$	$x \Leftarrow y$
$x \%supseteqq\% y$	$x \supseteq y$	<code>boldup(y)</code>	$x \uparrow y$
$x \%notsubset\% y$	$x \not\subset y$	<code>bolddown(y)</code>	$x \downarrow y$
$x \%in\% y$	$x \in y$	Symbolic Names	
$x \%notin\% y$	$x \notin y$	Alpha - Omega	$A - \Omega$
Accents		alpha - omega	$\alpha - \omega$
<code>hat(x)</code>	$\hat{x}$	$\phi_1 + \sigma_1$	$\varphi + \varsigma$
<code>tilde(x)</code>	$\tilde{x}$	<code>Upsilon1</code>	$\Upsilon$
<code>ring(x)</code>	$\ddot{x}$	infinity	$\infty$
<code>bar(xy)</code>	$\overline{xy}$	32 * degree	$32^\circ$
<code>widehat(xy)</code>	$\widehat{xy}$	60 * minute	$60'$
<code>widetilde(xy)</code>	$\widetilde{xy}$	30 * second	$30''$

Style	
displaystyle(x)	x
textstyle(x)	x
scriptstyle(x)	x
scriptscriptstyle(x)	x
Spacing	
x ~ ~y	x y

Fractions	
x + phantom(0) + y	x + +y
x + over(1, phantom(0))	$\frac{1}{x}$
frac(x, y)	$\frac{x}{y}$
over(x, y)	$\frac{x}{y}$
atop(x, y)	$\frac{x}{y}$

Big Operators	
sum(x[i], i = 1, n)	$\sum_1^n x_i$
prod(plain(P)(X == x), x)	$\prod_x P(X=x)$
integral(f(x) * dx, a, b)	$\int_a^b f(x)dx$
union(A[i], i == 1, n)	$\bigcup_{i=1}^n A_i$
intersect(A[i], i == 1, n)	$\bigcap_{i=1}^n A_i$
lim(f(x), x %->% 0)	$\lim_{x \rightarrow 0} f(x)$
min(g(x), x >= 0)	$\min_{x \geq 0} g(x)$
inf(S)	$\inf S$
sup(S)	$\sup S$

Grouping	
$\{x, y\}$	$(x, y)$
$(x + y) * z$	$(x + y)z$
$x^y + z$	$x^y + z$
$x^y(y + z)$	$x^{(y+z)}$
$x^y\{y + z\}$	$x^{y+z}$
group("(", list(a, b), ")")	$(a, b]$
bgroup("(", atop(x, y), ")")	$\begin{pmatrix} x \\ y \end{pmatrix}$
group(lceil, x, rceil)	$\lceil x \rceil$
group(lfloor, x, rfloor)	$\lfloor x \rfloor$
group("  ", x, "  ")	$ x $

# Calculer un âge

Rappel sur les âges . . . . .	1103
Le package lubridate . . . . .	1104
Calcul d'un âge exact . . . . .	1104
Cas particulier des personnes nées un 29 février . . . . .	1106
Âge révolu ou âge au dernier anniversaire . . . . .	1108
Âge par différence de millésimes . . . . .	1108
Calcul d'un âge moyen . . . . .	1109
Notes . . . . .	1109

## NOTE

La version originale de cette astuce a été publiée par Joseph Larmarange sur <http://joseph.larmarange.net/?Calculer-proprement-un-age-sous-R>.

Le calcul d'un âge sous R n'est pas forcément aussi trivial qu'il n'y paraît.

## Rappel sur les âges

Il convient en premier lieu de rappeler les principaux âges utilisés les démographes :

*L'âge - on précise parfois âge chronologique - est une des caractéristiques fondamentales de la structure des populations. On l'exprime généralement en années, ou en années et mois, voire en mois et jours, pour les enfants en bas âge ; parfois en années et fractions décimales d'année. Les démographes arrondissent d'ordinaire l'âge à l'unité inférieure, l'exprimant ainsi en années révolues, ou années accomplies, le cas échéant en mois révolus, ou mois accomplis. Cet âge est aussi*

*l'âge au dernier anniversaire. On trouve aussi, dans les statistiques, l'âge atteint dans l'année, qui est égal à la différence de millésimes entre l'année considérée et l'année de naissance. [...] On est parfois conduit à préciser que l'on considère un âge exact, pour éviter toute confusion avec un âge en années révolues, qui représente en fait une classe d'âges exacts.*

— Source : [Demopædia \(322\)](#)

## Le package lubridate

Le package **lubridate** est spécialement développé pour faciliter la manipulation et le calcul autour des dates. La version de développement intègre depuis peu une fonction `time_length` adaptée au calcul des âges exacts.

### NOTE

La fonction `time_length` étant récente, elle n'est pas encore disponible dans la version stable du package. Pour installer la version de développement de **lubridate**, on aura recours au package **devtools** :

```
R> library(devtools)
install_github("tidyverse/lubridate")
```

Nous noterons `naiss` la date de naissance et `evt` la date à laquelle nous calculerons l'âge.

## Calcul d'un âge exact

On appelle âge exact l'expression d'un âge avec sa partie décimale.

Une approche simple consiste à calculer une différence en jours puis à diviser par 365. Or, le souci c'est que toutes les années n'ont pas le même nombre de jours. Regardons par exemple ce qui se passe si l'on calcule l'âge au 31 décembre 1999 d'une personne née le 1<sup>er</sup> janvier 1900.

```
R> library(lubridate)
naiss <- ymd("1900-01-01")
evt <- ymd("1999-12-31")
time_length(interval(naiss, evt), "days")
```

[1] 36523

```
R> time_length(interval(naiss, evt), "days") / 365
```

[1] 100.063

Or, au 31 décembre 1999, cette personne n'a pas encore fêté son centième anniversaire. Le calcul précédent ne prend pas en compte les années bissextiles. Une approche plus correcte serait de considérer que les années durent en moyenne 365,25 jours.

```
R> time_length(interval(naiss, evt), "days") / 365.25
```

[1] 99.99452

Si cette approche semble fonctionner avec cet exemple, ce n'est plus le cas dans d'autres situations.

```
R> evt <- ymd("1903-01-01")
time_length(interval(naiss, evt), "days") / 365.25
```

[1] 2.997947

Or, à la date du premier janvier 1903, cette personne a bien fêté son troisième anniversaire.

Pour calculer proprement un âge en années (ou en mois), il est dès lors nécessaire de prendre en compte la date anniversaire et le fait que la durée de chaque année (ou mois) est variable. C'est justement ce que fait la fonction `time_length` appliquée à un objet de type `Interval`. On détermine le dernier et le prochain anniversaire et l'on rajoute, à l'âge atteint au dernier anniversaire, le ratio entre le nombre de jours entre l'événement et le dernier anniversaire par le nombre de jours entre le prochain et le dernier anniversaire.

```
R> naiss <- ymd("1900-01-01")
evt <- ymd("1999-12-31")
time_length(interval(naiss, evt), "years")
```

[1] 99.99726

```
R> evt <- ymd("1903-01-01")
  time_length(interval(naiss, evt), "years")
```

```
[1] 3
```

```
R> evt <- ymd("1918-11-11")
  time_length(interval(naiss, evt), "years")
```

```
[1] 18.86027
```

Attention, cela n’est valable que si l’on présente à la fonction `time_length` un objet de type `Interval` (pour lequel on connaît dès lors la date de début et la date de fin). Si l’on passe une durée (objet de type `Duration`) à la fonction `time_length`, le calcul s’effectuera alors en prenant en compte la durée moyenne d’une année (plus précisément 365 jours).

```
R> naiss <- ymd("1900-01-01")
  evt <- ymd("1999-12-31")
  time_length(interval(naiss, evt), "years")
```

```
[1] 99.99726
```

```
R> time_length(evt - naiss, "years")
```

```
[1] 99.99452
```

```
R> time_length(as.duration(interval(naiss, evt)), "years")
```

```
[1] 99.99452
```

## Cas particulier des personnes nées un 29 février

Pour les personnes nées un 29 février, il existe un certain flou concernant leur date d’anniversaire pour les années non bissextiles. Doit-on considérer qu’il s’agit du 28 février ou du 1<sup>er</sup> mars ?

Au sens strict, on peut considérer que leur anniversaire a lieu entre le 28 février soir à minuit et le 1<sup>er</sup> mars à 0 heure du matin, autrement dit que le 28 février ils n’ont pas encore fêté leur anniversaire. C’est la position adoptée par la fonction `time_length`.

```
R> naiss <- ymd("1992-02-29")
  evt <- ymd("2014-02-28")
  time_length(interval(naiss, evt), "years")
```

[1] 21.99726

```
R> evt <- ymd("2014-03-01")
  time_length(interval(naiss, evt), "years")
```

[1] 22

Cette approche permet également d'être cohérent avec la manière dont les dates sont prises en compte informatiquement. On considère en effet que lorsque seule la date est précisée (sans mention de l'heure), l'heure correspondante est 0:00. Autrement dit, "2014-03-01" est équivalent à "2014-03-01 00:00:00". L'approche adoptée permet donc d'être cohérent lorsque l'anniversaire est calculé en tenant compte des heures.

```
R> naiss <- ymd("1992-02-29")
  evt <- ymd_hms("2014-02-28 23:00:00")
  time_length(interval(naiss, evt), "years")
```

[1] 21.99989

```
R> evt <- ymd_hms("2014-03-01 00:00:00")
  time_length(interval(naiss, evt), "years")
```

[1] 22

```
R> evt <- ymd_hms("2014-03-01 01:00:00")
  time_length(interval(naiss, evt), "years")
```

[1] 22.00011

```
R> naiss <- ymd_hms("1992-02-29 12:00:00")
  evt <- ymd_hms("2014-03-01 01:00:00")
  time_length(interval(naiss, evt), "years")
```

```
[1] 22.00011
```

## Âge révolu ou âge au dernier anniversaire

Une fois que l’on sait calculer un âge exact, le calcul d’un âge révolu ou âge au dernier anniversaire est assez simple. Il suffit de ne garder que la partie entière de l’âge exact (approche conseillée).

```
R> naiss <- ymd("1980-01-09")
  evt <- ymd("2015-01-01")
  time_length(interval(naiss, evt), "years")
```

```
[1] 34.97808
```

```
R> trunc(time_length(interval(naiss, evt), "years"))
```

```
[1] 34
```

Une autre approche consiste à convertir l’intervalle en objet de type `Period` et à ne prendre en compte que les années.

```
R> as.period(interval(naiss, evt))
```

```
[1] "34y 11m 23d 0H 0M 0S"
```

```
R> as.period(interval(naiss, evt))@year
```

```
[1] 34
```

## Âge par différence de millésimes

L’âge par différence de millésimes, encore appelé âge atteint dans l’année, s’obtient tout simplement en

soustrayant l'année de naissance à l'année de l'événement.

```
R> naiss <- ymd("1980-01-09")
  evt <- ymd("2015-01-01")
  year(evt) - year(naiss)
```

```
[1] 35
```

## Calcul d'un âge moyen

Le calcul d'un âge moyen s'effectue normalement à partir d'âges exacts. Il arrive fréquemment que l'on ne dispose dans les données d'enquêtes que de l'âge révolu. Auquel cas, il faut bien penser à rajouter 0,5 au résultat obtenu. En effet, un âge révolu peut être vu comme une classe d'âges exacts : les individus ayant 20 ans révolus ont entre 20 et 21 ans exacts, soit en moyenne 20,5 ans !

## Notes

L'ensemble des fonctions présentées peuvent être appliquées à des vecteurs et, par conséquent, aux colonnes d'un tableau de données (*data.frame*).

En l'absence du package **lubridate**, il reste facile de calculer une durée en jours avec les fonctions de base de R :

```
R> naiss <- as.Date("1900-01-01")
  evt <- as.Date("1999-12-31")
  evt - naiss
```

```
Time difference of 36523 days
```

```
R> as.integer(evt - naiss)
```

```
[1] 36523
```



# Diagramme de Lexis

## IMPORTANT

Ce chapitre est en cours d'écriture.

Pour réaliser des diagrammes de Lexis, voir l'extension **LexisPlotR** et sa vignette (en anglais) :  
<https://cran.r-project.org/web/packages/LexisPlotR/vignettes/LexisPlotR.html>.



# Index des concepts

## A

### ACM

Analyse des correspondances multiples (ACM) . . . . . 557

### ACP

Analyse des correspondances multiples (ACM) . . . . . 557

### AFC

Analyse des correspondances multiples (ACM) . . . . . 557

### âge

Calculer un âge . . . . . 1103

### âge atteint dans l'année

Calculer un âge . . . . . 1103

### âge au dernier anniversaire

Calculer un âge . . . . . 1103

### âge exact

Calculer un âge . . . . . 1103

### AIC

Régression logistique binaire, multinomiale et ordinale . . . . . 507

### aide

Premier contact . . . . . 13

### aide en ligne

Où trouver de l'aide ? . . . . . 163

### Akaike Information Criterion

Analyse de survie . . . . . 651

Régression logistique binaire, multinomiale et ordinale . . . . . 507

### aléatoire, échantillonnage

Définir un plan d'échantillonnage complexe . . . . . 501

### aléatoire, effet

Trajectoires de soins: un exemple de données longitudinales . . . . . 705

### analyse de séquences

Analyse de séquences . . . . . 679

Trajectoires de soins: un exemple de données longitudinales . . . . . 705

### analyse de survie

Analyse de survie . . . . . 651

### analyse des biographies

Analyse de séquences . . . . . 679

### analyse des correspondances multiples

Analyse des correspondances multiples (ACM) . . . . . 557

### analyse en composante principale

Analyse des correspondances multiples (ACM) . . . . . 557

### analyse factorielle

Analyse des correspondances multiples (ACM) . . . . . 557

### analyse factorielle avec données mixtes

Analyse des correspondances multiples (ACM) . . . . . 557

### analyse factorielle des correspondances

Analyse des correspondances multiples (ACM) . . . . . 557

### ANOVA

Régression logistique binaire, multinomiale et ordinale . . . . . 507

### appariement optimal

Analyse de séquences . . . . . 679

### arbre de classification

Classification ascendante hiérarchique (CAH) . . . . . 595

### argument

Premier contact . . . . . 13

<b>argument nommé</b>	
Premier contact . . . . .	13
<b>argument non nommé</b>	
Premier contact . . . . .	13
<b>assignation par indexation</b>	
Listes et Tableaux de données . . . . .	87
<b>assignation, opérateur</b>	
Premier contact . . . . .	13
<b>attribut</b>	
Import de données . . . . .	145
<b>autocomplete</b>	
Organiser ses fichiers . . . . .	135
Premier contact . . . . .	13
Présentation et Philosophie . . . . .	7
 <b>B</b>	
<b>barres cumulées, diagramme en</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>barres cumulées, graphique</b>	
Statistique bivariée . . . . .	365
<b>barres, diagramme en</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>bâton, diagramme</b>	
Statistique univariée . . . . .	339
<b>bâtons, diagramme en</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>Bayesian Highest Density Intervals</b>	
Statistique bivariée . . . . .	365
<b>binnaire, régression logistique</b>	
Régression logistique binaire, multinomiale et ordinaire . . . . .	507
<b>biographie, analyse</b>	
Analyse de séquences . . . . .	679
<b>bitmap</b>	
Export de graphiques . . . . .	331
 <b>C</b>	
<b>boîte à moustache</b>	
Statistique bivariée . . . . .	365
<b>boîte à moustaches</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Statistique univariée . . . . .	339
<b>booléenne, valeur</b>	
Vecteurs, indexation et assignation . . . . .	67
<b>booléenne, variable</b>	
Régression logistique binaire, multinomiale et ordinaire . . . . .	507
<b>boxplot</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Statistique bivariée . . . . .	365
Statistique univariée . . . . .	339
<b>CAH</b>	
Analyse de séquences . . . . .	679
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>camembert, graphique</b>	
Statistique univariée . . . . .	339
<b>caractères, chaîne</b>	
Vecteurs, indexation et assignation . . . . .	67
<b>catégorielle, variable</b>	
Facteurs et vecteurs labellisés . . . . .	117
<b>censure à droite</b>	
Analyse de survie . . . . .	651
<b> cercle de corrélation</b>	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>chaîne de caractères</b>	
Vecteurs, indexation et assignation . . . . .	67
<b>character</b>	
Vecteurs, indexation et assignation . . . . .	67
<b>chemin relatif</b>	
Organiser ses fichiers . . . . .	135
<b>Chi<sup>2</sup>, distance</b>	
Classification ascendante hiérarchique (CAH) . . . . .	595

<b>Chi<sup>2</sup>, résidus</b>	
Comparaisons (moyennes et proportions) . . . . .	485
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>Chi<sup>2</sup>, test</b>	
Comparaisons (moyennes et proportions) . . . . .	485
Données pondérées . . . . .	465
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>chunk</b>	
R Markdown : les rapports automatisés . . . . .	979
<b>classe de valeurs</b>	
Statistique univariée . . . . .	339
<b>classe, homogénéité</b>	
Analyse de séquences . . . . .	679
<b>classes latentes, modèle mixte</b>	
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>classification ascendante hiérarchique</b>	
Analyse de séquences . . . . .	679
Classification ascendante hiérarchique (CAH) . . . . .	595
Trajettoires de soins: un exemple de données longitudinales . . . . .	705
<b>classification, arbre</b>	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>Cleveland, diagramme</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Statistique univariée . . . . .	339
<b>cluster</b>	
Définir un plan d'échantillonnage complexe . . . . .	501
<b>coefficient de contingence de Cramer</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>coefficient de corrélation</b>	
Statistique bivariée . . . . .	365
<b>coefficient, modèle</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>coefficients du modèle</b>	
Modèles à effets aléatoires (modèles mixtes et GEE)	
<b>colinéarité</b>	
Multicolinéarité dans la régression . . . . .	645
<b>coloration syntaxique</b>	
Premier contact . . . . .	13
Présentation et Philosophie . . . . .	7
<b>commentaire</b>	
Premier travail avec des données . . . . .	33
<b>comparaison de courbes de survie (test du logrank)</b>	
Analyse de survie . . . . .	651
<b>comparaison de médianes, test</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>comparaison de moyennes</b>	
Comparaisons (moyennes et proportions) . . . . .	485
Données pondérées . . . . .	465
<b>comparaison de proportions, test</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>comparaison, opérateur</b>	
Conditions et comparaisons . . . . .	925
Vecteurs, indexation et assignation . . . . .	67
<b>Comprehensive R Archive Network</b>	
Extensions (installation, mise à jour) . . . . .	55
<b>condition, indexation</b>	
Listes et Tableaux de données . . . . .	87
Vecteurs, indexation et assignation . . . . .	67
<b>confusion, matrice</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>console</b>	
Premier contact . . . . .	13
<b>corrélation</b>	
Multicolinéarité dans la régression . . . . .	645
<b>corrélation, cercle</b>	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>corrélation, coefficient</b>	
Statistique bivariée . . . . .	365
<b>corrélation, matrice de</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>correspondances, analyse factorielle</b>	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>couleur</b>	
Couleurs et Palettes . . . . .	1021

<b>couleur, palette</b>	<b>diagramme de Lexis</b>
Analyse des correspondances multiples (ACM) . . . . .	Diagramme de Lexis . . . . . 1111
<b>courbe de densité</b>	<b>diagramme de Sankey</b>
Graphiques univariés et bivariés avec ggplot2 . . . . .	Autres extensions graphiques . . . . . 897
<b>Cox, modèle</b>	<b>diagramme en barres</b>
Analyse de survie . . . . .	Graphiques univariés et bivariés avec ggplot2 . . . . . 417
<b>Cramer, coefficient de contingence</b>	<b>diagramme en barres cumulées</b>
Comparaisons (moyennes et proportions) . . . . .	Graphiques univariés et bivariés avec ggplot2 . . . . . 417
<b>CRAN</b>	Trajectoires de soins: un exemple de données longitudinales . . . . . 705
Extensions (installation, mise à jour) . . . . .	<b>diagramme en bâtons</b>
<b>croisé, tableau</b>	Graphiques univariés et bivariés avec ggplot2 . . . . . 417
Statistique bivariée . . . . .	Statistique univariée . . . . . 339
<b>CSV, fichier</b>	<b>diagramme en secteur</b>
Import de données . . . . .	Statistique univariée . . . . . 339
<b>D</b>	
<b>data frame</b>	<b>distance</b>
Listes et Tableaux de données . . . . .	Classification ascendante hiérarchique (CAH) . . . . . 595
Premier travail avec des données . . . . .	<b>distance de Gower</b>
	Classification ascendante hiérarchique (CAH) . . . . . 595
<b>data.frame</b>	<b>distance du Chi<sup>2</sup></b>
Import de données . . . . .	Classification ascendante hiérarchique (CAH) . . . . . 595
<b>date, variable</b>	<b>distance du Phi<sup>2</sup></b>
Calculer un âge . . . . .	Classification ascendante hiérarchique (CAH) . . . . . 595
<b>dendrogramme</b>	<b>distance, matrice</b>
Classification ascendante hiérarchique (CAH) . . . . .	Analyse de séquences . . . . . 679
<b>densité, courbe de</b>	<b>distribution</b>
Graphiques univariés et bivariés avec ggplot2 . . . . .	Statistique bivariée . . . . . 365
<b>densité, estimation locale</b>	Statistique univariée . . . . . 339
Graphiques univariés et bivariés avec ggplot2 . . . . .	<b>donnée labelisée</b>
Statistique bivariée . . . . .	Facteurs et vecteurs labellisés . . . . . 117
Statistique univariée . . . . .	<b>données pondérées</b>
<b>descriptive, statistique</b>	Données pondérées . . . . . 465
Statistique bivariée . . . . .	<b>données, exporter</b>
Statistique univariée . . . . .	Export de données . . . . . 327
<b>design</b>	Import de données . . . . . 145
Données pondérées . . . . .	<b>données, tableau</b>
<b>diagramme de Cleveland</b>	Listes et Tableaux de données . . . . . 87
Graphiques univariés et bivariés avec ggplot2 . . . . .	<b>droite de régression</b>
Statistique univariée . . . . .	Graphiques univariés et bivariés avec ggplot2 . . . . . 417

<b>droite, censure</b>	
Analyse de survie . . . . .	651
<b>E</b>	
<b>écart-type</b>	
Statistique univariée . . . . .	339
<b>écart interquartile</b>	
Statistique univariée . . . . .	339
<b>échantillonnage aléatoire simple</b>	
Définir un plan d'échantillonnage complexe . . . . .	501
<b>échantillonnage équiprobable</b>	
Définir un plan d'échantillonnage complexe . . . . .	501
<b>échantillonnage par grappes</b>	
Définir un plan d'échantillonnage complexe . . . . .	501
<b>échantillonnage stratifié</b>	
Définir un plan d'échantillonnage complexe . . . . .	501
<b>échantillonnage, plan</b>	
Données pondérées . . . . .	465
<b>éditeur de script</b>	
Présentation et Philosophie . . . . .	7
<b>effet aléatoire</b>	
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>effet d'interaction</b>	
Effets d'interaction dans un modèle . . . . .	617
<b>empirical cumulative distribution function</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Statistique univariée . . . . .	339
<b>entier</b>	
Premier travail avec des données . . . . .	33
<b>entier, nombre</b>	
Vecteurs, indexation et assignation . . . . .	67
<b>entropie transversale</b>	
Analyse de séquences . . . . .	679
<b>environnement de développement</b>	
Présentation et Philosophie . . . . .	7
<b>environnement de travail</b>	
Premier contact . . . . .	13
<b>équiprobable, échantillonnage</b>	
Définir un plan d'échantillonnage complexe . . . . .	501
<b>erreurs</b>	
Modèles à effets aléatoires (modèles mixtes et GEE)	
<b>estimation locale de densité</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Statistique bivariée . . . . .	365
<b>estimation par noyau</b>	
Statistique univariée . . . . .	339
<b>étendue</b>	
Statistique univariée . . . . .	339
<b>étiquette de valeur</b>	
Facteurs et vecteurs labellisés . . . . .	117
Import de données . . . . .	145
<b>étiquette de variable</b>	
Facteurs et vecteurs labellisés . . . . .	117
Import de données . . . . .	145
<b>étiquettes de valeurs</b>	
Import de données . . . . .	145
<b>event history analysis</b>	
Analyse de séquences . . . . .	679
<b>exact, âge</b>	
Calculer un âge . . . . .	1103
<b>explicative, variable</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>export de graphiques</b>	
Export de graphiques . . . . .	331
Statistique univariée . . . . .	339
<b>exporter des données</b>	
Export de données . . . . .	327
Import de données . . . . .	145
<b>expression régulière</b>	
Expressions régulières . . . . .	977
<b>extension</b>	
Extensions (installation, mise à jour) . . . . .	55

**F****facteur**

Facteurs et vecteurs labellisés . . . . .	117
Import de données . . . . .	145
Premier travail avec des données . . . . .	33
Régression logistique binaire, multinomiale et ordinale . . . . .	507

**facteurs d'inflation de la variance**

Multicolinéarité dans la régression . . . . .	645
-----------------------------------------------	-----

**factor**

Facteurs et vecteurs labellisés . . . . .	117
Premier travail avec des données . . . . .	33

**factoriel, plan**

Analyse des correspondances multiples (ACM) . . . . .	557
-------------------------------------------------------	-----

**factorielle, analyse**

Analyse des correspondances multiples (ACM) . . . . .	557
-------------------------------------------------------	-----

**fichier CSV**

Import de données . . . . .	145
-----------------------------	-----

**fichier de commandes**

Premier travail avec des données . . . . .	33
--------------------------------------------	----

**fichier texte**

Import de données . . . . .	145
-----------------------------	-----

**fichiers Shapefile**

Import de données . . . . .	145
-----------------------------	-----

**Fisher, test exact**

Comparaisons (moyennes et proportions) . . . . .	485
--------------------------------------------------	-----

**FIV**

Multicolinéarité dans la régression . . . . .	645
-----------------------------------------------	-----

**fonction**

Premier contact . . . . .	13
---------------------------	----

**fonction de répartition empirique**

Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Statistique univariée . . . . .	339

**formule**

Régression logistique binaire, multinomiale et ordinale . . . . .	507
Statistique bivariée . . . . .	365

**fréquence, tableau**

Statistique univariée . . . . .	339
---------------------------------	-----

**fusion**

Fusion de tables . . . . .	269
----------------------------	-----

**fusion de tables**

Fusion de tables . . . . .	269
----------------------------	-----

**G****GEE, modèle**

Trajectoires de soins: un exemple de données longitudinales . . . . .	705
-----------------------------------------------------------------------	-----

**gestionnaire de versions**

Organiser ses fichiers . . . . .	135
----------------------------------	-----

**Gower, distance**

Classification ascendante hiérarchique (CAH) . . . . .	595
--------------------------------------------------------	-----

**Gower, indice de similarité**

Classification ascendante hiérarchique (CAH) . . . . .	595
--------------------------------------------------------	-----

**grammaire des tableaux**

R Markdown : les rapports automatisés . . . . .	979
-------------------------------------------------	-----

**graphique de pirates**

Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Statistique bivariée . . . . .	365

**graphique en mosaïque**

Graphiques univariés et bivariés avec ggplot2 . . . . .	417
---------------------------------------------------------	-----

**graphique en violon**

Graphiques univariés et bivariés avec ggplot2 . . . . .	417
---------------------------------------------------------	-----

**graphique, export**

Export de graphiques . . . . .	331
Statistique univariée . . . . .	339

**grappe, échantillonnage**

Définir un plan d'échantillonnage complexe . . . . .	501
------------------------------------------------------	-----

**H****hazard ratio**

Trajectoires de soins: un exemple de données longitudinales . . . . .	705
-----------------------------------------------------------------------	-----

**hazard ratio (HR)**

Analyse de survie . . . . .	651
-----------------------------	-----

<b>HDI</b>	
Statistique bivariée . . . . .	365
<b>histogramme</b>	
Comparaisons (moyennes et proportions) . . . . .	485
Statistique univariée . . . . .	339
<b>historique des commandes</b>	
Premier contact . . . . .	13
<b>homogénéité des classes</b>	
Analyse de séquences . . . . .	679
<b>image bitmap</b>	
Export de graphiques . . . . .	331
<b>image matricielle</b>	
Export de graphiques . . . . .	331
<b>image vectorielle</b>	
Export de graphiques . . . . .	331
<b>indépendance</b>	
Statistique bivariée . . . . .	365
<b>indertie, perte relative</b>	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>index plots</b>	
Analyse de séquences . . . . .	679
<b>indexation</b>	
Vecteurs, indexation et assignation . . . . .	67
<b>indexation bidimensionnelle</b>	
Listes et Tableaux de données . . . . .	87
<b>indexation directe</b>	
Vecteurs, indexation et assignation . . . . .	67
<b>indexation par condition</b>	
Listes et Tableaux de données . . . . .	87
Vecteurs, indexation et assignation . . . . .	67
<b>indexation par nom</b>	
Listes et Tableaux de données . . . . .	87
<b>indexation par position</b>	
Vecteurs, indexation et assignation . . . . .	67
<b>indexation, assignation</b>	
Listes et Tableaux de données . . . . .	87
<b>indice de similarité</b>	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>indice de similarité de Gower</b>	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>inertie</b>	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>inertie, saut</b>	
Analyse de séquences . . . . .	679
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>installation</b>	
Installation de R et RStudio . . . . .	11
<b>integer</b>	
Premier travail avec des données . . . . .	33
Vecteurs, indexation et assignation . . . . .	67
<b>interaction</b>	
Effets d'interaction dans un modèle . . . . .	617
<b>intercept</b>	
Modèles à effets aléatoires (modèles mixtes et GEE)	
<b>interface</b>	
Premier contact . . . . .	13
<b>interquartilen écart</b>	
Statistique univariée . . . . .	339
<b>intervalle de confiance</b>	
Comparaisons (moyennes et proportions) . . . . .	485
Intervalles de confiance . . . . .	477
<b>intervalle de confiance d'un odds ratio</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>intervalle de confiance d'une moyenne</b>	
Intervalles de confiance . . . . .	477
<b>intervalle de confiance d'une proportion</b>	
Données pondérées . . . . .	465
Intervalles de confiance . . . . .	477
<b>invite de commande</b>	
Premier contact . . . . .	13

**K****Kaplan-Meier**

Analyse de survie ..... 651

**L****labelled data**

Facteurs et vecteurs labellisés ..... 117

**labellisé, vecteur**

Facteurs et vecteurs labellisés ..... 117

Import de données ..... 145

**labellisée, donnée**

Facteurs et vecteurs labellisés ..... 117

**labellisée, variable**

Import de données ..... 145

**latente, modèle mixte à classes latentes**

Trajectoires de soins: un exemple de données longitudinales ..... 705

**LCS**

Analyse de séquences ..... 679

**level, factor**

Premier travail avec des données ..... 33

**Lexis, diagramme**

Diagramme de Lexis ..... 1111

**libre, logiciel**

Présentation et Philosophie ..... 7

**life course analysis**

Analyse de séquences ..... 679

**linéaire, régression**

Statistique bivariée ..... 365

**liste**

Listes et Tableaux de données ..... 87

**logical**

Vecteurs, indexation et assignation ..... 67

**logiciel libre**

Présentation et Philosophie ..... 7

**logique, opérateur**

Conditions et comparaisons ..... 925

Vecteurs, indexation et assignation ..... 67

**logique, valeur**

Vecteurs, indexation et assignation ..... 67

**logistique, régression**

Données pondérées ..... 465

Régression logistique binaire, multinomiale et ordinale ..... 507

**logrank, test (comparaison de courbes de survie)**

Analyse de survie ..... 651

**loi normale**

Comparaisons (moyennes et proportions) ..... 485

**Longest Common Subsequence**

Analyse de séquences ..... 679

**M****Mann-Whitney, test**

Comparaisons (moyennes et proportions) ..... 485

**manquante, valeur**

Analyse des correspondances multiples (ACM) ..... 557

Classification ascendante hiérarchique (CAH) ..... 595

Import de données ..... 145

Régression logistique binaire, multinomiale et ordinale ..... 507

Statistique bivariée ..... 365

Vecteurs, indexation et assignation ..... 67

**Markdown**

R Markdown : les rapports automatisés ..... 979

**matching, optimal**

Analyse de séquences ..... 679

**matrice de confusion**

Régression logistique binaire, multinomiale et ordinale ..... 507

**matrice de corrélation**

Graphiques univariés et bivariés avec ggplot2 ..... 417

**matrice de distances**

Classification ascendante hiérarchique (CAH) ..... 595

**maximum**

Statistique univariée ..... 339

<b>médiane</b>	
Comparaisons (moyennes et proportions) . . . . .	485
Statistique univariée . . . . .	339
<b>médiane, test de comparaison</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>métadonnée</b>	
Import de données . . . . .	145
<b>méthode de Ward</b>	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>minimum</b>	
Statistique univariée . . . . .	339
<b>mise à jour, R</b>	
Installation de R et RStudio . . . . .	11
<b>mixte, modèle</b>	
Modèles à effets aléatoires (modèles mixtes et GEE)	
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>mixte, modèle mixte à classes latentes</b>	
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>mixtes, analyse factorielle avec données</b>	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>modalité</b>	
Statistique univariée . . . . .	339
<b>modalité de référence</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>modalité, facteur</b>	
Premier travail avec des données . . . . .	33
<b>modèle à effets aléatoires</b>	
Modèles à effets aléatoires (modèles mixtes et GEE)	
<b>modèle de Cox</b>	
Analyse de survie . . . . .	651
<b>modèle GEE</b>	
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>modèle linéaire généralisé</b>	
Données pondérées . . . . .	465
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>modèle mixte</b>	
Modèles à effets aléatoires (modèles mixtes et GEE)	
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>modèle, coefficients</b>	
Modèles à effets aléatoires (modèles mixtes et GEE)	
<b>modèle, résidus</b>	
Modèles à effets aléatoires (modèles mixtes et GEE)	
<b>mosaïque, graphique</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Statistique bivariée . . . . .	365
<b>moustaches, boîte</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Statistique bivariée . . . . .	365
Statistique univariée . . . . .	339
<b>moyenne</b>	
Données pondérées . . . . .	465
Statistique bivariée . . . . .	365
Statistique univariée . . . . .	339
<b>moyenne, âge</b>	
Calculer un âge . . . . .	1103
<b>moyenne, comparaison</b>	
Comparaisons (moyennes et proportions) . . . . .	485
Données pondérées . . . . .	465
<b>moyenne, intervalle de confiance</b>	
Intervalles de confiance . . . . .	477
<b>multi-états, modèle de survie</b>	
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>multicolinéarité</b>	
Multicolinéarité dans la régression . . . . .	645
<b>multicolinéarité parfaite</b>	
Multicolinéarité dans la régression . . . . .	645
<b>multidimensional scaling</b>	
Analyse de séquences . . . . .	679
<b>multinomiale, régression logistique</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Trajectoires de soins: un exemple de données longitudinales . . . . .	705

**N****NA**

Premier contact ..... 13

**nom, indexation**Listes et Tableaux de données ..... 87  
Vecteurs, indexation et assignation ..... 67**nombre entier**

Vecteurs, indexation et assignation ..... 67

**nombre réel**

Vecteurs, indexation et assignation ..... 67

**normale, loi**

Comparaisons (moyennes et proportions) ..... 485

**normalité, test de Shapiro-Wilk**

Comparaisons (moyennes et proportions) ..... 485

**notation formule**

Formules ..... 929

**notation scientifique**

Comparaisons (moyennes et proportions) ..... 485

**noyau, estimation**

Statistique univariée ..... 339

**nuage de points**

Graphiques univariés et bivariés avec ggplot2 ..... 417

**numeric**Premier travail avec des données ..... 33  
Vecteurs, indexation et assignation ..... 67**numérique, variable**Premier travail avec des données ..... 33  
Statistique univariée ..... 339**O****observation**

Premier travail avec des données ..... 33

**observations répétées, modèle**

Trajectoires de soins: un exemple de données longitudinales ..... 705

**odds ratio**Comparaisons (moyennes et proportions) ..... 485  
Régression logistique binaire, multinomiale et ordinale ..... 507**odds ratio, intervalle de confiance**

Régression logistique binaire, multinomiale et ordinale ..... 507

**opérateur de comparaison**

Conditions et comparaisons ..... 925

**opérateur logique**Conditions et comparaisons ..... 925  
Vecteurs, indexation et assignation ..... 67**optimal matching**

Analyse de séquences ..... 679

**optimal, appariement**

Analyse de séquences ..... 679

**ordinaire, régression logistique**

Régression logistique binaire, multinomiale et ordinale ..... 507

**ordinale, régression logistique**Régression logistique binaire, multinomiale et ordinale ..... 507  
Trajectoires de soins: un exemple de données longitudinales ..... 705**ordonner le tri à plat**

Statistique univariée ..... 339

**P****package**

Extensions (installation, mise à jour) ..... 55

**palette de couleurs**

Analyse des correspondances multiples (ACM) .... 557

**PAM**

Trajectoires de soins: un exemple de données longitudinales ..... 705

**partition**Analyse de séquences ..... 679  
Classification ascendante hiérarchique (CAH) .... 595

<b>Partitioning Around Medoids</b>	
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>pas à pas, sélection descendante</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>patron</b>	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>perte relative d'inertie</b>	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>Phi<sup>2</sup>, distance</b>	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>pirate, graphique</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Statistique bivariée . . . . .	365
<b>pirateplot</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Statistique bivariée . . . . .	365
<b>plan d'échantillonnage</b>	
Données pondérées . . . . .	465
<b>plan factoriel</b>	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>poids de réPLICATION</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>points, nuage de</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>pondération</b>	
Données pondérées . . . . .	465
<b>pooled variance</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>position, indexation</b>	
Listes et Tableaux de données . . . . .	87
Vecteurs, indexation et assignation . . . . .	67
<b>projets</b>	
Organiser ses fichiers . . . . .	135
<b>prompt</b>	
Premier contact . . . . .	13
<b>proportion, intervalle de confiance</b>	
Données pondérées . . . . .	465
Intervalles de confiance . . . . .	477
<b>proportion, test de comparaison</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>proxy</b>	
Installation de R et RStudio . . . . .	11
<b>Q</b>	
<b>qualitative, variable</b>	
Analyse des correspondances multiples (ACM) . . . . .	557
Classification ascendante hiérarchique (CAH) . . . . .	595
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Statistique bivariée . . . . .	365
Statistique univariée . . . . .	339
<b>quantile</b>	
Données pondérées . . . . .	465
Statistique univariée . . . . .	339
<b>quantitative, variable</b>	
Analyse des correspondances multiples (ACM) . . . . .	557
Classification ascendante hiérarchique (CAH) . . . . .	595
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Statistique bivariée . . . . .	365
Statistique univariée . . . . .	339
<b>quartile</b>	
Statistique univariée . . . . .	339
<b>R</b>	
<b>R Markdown</b>	
R Markdown : les rapports automatisés . . . . .	979
<b>rapport des cotes</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>raster</b>	
Analyse spatiale . . . . .	787
<b>recodage de variables</b>	
Recodage de variables . . . . .	197

<b>recyclage</b>	
Premier contact . . . . .	13
<b>recycling rule</b>	
Premier contact . . . . .	13
<b>réel, nombre</b>	
Vecteurs, indexation et assignation . . . . .	67
<b>référence, modalité</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>régression linéaire</b>	
Statistique bivariée . . . . .	365
<b>régression logistique</b>	
Données pondérées . . . . .	465
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>régression logistique binaire</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>régression logistique multinomiale</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>régression logistique ordinaire</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>régression logistique ordinale</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>régression, droite</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Statistique bivariée . . . . .	365
<b>régulière, expression</b>	
Expressions régulières . . . . .	977
Manipuler du texte avec stringr . . . . .	289
<b>relatif, risque</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>relation fonctionnelle</b>	
Formules . . . . .	929
lattice : graphiques et formules . . . . .	879
<b>répartition empirique, fonction</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Statistique univariée . . . . .	339
<b>répertoire de travail</b>	
Organiser ses fichiers . . . . .	135
<b>réplique, poids</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>résidus de Schoenfeld</b>	
Analyse de survie . . . . .	651
<b>résidus du modèle</b>	
Modèles à effets aléatoires (modèles mixtes et GEE)	
<b>résidus, test du Chi<sup>2</sup></b>	
Comparaisons (moyennes et proportions) . . . . .	485
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>résolution</b>	
Export de graphiques . . . . .	331
<b>ressemblance</b>	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>révolu, âge</b>	
Calculer un âge . . . . .	1103
<b>risque relatif</b>	
Comparaisons (moyennes et proportions) . . . . .	485
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>S</b>	
<b>Sankey, diagramme</b>	
Autres extensions graphiques . . . . .	897
<b>SAS, fichier</b>	
Import de données . . . . .	145
<b>saut d'inertie</b>	
Analyse de séquences . . . . .	679
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>Schoenfeld, résidus</b>	
Analyse de survie . . . . .	651
<b>scientifique, notation</b>	
Comparaisons (moyennes et proportions) . . . . .	485

<b>script</b>	
Organiser ses fichiers . . . . .	135
Premier travail avec des données . . . . .	33
<b>scripts</b>	
Présentation et Philosophie . . . . .	7
<b>secteur, diagramme</b>	
Statistique univariée . . . . .	339
<b>section</b>	
Premier travail avec des données . . . . .	33
<b>sélection descendante pas à pas</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>séparateur de champs</b>	
Import de données . . . . .	145
<b>séparateur décimal</b>	
Import de données . . . . .	145
<b>séquence, analyse</b>	
Analyse de séquences . . . . .	679
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>séquence, tapis</b>	
Analyse de séquences . . . . .	679
<b>Shapiro-Wilk, test de normalité</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>similarité, indice</b>	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>sous-échantillon</b>	
Définir un plan d'échantillonnage complexe . . . . .	501
Données pondérées . . . . .	465
<b>SPSS, fichier</b>	
Import de données . . . . .	145
<b>statistique bivariée</b>	
Statistique bivariée . . . . .	365
<b>statistique descriptive</b>	
Statistique bivariée . . . . .	365
Statistique univariée . . . . .	339
<b>statistique univariée</b>	
Statistique univariée . . . . .	339
<b>strate</b>	
Définir un plan d'échantillonnage complexe . . . . .	501
<b>stratifié, échantillonnage</b>	
Définir un plan d'échantillonnage complexe . . . . .	501
<b>structure d'un objet</b>	
Premier travail avec des données . . . . .	33
Visualiser ses données . . . . .	175
<b>Student, test-t</b>	
Données pondérées . . . . .	465
<b>Student, test t</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>survie, analyse</b>	
Analyse de survie . . . . .	651
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>survie, modèle multi-états</b>	
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
 <b>T</b>	
<b>tableau croisé</b>	
Données pondérées . . . . .	465
Statistique bivariée . . . . .	365
<b>tableau croisé, coefficient de contingence de Cramer</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>tableau croisé, graphique en mosaïque</b>	
Statistique bivariée . . . . .	365
<b>tableau croisé, test exact de Fisher</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>tableau de donnée</b>	
Premier travail avec des données . . . . .	33
<b>tableau de données, fusion</b>	
Fusion de tables . . . . .	269
<b>tableau de données, tri</b>	
Tris . . . . .	257
<b>tableau de fréquences</b>	
Statistique univariée . . . . .	339
<b>tagged missing value</b>	
Import de données . . . . .	145

<b>tagged NA</b>	
Import de données . . . . .	145
<b>tapis</b>	
Analyse de séquences . . . . .	679
<b>task view</b>	
Extensions (installation, mise à jour) . . . . .	55
<b>test d’égalité des variances</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>test de comparaison de deux proportions</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>test de normalité de Shapiro-Wilk</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>test de Wilcoxon/Mann-Whitney</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>test du Chi<sup>2</sup></b>	
Comparaisons (moyennes et proportions) . . . . .	485
Données pondérées . . . . .	465
<b>test du Chi<sup>2</sup>, résidus</b>	
Comparaisons (moyennes et proportions) . . . . .	485
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>test exact de Fisher</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>test t de Student</b>	
Comparaisons (moyennes et proportions) . . . . .	485
Données pondérées . . . . .	465
<b>texte</b>	
Vecteurs, indexation et assignation . . . . .	67
<b>texte tabulé, fichier</b>	
Import de données . . . . .	145
<b>texte, fichier</b>	
Import de données . . . . .	145
<b>tibble</b>	
Import de données . . . . .	145
<b>tidy data</b>	
Introduction au tidyverse . . . . .	59
Réorganiser ses données avec tidyr . . . . .	299
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>tidyverse</b>	
Extensions (installation, mise à jour) . . . . .	55
Import de données . . . . .	145
Introduction au tidyverse . . . . .	59
<b>total</b>	
Données pondérées . . . . .	465
<b>trajectoire biographique</b>	
Analyse de séquences . . . . .	679
<b>transversale, entropie</b>	
Analyse de séquences . . . . .	679
<b>tri à plat</b>	
Données pondérées . . . . .	465
Statistique univariée . . . . .	339
<b>tri à plat, ordonner</b>	
Statistique univariée . . . . .	339
<b>U</b>	
<b>univariée, statistique</b>	
Données pondérées . . . . .	465
Statistique univariée . . . . .	339
<b>V</b>	
<b>valeur booléenne</b>	
Vecteurs, indexation et assignation . . . . .	67
<b>valeur logique</b>	
Vecteurs, indexation et assignation . . . . .	67
<b>valeur manquante</b>	
Classification ascendante hiérarchique (CAH) . . . . .	595
Import de données . . . . .	145
Statistique bivariée . . . . .	365
Statistique univariée . . . . .	339
Vecteurs, indexation et assignation . . . . .	67
<b>valeur manquante déclarée</b>	
Import de données . . . . .	145
<b>valeur, étiquette</b>	
Facteurs et vecteurs labellisés . . . . .	117
Import de données . . . . .	145

<b>variable</b>	Premier travail avec des données . . . . .	33
<b>variable catégorielle</b>	Facteurs et vecteurs labellisés . . . . .	117
<b>variable d'intérêt</b>	Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>variable explicative</b>	Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>variable labellisée</b>	Import de données . . . . .	145
<b>variable numérique</b>	Premier travail avec des données . . . . .	33
<b>variable qualitative</b>	Analyse des correspondances multiples (ACM) . . . . .	557
	Classification ascendante hiérarchique (CAH) . . . . .	595
	Régression logistique binaire, multinomiale et ordinale . . . . .	507
	Statistique bivariée . . . . .	365
	Statistique univariée . . . . .	339
<b>variable quantitative</b>	Analyse des correspondances multiples (ACM) . . . . .	557
	Classification ascendante hiérarchique (CAH) . . . . .	595
	Régression logistique binaire, multinomiale et ordinale . . . . .	507
	Statistique bivariée . . . . .	365
	Statistique univariée . . . . .	339
<b>variable, étiquette</b>	Facteurs et vecteurs labellisés . . . . .	117
	Import de données . . . . .	145
<b>variable, recodage</b>	Recodage de variables . . . . .	197
<b>variance</b>	Comparaisons (moyennes et proportions) . . . . .	485
	Données pondérées . . . . .	465
<b>variance inflation factor</b>	Multicolinéarité dans la régression . . . . .	645
<b>variance, analyse de</b>	Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>variance, pooled</b>	Comparaisons (moyennes et proportions) . . . . .	485
<b>variance, test d'égalité</b>	Comparaisons (moyennes et proportions) . . . . .	485
<b>vecteur</b>	Premier contact . . . . .	13
	Vecteurs, indexation et assignation . . . . .	67
<b>vecteur labellisé</b>	Import de données . . . . .	145
<b>vector</b>	Premier contact . . . . .	13
<b>viewer</b>	Premier travail avec des données . . . . .	33
	Visualiser ses données . . . . .	175
<b>VIF</b>	Multicolinéarité dans la régression . . . . .	645
<b>violin plot</b>	Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>violon, graphique en</b>	Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>visionneusee</b>	Visualiser ses données . . . . .	175
<b>W</b>		
<b>Ward, méthode</b>	Classification ascendante hiérarchique (CAH) . . . . .	595
<b>Wilcoxon, test</b>	Comparaisons (moyennes et proportions) . . . . .	485



# Index des fonctions

## %

<code>%in%</code> (base)	Conditions et comparaisons . . . . .	925
•		
<code>.N</code> (data.table)	Manipulations avancées avec data.table . . . . .	251

## A

<code>A2Rplot</code> (JLutils)	Classification ascendante hiérarchique (CAH) . . . . .	595
	Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<code>aply</code> (plyr)	Vectorisation . . . . .	953
<code>abline</code> (graphics)	Statistique bivariée . . . . .	365
<code>add.NA</code> (base)	Régression logistique binaire, multinomiale et ordinale . . . . .	507
<code>add_global_p</code> (gtsummary)	Régression logistique binaire, multinomiale et ordinale . . . . .	507
<code>add_overall</code> (gtsummary)	Statistique bivariée . . . . .	365
	Trajectoires de soins: un exemple de données longitudinales . . . . .	705

## `add_p` (gtsummary)

Comparaisons (moyennes et proportions) . . . . .	485
Statistique bivariée . . . . .	365

## `addNA` (base)

Facteurs et vecteurs labellisés . . . . .	117
-------------------------------------------	-----

## `addNAstr` (questionr)

Facteurs et vecteurs labellisés . . . . .	117
Régression logistique binaire, multinomiale et ordinale . . . . .	507

## `adply` (plyr)

Vectorisation . . . . .	953
-------------------------	-----

## `aes` (ggplot2)

Exemples de graphiques avancés . . . . .	839
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
Trajectoires de soins: un exemple de données longitudinales . . . . .	705

## `after_stat` (ggplot2)

Exemples de graphiques avancés . . . . .	839
------------------------------------------	-----

## `aggregate` (stats)

Analyse de séquences . . . . .	679
Formules . . . . .	929

## `agnes` (cluster)

Classification ascendante hiérarchique (CAH) . . . . .	595
Classification ascendante hiérarchique (CAH) . . . . .	595

## `AIC` (stats)

Régression logistique binaire, multinomiale et ordinale . . . . .	507
-------------------------------------------------------------------	-----

## `AIC.svyglm` (survey)

Régression logistique binaire, multinomiale et ordinale . . . . .	507
-------------------------------------------------------------------	-----

<b>airlines</b> (nycflights13)	
Manipuler les données avec dplyr .....	233
<b>airports</b> (nycflights13)	
Manipuler les données avec dplyr .....	233
<b>allEffects</b> (effects)	
Effets d'interaction dans un modèle .....	617
Régression logistique binaire, multinomiale et ordinale .....	507
Régression logistique binaire, multinomiale et ordinale .....	507
<b>alply</b> (plyr)	
Vectorisation .....	953
<b>anova</b> (geepack)	
Modèles à effets aléatoires (modèles mixtes et GEE)	
<b>anova</b> (stats)	
Effets d'interaction dans un modèle .....	617
Régression logistique binaire, multinomiale et ordinale .....	507
<b>Anova</b> (car)	
Régression logistique binaire, multinomiale et ordinale .....	507
<b>anti_join</b> (dplyr)	
Fusion de tables .....	269
<b>aov</b> (stats)	
lattice : graphiques et formules .....	879
<b>append</b> (base)	
Listes et Tableaux de données .....	87
<b>apply</b> (base)	
Vectorisation .....	953
<b>arrange</b> (dplyr)	
Manipuler les données avec dplyr .....	233
Tris .....	257
<b>array</b> (base)	
Vectorisation .....	953
<b>as.character</b> (base)	
Recodage de variables .....	197
<b>as.clustrange</b> (WeightedCluster)	
Classification ascendante hiérarchique (CAH) .....	595
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>as.data.frame</b> (base)	
Graphiques univariés et bivariés avec ggplot2 .....	417
<b>as.data.frame</b> (tibble)	
Introduction au tidyverse .....	59
<b>as.data.table</b> (data.table)	
Manipulations avancées avec data.table .....	251
<b>as.hclust</b> (cluster)	
Classification ascendante hiérarchique (CAH) .....	595
<b>as.integer</b> (base)	
Définir un plan d'échantillonnage complexe .....	501
<b>as.matrix</b> (base)	
Statistique univariée .....	339
<b>as.numeric</b> (base)	
Recodage de variables .....	197
<b>as.seqtree</b> (WeightedCluster)	
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>as_factor</b> (haven)	
Recodage de variables .....	197
<b>as_tibble</b> (tibble)	
Introduction au tidyverse .....	59
Manipuler les données avec dplyr .....	233
<b>B</b>	
<b>barchart</b> (lattice)	
lattice : graphiques et formules .....	879
<b>barplot</b> (graphics)	
Analyse des correspondances multiples (ACM) ...	557
<b>base</b> (base)	
Vectorisation .....	953
<b>beep</b> (beepr)	
Structures conditionnelles .....	945
<b>best.cutree</b> (JLutils)	
Classification ascendante hiérarchique (CAH) .....	595
<b>bind_cols</b> (dplyr)	
Fusion de tables .....	269

<b>bind_rows</b> (dplyr)	
Fusion de tables . . . . .	269
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>biplot</b> (ade4)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>boxplot</b> (ade4)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>boxplot</b> (graphics)	
lattice : graphiques et formules . . . . .	879
Statistique bivariée . . . . .	365
Statistique univariée . . . . .	339
<b>break</b> (base)	
Structures conditionnelles . . . . .	945
<b>brewer.pal</b> (RColorBrewer)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>bwplot</b> (lattice)	
lattice : graphiques et formules . . . . .	879
<b>by</b> (base)	
Sous-ensembles . . . . .	261
<b>C</b>	
<b>c</b> (base)	
Manipulations avancées avec data.table . . . . .	251
Premier contact . . . . .	13
Vecteurs, indexation et assignation . . . . .	67
<b>CA</b> (FactoMineR)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>case_when</b> (dplyr)	
Manipuler les données avec dplyr . . . . .	233
Recodage de variables . . . . .	197
<b>cbind</b> (base)	
Fusion de tables . . . . .	269
<b>chisq.residuals</b> (questionr)	
Comparaisons (moyennes et proportions) . . . . .	485
<b>chisq.test</b> (stats)	
Comparaisons (moyennes et proportions) . . . . .	485
Données pondérées . . . . .	465
<b>class</b> (base)	
Facteurs et vecteurs labellisés . . . . .	117
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
Vecteurs, indexation et assignation . . . . .	67
<b>clm</b> (ordinal)	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>clmm</b> (ordinal)	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>cm.colors</b> (grDevices)	
Couleurs et Palettes . . . . .	1021
<b>cmdscale</b> (stats)	
Analyse de séquences . . . . .	679
<b>coef</b> (stats)	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>coef</b> (survey)	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>collect</b> (dplyr)	
Import de données . . . . .	145
<b>colorblind_pal</b> (ggthemes)	
Couleurs et Palettes . . . . .	1021
<b>colors</b> (grDevices)	
Statistique univariée . . . . .	339
<b>column_to_rownames</b> (tibble)	
Introduction au tidyverse . . . . .	59
<b>comma</b> (scales)	
Mettre en forme des nombres . . . . .	1019
<b>complete</b> (tidyverse)	
Réorganiser ses données avec tidyverse . . . . .	299
<b>complete.cases</b> (stats)	
Analyse des correspondances multiples (ACM) . . . . .	557
Statistique bivariée . . . . .	365
<b>confint</b> (stats)	
Régression logistique binaire, multinomiale et ordinale . . . . .	507

<b>confint</b> (survey)	
Intervalles de confiance . . . . .	477
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>confint</b> (svrmpmisc)	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>contains</b> (dplyr)	
Manipuler les données avec dplyr . . . . .	233
Visualiser ses données . . . . .	175
<b>contour</b> (graphics)	
Statistique bivariée . . . . .	365
<b>coord_capped_cart</b> (lemon)	
Étendre ggplot2 . . . . .	793
<b>coord_capped_flip</b> (lemon)	
Étendre ggplot2 . . . . .	793
<b>coord_flip</b> (ggplot2)	
Exemples de graphiques avancés . . . . .	839
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>copy_labels_from</b> (labelled)	
Analyse de survie . . . . .	651
<b>cor</b> (stats)	
Statistique bivariée . . . . .	365
<b>corresp</b> (MASS)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>count</b> (dplyr)	
Formules . . . . .	929
Manipuler les données avec dplyr . . . . .	233
<b>cox.zph</b> (survival)	
Analyse de survie . . . . .	651
<b>cprop</b> (questionr)	
Données pondérées . . . . .	465
Statistique bivariée . . . . .	365
<b>cramer.v</b> (questionr)	
Comparaisons (moyennes et proportions) . . . . .	485
<b>create_report</b> (DataExplorer)	
Visualiser ses données . . . . .	175
<b>cut</b> (base)	
Recodage de variables . . . . .	197
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>cutree</b> (stats)	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>D</b>	
<b>daisy</b> (cluster)	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>daply</b> (plyr)	
Vectorisation . . . . .	953
<b>data</b> (utils)	
Premier travail avec des données . . . . .	33
<b>data.frame</b> (base)	
Listes et Tableaux de données . . . . .	87
Manipuler les données avec dplyr . . . . .	233
<b>datatable</b> (DT)	
R Markdown : les rapports automatisés . . . . .	979
<b>dbConnect</b> (DBI)	
Import de données . . . . .	145
<b>dbDisconnect</b> (DBI)	
Import de données . . . . .	145
<b>dbGetQuery</b> (DBI)	
Import de données . . . . .	145
<b>dbListFields</b> (DBI)	
Import de données . . . . .	145
<b>dbListTables</b> (DBI)	
Import de données . . . . .	145
<b>dbReadTable</b> (DBI)	
Import de données . . . . .	145
<b>dcast</b> (data.table)	
Manipulations avancées avec data.table . . . . .	251
Réorganiser ses données avec tidyr . . . . .	299
<b>ddply</b> (plyr)	
Vectorisation . . . . .	953
<b>density</b> (stats)	
Statistique univariée . . . . .	339
<b>densityplot</b> (lattice)	
lattice : graphiques et formules . . . . .	879

<b>describe</b> (questionr)	
Facteurs et vecteurs labellisés . . . . .	117
Import de données . . . . .	145
Listes et Tableaux de données . . . . .	87
Premier travail avec des données . . . . .	33
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
Visualiser ses données . . . . .	175
<b>dev.off</b> (grDevices)	
Export de graphiques . . . . .	331
<b>dev.print</b> (grDevices)	
Export de graphiques . . . . .	331
<b>dim</b> (base)	
Formules . . . . .	929
Listes et Tableaux de données . . . . .	87
Premier travail avec des données . . . . .	33
<b>dimdesc</b> (FactoMineR)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>display.brewer.all</b> (RColorBrewer)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>disscenter</b> (TraMineR)	
Analyse de séquences . . . . .	679
<b>dist</b> (stats)	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>dist.dudi</b> (ade4)	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>distinct</b> (dplyr)	
Manipuler les données avec dplyr . . . . .	233
<b>diverging_bar_chart</b> (ggcharts)	
Étendre ggplot2 . . . . .	793
<b>diverging_lollipop_chart</b> (ggcharts)	
Étendre ggplot2 . . . . .	793
<b>dply</b> (plyr)	
Vectorisation . . . . .	953
<b>do.call</b> (base)	
Vectorisation . . . . .	953
<b>dotchart</b> (graphics)	
Statistique univariée . . . . .	339
<b>dotplot</b> (lattice)	
lattice : graphiques et formules . . . . .	879
<b>dput</b> (base)	
Recodage de variables . . . . .	197
<b>drop1</b> (stats)	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>dudi.acm</b> (ade4)	
Analyse des correspondances multiples (ACM) . . . . .	557
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>dudi.coa</b> (ade4)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>dudi.mix</b> (ade4)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>dudi.pca</b> (ade4)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>Duration</b> (lubridate)	
Calculer un âge . . . . .	1103
<b>E</b>	
<b>effect</b> (effects)	
Effets d'interaction dans un modèle . . . . .	617
<b>else</b> (base)	
Structures conditionnelles . . . . .	945
<b>ends_width</b> (dplyr)	
Manipuler les données avec dplyr . . . . .	233
<b>example</b> (utils)	
Où trouver de l'aide? . . . . .	163
<b>exp</b> (base)	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>expression</b> (base)	
Annotations mathématiques . . . . .	1097
<b>extract</b> (tidyverse)	
Manipuler du texte avec stringr . . . . .	289
Réorganiser ses données avec tidyverse . . . . .	299

# F

<b>facet_grid (ggplot2)</b>	
Analyse de séquences . . . . .	679
Exemples de graphiques avancés . . . . .	839
Formules . . . . .	929
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
lattice : graphiques et formules . . . . .	879
<b>facet_rep_grid (lemon)</b>	
Étendre ggplot2 . . . . .	793
<b>facet_rep_wrap (lemon)</b>	
Étendre ggplot2 . . . . .	793
<b>facet_wrap (ggplot2)</b>	
Formules . . . . .	929
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
lattice : graphiques et formules . . . . .	879
<b>factor (base)</b>	
Facteurs et vecteurs labellisés . . . . .	117
Recodage de variables . . . . .	197
<b>FAMD (FactoMineR)</b>	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>fct_collapse (forcats)</b>	
Recodage de variables . . . . .	197
<b>fct_explicit_na (forcats)</b>	
Exemples de graphiques avancés . . . . .	839
Recodage de variables . . . . .	197
<b>fct_inorder (forcats)</b>	
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>fct_lump (forcats)</b>	
Recodage de variables . . . . .	197
<b>fct_other (forcats)</b>	
Recodage de variables . . . . .	197
<b>fct_recode (forcats)</b>	
Exemples de graphiques avancés . . . . .	839
Recodage de variables . . . . .	197
<b>fct_relevel (forcats)</b>	
Exemples de graphiques avancés . . . . .	839
<b>fct_reorder (forcats)</b>	
Exemples de graphiques avancés . . . . .	839
<b>few_pal (ggthemes)</b>	
Couleurs et Palettes . . . . .	1021
<b>filled.contour (graphics)</b>	
Statistique bivariée . . . . .	365
<b>filter (dplyr)</b>	
Manipuler du texte avec stringr . . . . .	289
Manipuler les données avec dplyr . . . . .	233
Sous-ensembles . . . . .	261
<b>finalfit (finalfit)</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>first (data.table)</b>	
Manipulations avancées avec data.table . . . . .	251
<b>fisher.test (stats)</b>	
Comparaisons (moyennes et proportions) . . . . .	485
<b>fixed (stringr)</b>	
Manipuler du texte avec stringr . . . . .	289
<b>flights (nycflights13)</b>	
Manipuler les données avec dplyr . . . . .	233
<b>for (base)</b>	
Structures conditionnelles . . . . .	945
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>forest_model (forestmodel)</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>format (base)</b>	
Premier contact . . . . .	13
<b>formula (stats)</b>	
Formules . . . . .	929
<b>freq (questionr)</b>	
Données pondérées . . . . .	465
Import de données . . . . .	145
R Markdown : les rapports automatisés . . . . .	979
Recodage de variables . . . . .	197
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Statistique univariée . . . . .	339

<b>ft_pal</b> (hrbrthemes)	
Couleurs et Palettes . . . . .	1021
<b>full_join</b> (dplyr)	
Fusion de tables . . . . .	269
<b>G</b>	
<b>gather</b> (tidyverse)	
Analyse de séquences . . . . .	679
Réorganiser ses données avec tidyverse . . . . .	299
<b>geom_alluvium</b> (ggalluvial)	
Autres extensions graphiques . . . . .	897
<b>geom_area</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>geom_bar</b> (ggplot2)	
Exemples de graphiques avancés . . . . .	839
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>geom_bin2d</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>geom_boxplot</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
lattice : graphiques et formules . . . . .	879
<b>geom_braquet</b> (ggpubr)	
Étendre ggplot2 . . . . .	793
<b>geom_count</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>geom_density</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>geom_density_2d</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>geom_density_ridges_gradient</b> (ggridges)	
Étendre ggplot2 . . . . .	793
<b>geom_dumbbell</b> (ggalt)	
Étendre ggplot2 . . . . .	793
<b>geom_hex</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>geom_histogram</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>geom_label_repel</b> (ggrepel)	
Étendre ggplot2 . . . . .	793
<b>geom_line</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>geom_lollipop</b> (ggalt)	
Étendre ggplot2 . . . . .	793
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>geom_mosaic</b> (ggmosaic)	
Étendre ggplot2 . . . . .	793
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>geom_pirate</b> (ggpirate)	
Étendre ggplot2 . . . . .	793
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>geom_point</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
<b>geom_raster</b> (ggplot2)	
Analyse de séquences . . . . .	679
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>geom_rug</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>geom_smooth</b> (ggplot2)	
Données pondérées . . . . .	465
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
<b>geom_stratum</b> (ggalluvial)	
Autres extensions graphiques . . . . .	897
<b>geom_text</b> (ggplot2)	
Exemples de graphiques avancés . . . . .	839
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>geom_text_repel</b> (ggrepel)	
Étendre ggplot2 . . . . .	793
<b>geom_violin</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>geom_vline</b> (ggplot2)	
Introduction à ggplot2, la grammaire des graphiques . . . . .	395

<b>geom_waffle (waffle)</b>	
Étendre ggplot2 . . . . .	793
<b>get_legend (cowplot)</b>	
Combiner plusieurs graphiques . . . . .	825
<b>getValues (raster)</b>	
Import de données . . . . .	145
<b>getwd (base)</b>	
Organiser ses fichiers . . . . .	135
<b>ggbivariate (GGally)</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>ggcoef (GGally)</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>ggcoef_compare (GGally)</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>ggcoef_model (GGally)</b>	
Analyse de survie . . . . .	651
Effets d'interaction dans un modèle . . . . .	617
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>ggcoef_multinom (GGally)</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>ggcorr (GGally)</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>ggcoxzph (survminer)</b>	
Analyse de survie . . . . .	651
<b>ggdendrogram (ggdendro)</b>	
Autres extensions graphiques . . . . .	897
<b>ggduo (GGally)</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>ggeffect (ggeffect)</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>ggeffect (ggeffects)</b>	
Effets d'interaction dans un modèle . . . . .	617
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>ggforest (survminer)</b>	
Analyse de survie . . . . .	651
<b>gghighlight (gghighlight)</b>	
Étendre ggplot2 . . . . .	793
<b>ggpairs (GGally)</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>ggplot (ggplot2)</b>	
Données pondérées . . . . .	465
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
<b>ggplotly (plotly)</b>	
Graphiques interactifs . . . . .	875
<b>ggsave (ggplot2)</b>	
Export de graphiques . . . . .	331
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
<b>ggsurv (GGally)</b>	
Analyse de survie . . . . .	651
<b>ggsurvey (questionr)</b>	
Données pondérées . . . . .	465
<b>ggsurvplot (survminer)</b>	
Analyse de survie . . . . .	651
<b>ggtable (GGally)</b>	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>ggtitle (ggplot2)</b>	
Exemples de graphiques avancés . . . . .	839
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>glance (dplyr)</b>	
Facteurs et vecteurs labellisés . . . . .	117
<b>glance (tibble)</b>	
Listes et Tableaux de données . . . . .	87
Visualiser ses données . . . . .	175
<b>glm (stats)</b>	
Données pondérées . . . . .	465
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>glmulti (ggmulti)</b>	
Effets d'interaction dans un modèle . . . . .	617

<b>grid_arrange_shared_legend</b> (lemeon)	
Combiner plusieurs graphiques . . . . .	825
<b>group_by</b> (dplyr)	
Manipuler les données avec dplyr . . . . .	233
<b>grViz</b> (DiagrammeR)	
Autres extensions graphiques . . . . .	897
<b>guides</b> (ggplot2)	
Exemples de graphiques avancés . . . . .	839
<b>H</b>	
<b>haven_labelled</b> (haven)	
Import de données . . . . .	145
<b>haven_labelled</b> (labelled)	
Facteurs et vecteurs labellisés . . . . .	117
<b>hazard.msm</b> (msm)	
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>hclust</b> (fastcluster)	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>hclust</b> (stats)	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>HCPC</b> (FactoMineR)	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>hdv2003</b> (questionr)	
Formules . . . . .	929
Premier travail avec des données . . . . .	33
<b>head</b> (utils)	
Conditions et comparaisons . . . . .	925
Listes et Tableaux de données . . . . .	87
Premier travail avec des données . . . . .	33
<b>heat.colors</b> (grDevices)	
Couleurs et Palettes . . . . .	1021
<b>help</b> (utils)	
Où trouver de l'aide ? . . . . .	163
Premier contact . . . . .	13
<b>help.search</b> (utils)	
Où trouver de l'aide ? . . . . .	163

<b>help.start</b> (utils)	
Où trouver de l'aide ? . . . . .	163
<b>hist</b> (graphics)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Statistique univariée . . . . .	339
<b>histogram</b> (lattice)	
lattice : graphiques et formules . . . . .	879
<b>I</b>	
<b>icut</b> (questionr)	
Recodage de variables . . . . .	197
<b>if</b> (base)	
Structures conditionnelles . . . . .	945
<b>if_else</b> (dplyr)	
Manipuler les données avec dplyr . . . . .	233
Recodage de variables . . . . .	197
<b>ifelse</b> (base)	
Recodage de variables . . . . .	197
<b>image</b> (graphics)	
Statistique bivariée . . . . .	365
<b>inertia.dudi</b> (ade4)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>inner_join</b> (dplyr)	
Fusion de tables . . . . .	269
<b>install.packages</b> (utils)	
Extensions (installation, mise à jour) . . . . .	55
<b>install_github</b> (devtools)	
Analyse de séquences . . . . .	679
Analyse des correspondances multiples (ACM) . . . . .	557
Classification ascendante hiérarchique (CAH) . . . . .	595
Extensions (installation, mise à jour) . . . . .	55
Intervalles de confiance . . . . .	477
<b>interaction</b> (base)	
Recodage de variables . . . . .	197
<b>Interval</b> (lubridate)	
Calculer un âge . . . . .	1103
<b>invisible</b> (base)	
Vectorisation . . . . .	953

<b>iorder</b> (questionr)		<b>lapply</b> (base)	
Facteurs et vecteurs labellisés . . . . .	117	Vectorisation . . . . .	953
<b>ipsum_pal</b> (hrbrthemes)		Vectorisation . . . . .	953
Couleurs et Palettes . . . . .	1021		
<b>irec</b> (questionr)		<b>last</b> (data.table)	
Recodage de variables . . . . .	197	Manipulations avancées avec data.table . . . . .	251
<b>is.na</b> (base)			
Recodage de variables . . . . .	197	<b>lcmm</b> (lcmm)	
Vecteurs, indexation et assignation . . . . .	67	Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>is.numeric</b> (base)		<b>lead</b> (dplyr)	
Facteurs et vecteurs labellisés . . . . .	117	Manipuler les données avec dplyr . . . . .	233
<b>J</b>		<b>left_join</b> (dplyr)	
<b>jpeg</b> (grDevices)		Fusion de tables . . . . .	269
Export de graphiques . . . . .	331	<b>length</b> (base)	
<b>K</b>		Listes et Tableaux de données . . . . .	87
<b>kable</b> (knitr)		Premier contact . . . . .	13
R Markdown : les rapports automatisés . . . . .	979	Vecteurs, indexation et assignation . . . . .	67
Régression logistique binaire, multinomiale et ordinale . . . . .	507	<b>letters</b> (base)	
<b>kde2d</b> (MASS)		Vecteurs, indexation et assignation . . . . .	67
Graphiques univariés et bivariés avec ggplot2 . . . . .	417	<b>LETTERS</b> (base)	
Statistique bivariée . . . . .	365	Vecteurs, indexation et assignation . . . . .	67
<b>L</b>		<b>levels</b> (base)	
<b>label_wrap</b> (scales)		Facteurs et vecteurs labellisés . . . . .	117
Exemples de graphiques avancés . . . . .	839	Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>labelled</b> (labelled)		Tris . . . . .	257
Facteurs et vecteurs labellisés . . . . .	117	<b>library</b> (base)	
<b>labs</b> (ggplot2)		Extensions (installation, mise à jour) . . . . .	55
Exemples de graphiques avancés . . . . .	839	Organiser ses fichiers . . . . .	135
Graphiques univariés et bivariés avec ggplot2 . . . . .	417	Premier travail avec des données . . . . .	33
<b>lag</b> (dplyr)		<b>list</b> (base)	
Manipuler les données avec dplyr . . . . .	233	Listes et Tableaux de données . . . . .	87
		Manipulations avancées avec data.table . . . . .	251
<b>llply</b> (plyr)		<b>lm</b> (stats)	
		Données pondérées . . . . .	465
		lattice : graphiques et formules . . . . .	879
		Statistique bivariée . . . . .	365
<b>load</b> (base)			
		Export de données . . . . .	327
		Import de données . . . . .	145

<b>log</b> (base)	
Vectorisation . . . . .	953
<b>look_for</b> (labelled)	
Facteurs et vecteurs labellisés . . . . .	117
Listes et Tableaux de données . . . . .	87
Visualiser ses données . . . . .	175
<b>lookfor</b> (questionr)	
Visualiser ses données . . . . .	175
<b>lprop</b> (questionr)	
Données pondérées . . . . .	465
Statistique bivariée . . . . .	365
<b>ltabs</b> (questionr)	
Statistique bivariée . . . . .	365
<b>M</b>	
<b>makeCodebook</b> (dataMaid)	
Visualiser ses données . . . . .	175
<b>maply</b> (plyr)	
Vectorisation . . . . .	953
<b>mapply</b> (base)	
Vectorisation . . . . .	953
<b>matches</b> (dplyr)	
Manipuler les données avec dplyr . . . . .	233
<b>max</b> (base)	
Manipulations avancées avec data.table . . . . .	251
Premier contact . . . . .	13
Premier travail avec des données . . . . .	33
Statistique univariée . . . . .	339
<b>mca</b> (MASS)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>MCA</b> (FactoMineR)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>mcmcsamp</b> (lme4)	
Modèles à effets aléatoires (modèles mixtes et GEE)	
<b>mdply</b> (plyr)	
Vectorisation . . . . .	953
<b>mean</b> (base)	
Données pondérées . . . . .	465
Manipulations avancées avec data.table . . . . .	251
Où trouver de l'aide ? . . . . .	163
Premier contact . . . . .	13
Premier travail avec des données . . . . .	33
Sous-ensembles . . . . .	261
Statistique univariée . . . . .	339
<b>median</b> (stats)	
Formules . . . . .	929
Manipulations avancées avec data.table . . . . .	251
Premier travail avec des données . . . . .	33
Statistique univariée . . . . .	339
<b>melt</b> (data.table)	
Manipulations avancées avec data.table . . . . .	251
Réorganiser ses données avec tidyr . . . . .	299
<b>merge</b> (base)	
Fusion de tables . . . . .	269
Import de données . . . . .	145
<b>merge</b> (data.table)	
Fusion de tables . . . . .	269
<b>mermaid</b> (DiagrammeR)	
Autres extensions graphiques . . . . .	897
<b>min</b> (base)	
Manipulations avancées avec data.table . . . . .	251
Premier contact . . . . .	13
Premier travail avec des données . . . . .	33
Statistique univariée . . . . .	339
<b>mlply</b> (plyr)	
Vectorisation . . . . .	953
<b>month.abb</b> (base)	
Vecteurs, indexation et assignation . . . . .	67
<b>month.name</b> (base)	
Vecteurs, indexation et assignation . . . . .	67
<b>mosaic</b> (vcd)	
Statistique bivariée . . . . .	365
<b>mosaicplot</b> (graphics)	
Statistique bivariée . . . . .	365
<b>msm</b> (msm)	
Trajectoires de soins: un exemple de données longitudinales . . . . .	705

<b>multinom</b> (nnet)	Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>multiplot</b> (JLutils)	Combiner plusieurs graphiques . . . . .	825
<b>mutate</b> (dplyr)	Import de données . . . . .	145
	Manipuler les données avec dplyr . . . . .	233
<b>N</b>		
<b>NA</b> (base)	Vecteurs, indexation et assignation . . . . .	67
<b>na_range</b> (labelled)	Import de données . . . . .	145
<b>na_values</b> (labelled)	Import de données . . . . .	145
<b>names</b> (base)	Listes et Tableaux de données . . . . .	87
	Premier travail avec des données . . . . .	33
	Recodage de variables . . . . .	197
	Vecteurs, indexation et assignation . . . . .	67
	Visualiser ses données . . . . .	175
<b>ncol</b> (base)	Listes et Tableaux de données . . . . .	87
	Premier travail avec des données . . . . .	33
<b>nesting</b> (tidyverse)	Réorganiser ses données avec tidyverse . . . . .	299
<b>next</b> (base)	Structures conditionnelles . . . . .	945
<b>nolabel_to_na</b> (labelled)	Facteurs et vecteurs labellisés . . . . .	117
<b>nrow</b> (base)	Listes et Tableaux de données . . . . .	87
	Premier travail avec des données . . . . .	33
<b>nth</b> (dplyr)	Manipulations avancées avec data.table . . . . .	251
<b>NULL</b> (base)	Vecteurs, indexation et assignation . . . . .	67

<b>number</b> (scales)	Mettre en forme des nombres . . . . .	1019
	Trajectoires de soins: un exemple de données longitudinales . . . . .	705

**O**

<b>odds.ratio</b> (questionr)	Comparaisons (moyennes et proportions) . . . . .	485
	Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>oneway.test</b> (stats)	Comparaisons (moyennes et proportions) . . . . .	485
<b>or_plot</b> (final_fit)	Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>or_plot</b> (finalfilt)	Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>order</b> (base)	Tris . . . . .	257
<b>order</b> (data.table)	Tris . . . . .	257
<b>ordgee</b> (geepack)	Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>ordinal</b> (scales)	Mettre en forme des nombres . . . . .	1019
<b>ordLORgee</b> (multgee)	Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>P</b>		
<b>paged_table</b> (rmarkdown)	R Markdown: les rapports automatisés . . . . .	979
<b>pairs</b> (graphics)	Graphiques univariés et bivariés avec ggplot2 . . . . .	417
	Statistique bivariée . . . . .	365

<b>panel.xyplot</b> (lattice)	
lattice : graphiques et formules . . . . .	879
<b>par</b> (graphics)	
Analyse des correspondances multiples (ACM) . . . . .	557
Comparaisons (moyennes et proportions) . . . . .	485
<b>paste</b> (base)	
Annotations mathématiques . . . . .	1097
Manipuler du texte avec stringr . . . . .	289
<b>paste0</b> (base)	
Manipuler du texte avec stringr . . . . .	289
<b>PCA</b> (FactoMineR)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>pdf</b> (grDevices)	
Export de graphiques . . . . .	331
<b>percent</b> (scales)	
Exemples de graphiques avancés . . . . .	839
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Mettre en forme des nombres . . . . .	1019
<b>Period</b> (lubridate)	
Calculer un âge . . . . .	1103
<b>pi</b> (base)	
Vecteurs, indexation et assignation . . . . .	67
<b>pie</b> (graphics)	
Statistique univariée . . . . .	339
<b>pirateplot</b> (yarr)	
Statistique bivariée . . . . .	365
<b>pivot_longer</b> (tidyverse)	
Exemples de graphiques avancés . . . . .	839
Réorganiser ses données avec tidyverse . . . . .	299
<b>pivot_wider</b> (tidyverse)	
Réorganiser ses données avec tidyverse . . . . .	299
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>plot</b> (effects)	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>plot</b> (FactoMineR)	
Analyse des correspondances multiples (ACM) . . . . .	557
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>plot</b> (graphics)	
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
Premier travail avec des données . . . . .	33
Statistique bivariée . . . . .	365
Statistique univariée . . . . .	339
<b>plot</b> (stats)	
Analyse de séquences . . . . .	679
Classification ascendante hiérarchique (CAH) . . . . .	595
Statistique univariée . . . . .	339
Statistique univariée . . . . .	339
<b>plot.prevalence msm</b> (msm)	
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>plot_grid</b> (cowplot)	
Combiner plusieurs graphiques . . . . .	825
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>plotellipses</b> (FactoMineR)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>plotmath</b> (grDevices)	
Annotations mathématiques . . . . .	1097
<b>png</b> (grDevices)	
Export de graphiques . . . . .	331
<b>poly</b> (stats)	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>position_fill</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>position_stack</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>postprob</b> (lcmm)	
Trajettoires de soins: un exemple de données longitudinales . . . . .	705
<b>postscript</b> (grDevices)	
Export de graphiques . . . . .	331
<b>predict</b> (stats)	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>predict</b> (survey)	
Régression logistique binaire, multinomiale et ordinale . . . . .	507

<b>prevalence.msm</b> (msm)	
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>princomp</b> (stats)	
Analyse des correspondances multiples (ACM) .....	557
<b>print</b> (questionr)	
Statistique bivariée .....	365
<b>progress_bar</b> (progress)	
Structures conditionnelles .....	945
<b>prop</b> (questionr)	
Statistique bivariée .....	365
<b>prop.ci</b> (JLutils)	
Intervalles de confiance .....	477
<b>prop.ci.lower</b> (JLutils)	
Intervalles de confiance .....	477
<b>prop.ci.upper</b> (JLutils)	
Intervalles de confiance .....	477
<b>prop.table</b> (base)	
Statistique univariée .....	339
<b>prop.test</b> (stats)	
Comparaisons (moyennes et proportions) .....	485
Intervalles de confiance .....	477
<b>ptol_pal</b> (ggthemes)	
Couleurs et Palettes .....	1021
<b>pvalue</b> (scales)	
Mettre en forme des nombres .....	1019
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>pyramid_chart</b> (ggcharts)	
Étendre ggplot2 .....	793

## Q

<b>qnorm</b> (stats)	
Comparaisons (moyennes et proportions) .....	485
<b>qplot</b> (ggplot2)	
Introduction à ggplot2, la grammaire des graphiques .....	395

<b>quant.cut</b> (questionr)	
Recodage de variables .....	197

<b>quantile</b> (stats)	
Statistique univariée .....	339

## R

<b>rainbow</b> (grDevices)	
Couleurs et Palettes .....	1021
<b>range</b> (base)	
Statistique univariée .....	339
<b>range</b> (stats)	
Formules .....	929
<b>rank</b> (base)	
Analyse de survie .....	651
<b>raster</b> (raster)	
Import de données .....	145
<b>rbind</b> (base)	
Fusion de tables .....	269
<b>rbind.fill</b> (plyr)	
Fusion de tables .....	269
<b>read.csv</b> (utils)	
Import de données .....	145
Introduction à ggplot2, la grammaire des graphiques .....	395
<b>read.csv2</b> (utils)	
Import de données .....	145
<b>read.dbf</b> (foreign)	
Import de données .....	145
<b>read.delim</b> (utils)	
Import de données .....	145
<b>read.delim2</b> (utils)	
Import de données .....	145
<b>read.dta</b> (foreign)	
Import de données .....	145
<b>read.spss</b> (foreign)	
Import de données .....	145

<b>read.table</b> (utils)	
Import de données .....	145
<b>read.xlsx</b> (openxlsx)	
Import de données .....	145
<b>read.xlsx</b> (xlsx)	
Import de données .....	145
<b>read.xlsx2</b> (xlsx)	
Import de données .....	145
<b>read.xport</b> (foreign)	
Import de données .....	145
<b>read_csv</b> (readr)	
Import de données .....	145
Introduction à ggplot2, la grammaire des graphiques .....	395
<b>read_csv2</b> (readr)	
Import de données .....	145
<b>read_delim</b> (readr)	
Import de données .....	145
<b>read_dta</b> (haven)	
Import de données .....	145
<b>read_excel</b> (readxl)	
Import de données .....	145
<b>read_feather</b> (feather)	
Export de données .....	327
<b>read_sas</b> (haven)	
Import de données .....	145
<b>read_spss</b> (haven)	
Import de données .....	145
<b>read_stata</b> (haven)	
Import de données .....	145
<b>read_tsv</b> (readr)	
Import de données .....	145
<b>readAsciiGrid</b> (maptools)	
Import de données .....	145
<b>readShapeLines</b> (maptools)	
Import de données .....	145
<b>readShapePoints</b> (maptools)	
Import de données .....	145
<b>readShapePoly</b> (maptools)	
Import de données .....	145
<b>readShapeSpatial</b> (maptools)	
Import de données .....	145
<b>recode</b> (dplyr)	
Manipuler les données avec dplyr .....	233
Recodage de variables .....	197
<b>recode_haven_labelled</b> (labelled)	
Recodage de variables .....	197
<b>recode_factor</b> (dplyr)	
Manipuler les données avec dplyr .....	233
<b>rect.hclust</b> (stats)	
Classification ascendante hiérarchique (CAH) .....	595
<b>regex</b> (stringr)	
Manipuler du texte avec stringr .....	289
<b>registerDoMC</b> (doMC)	
Vectorisation .....	953
<b>relevel</b> (stats)	
Analyse de survie .....	651
Intervalles de confiance .....	477
Régression logistique binaire, multinomiale et ordinale .....	507
<b>relocate</b> (dplyr)	
Manipuler les données avec dplyr .....	233
<b>relrisk</b> (mosaic)	
Comparaisons (moyennes et proportions) .....	485
<b>remove.packages</b> (utils)	
Extensions (installation, mise à jour) .....	55
<b>remove_labels</b> (labelled)	
Recodage de variables .....	197
<b>rename</b> (dplyr)	
Fusion de tables .....	269
Manipuler les données avec dplyr .....	233
<b>rename.variable</b> (questionr)	
Recodage de variables .....	197
<b>rep</b> (base)	
Vecteurs, indexation et assignation .....	67
<b>repeat</b> (base)	
Structures conditionnelles .....	945

<b>require</b> (base)	
Extensions (installation, mise à jour) . . . . .	55
<b>rgb</b> (grDevices)	
Couleurs et Palettes . . . . .	1021
<b>right_join</b> (dplyr)	
Fusion de tables . . . . .	269
<b>row.names</b> (base)	
Listes et Tableaux de données . . . . .	87
<b>rownames_to_column</b> (tibble)	
Introduction au tidyverse . . . . .	59
<b>rownames_to_columns</b> (tibble)	
Introduction au tidyverse . . . . .	59
<b>rprop</b> (questionr)	
Statistique bivariée . . . . .	365
<b>rug</b> (graphics)	
lattice : graphiques et formules . . . . .	879
Statistique univariée . . . . .	339
<b>runif</b> (stats)	
Analyse de survie . . . . .	651
 <b>S</b>	
<b>s.arrow</b> (ade4)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>s.chull</b> (ade4)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>s.class</b> (ade4)	
Analyse des correspondances multiples (ACM) . . . . .	557
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>s.corcircle</b> (ade4)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>s.freq</b> (JLutils)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>s.hist</b> (ade4)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>s.label</b> (ade4)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>s.value</b> (ade4)	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>sample_frac</b> (dplyr)	
Manipuler les données avec dplyr . . . . .	233
<b>sample_n</b> (dplyr)	
Manipuler les données avec dplyr . . . . .	233
<b>sankeyNetwork</b> (networkD3)	
Autres extensions graphiques . . . . .	897
<b>sapply</b> (base)	
Vectorisation . . . . .	953
<b>save</b> (base)	
Export de données . . . . .	327
Import de données . . . . .	145
<b>save.image</b> (base)	
Import de données . . . . .	145
<b>scale_color_paletteer_c</b> (paletteer)	
Couleurs et Palettes . . . . .	1021
<b>scale_color_paletteer_d</b> (paletteer)	
Couleurs et Palettes . . . . .	1021
<b>scale_colour_brewer</b> (ggplot2)	
Couleurs et Palettes . . . . .	1021
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
<b>scale_colour_viridis_c</b> (ggplot2)	
Couleurs et Palettes . . . . .	1021
<b>scale_colour_viridis_d</b> (ggplot2)	
Couleurs et Palettes . . . . .	1021
<b>scale_coulour_gradientn</b> (ggplot2)	
Couleurs et Palettes . . . . .	1021
<b>scale_fill_brewer</b> (ggplot2)	
Couleurs et Palettes . . . . .	1021
<b>scale_fill_bright</b> (khroma)	
Exemples de graphiques avancés . . . . .	839
<b>scale_fill_gradientn</b> (ggplot2)	
Couleurs et Palettes . . . . .	1021
<b>scale_fill_manual</b> (ggplot2)	
Exemples de graphiques avancés . . . . .	839
<b>scale_fill_viridis_c</b> (ggplot2)	
Couleurs et Palettes . . . . .	1021

<b>scale_fill_viridis_d</b> (ggplot2)	
Couleurs et Palettes .....	1021
<b>scale_x_continuous</b> (ggplot2)	
Introduction à ggplot2, la grammaire des graphiques .....	395
<b>scale_y_continuous</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 .....	417
<b>scatter</b> (ade4)	
Analyse des correspondances multiples (ACM) .....	557
<b>score</b> (ade4)	
Analyse des correspondances multiples (ACM) .....	557
<b>screeplot</b> (ade4)	
Analyse des correspondances multiples (ACM) .....	557
<b>sd</b> (stats)	
Premier contact .....	13
Statistique univariée .....	339
<b>select</b> (dplyr)	
Manipuler les données avec dplyr .....	233
Réorganiser ses données avec tidyr .....	299
<b>select_helpers</b> (tidyselect)	
Réorganiser ses données avec tidyr .....	299
<b>semi_join</b> (dplyr)	
Fusion de tables .....	269
<b>separate</b> (tidyr)	
Manipuler du texte avec stringr .....	289
Réorganiser ses données avec tidyr .....	299
<b>seq</b> (base)	
Vecteurs, indexation et assignation .....	67
<b>seq_heatmap</b> (JLutils)	
Analyse de séquences .....	679
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>seqcost</b> (TraMineR)	
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>seqdef</b> (TraMineR)	
Analyse de séquences .....	679
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>seqdist</b> (TraMineR)	
Analyse de séquences .....	679
Classification ascendante hiérarchique (CAH) .....	595
<b>seqdplot</b> (TraMineR)	
Analyse de séquences .....	679
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>seqfplot</b> (TraMineR)	
Analyse de séquences .....	679
<b>seqHplot</b> (TraMineR)	
Analyse de séquences .....	679
<b>seqIplot</b> (TraMineR)	
Analyse de séquences .....	679
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>seqLength</b> (TraMineR)	
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>seqmsplot</b> (TraMineR)	
Analyse de séquences .....	679
<b>seqmtpplot</b> (TraMineR)	
Analyse de séquences .....	679
<b>seqplot</b> (TraMineR)	
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>seqrep</b> (TraMineR)	
Analyse de séquences .....	679
<b>seqrplot</b> (TraMineR)	
Analyse de séquences .....	679
<b>seqsubm</b> (TraMineR)	
Analyse de séquences .....	679
<b>seqtreedisplay</b> (TraMineR)	
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>setDF</b> (data.table)	
Manipulations avancées avec data.table .....	251
<b>setDT</b> (data.table)	
Analyse de survie .....	651
Manipulations avancées avec data.table .....	251
Trajectoires de soins: un exemple de données longitudinales .....	705

<b>setorder</b> (data.table)	
Tris . . . . .	257
<b>setwd</b> (base)	
Organiser ses fichiers . . . . .	135
<b>shapiro.test</b> (stats)	
Comparaisons (moyennes et proportions) . . . . .	485
<b>skim</b> (skimr)	
Visualiser ses données . . . . .	175
<b>slice</b> (dplyr)	
Manipuler les données avec dplyr . . . . .	233
<b>smean.sd</b> (Hmisc)	
Formules . . . . .	929
<b>smoothScatter</b> (graphics)	
Statistique bivariée . . . . .	365
<b>sort</b> (base)	
Statistique univariée . . . . .	339
Tris . . . . .	257
<b>sort_val_labels</b> (labelled)	
Facteurs et vecteurs labellisés . . . . .	117
<b>source</b> (base)	
Organiser ses fichiers . . . . .	135
<b>SpatialGridDataFrame</b> (sp)	
Import de données . . . . .	145
<b>spread</b> (tidyverse)	
Réorganiser ses données avec tidyverse . . . . .	299
<b>starts_with</b> (dplyr)	
Manipuler les données avec dplyr . . . . .	233
<b>stat_count</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>stat_density_2d</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>stat_ecdf</b> (ggplot2)	
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
<b>stat_prop</b> (GGally)	
Exemples de graphiques avancés . . . . .	839
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>statetable.msm</b> (msm)	
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>step</b> (stats)	
Analyse de survie . . . . .	651
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>stepAIC</b> (MASS)	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>str</b> (utils)	
Formules . . . . .	929
Listes et Tableaux de données . . . . .	87
Premier travail avec des données . . . . .	33
Statistique univariée . . . . .	339
Visualiser ses données . . . . .	175
<b>str_c</b> (stringr)	
Manipuler du texte avec stringr . . . . .	289
<b>str_count</b> (stringr)	
Manipuler du texte avec stringr . . . . .	289
<b>str_detect</b> (stringr)	
Manipuler du texte avec stringr . . . . .	289
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>str_extract</b> (stringr)	
Manipuler du texte avec stringr . . . . .	289
<b>str_extract_all</b> (stringr)	
Manipuler du texte avec stringr . . . . .	289
<b>str_glue_data</b> (stringr)	
Manipuler du texte avec stringr . . . . .	289
<b>str_replace</b> (stringr)	
Manipuler du texte avec stringr . . . . .	289
<b>str_replace_all</b> (stringr)	
Manipuler du texte avec stringr . . . . .	289
<b>str_split</b> (stringr)	
Manipuler du texte avec stringr . . . . .	289
<b>str_sub</b> (stringr)	
Analyse de séquences . . . . .	679
Manipuler du texte avec stringr . . . . .	289
<b>str_subset</b> (stringr)	
Manipuler du texte avec stringr . . . . .	289

<b>str_to_lower</b> (stringr)	
Manipuler du texte avec stringr .....	289
<b>str_to_title</b> (stringr)	
Manipuler du texte avec stringr .....	289
<b>str_to_upper</b> (stringr)	
Manipuler du texte avec stringr .....	289
<b>subset</b> (base)	
Sous-ensembles .....	261
<b>subset</b> (data.table)	
Sous-ensembles .....	261
<b>subset</b> (survey)	
Définir un plan d'échantillonnage complexe .....	501
Données pondérées .....	465
<b>substitute</b> (base)	
Annotations mathématiques .....	1097
<b>sum</b> (base)	
Vectorisation .....	953
<b>summarise</b> (dplyr)	
Formules .....	929
Manipuler les données avec dplyr .....	233
Réorganiser ses données avec tidyr .....	299
<b>summarize</b> (Hmisc)	
Formules .....	929
<b>summary</b> (ade4)	
Analyse des correspondances multiples (ACM) .....	557
<b>summary</b> (base)	
Conditions et comparaisons .....	925
lattice : graphiques et formules .....	879
Listes et Tableaux de données .....	87
Premier travail avec des données .....	33
Recodage de variables .....	197
Statistique univariée .....	339
Trajectoires de soins: un exemple de données longitudinales .....	705
Visualiser ses données .....	175
<b>summary</b> (stats)	
Régression logistique binaire, multinomiale et ordinale .....	507
<b>summary_factorlist</b> (finalfit)	
Régression logistique binaire, multinomiale et ordinale .....	507
<b>Surv</b> (survival)	
Analyse de survie .....	651
<b>survdiff</b> (survival)	
Analyse de survie .....	651
<b>survfit</b> (survival)	
Analyse de survie .....	651
<b>svg</b> (grDevices)	
Export de graphiques .....	331
<b>svyboxplot</b> (survey)	
Données pondérées .....	465
<b>svyby</b> (survey)	
Données pondérées .....	465
<b>svychisq</b> (survey)	
Comparaisons (moyennes et proportions) .....	485
Données pondérées .....	465
<b>svyciprop</b> (survey)	
Données pondérées .....	465
Intervalles de confiance .....	477
<b>svyclm</b> (svrepmisc)	
Régression logistique binaire, multinomiale et ordinale .....	507
<b>svycoxph</b> (survey)	
Analyse de survie .....	651
<b>svydesign</b> (survey)	
Définir un plan d'échantillonnage complexe .....	501
Données pondérées .....	465
<b>svyglm</b> (survey)	
Données pondérées .....	465
Régression logistique binaire, multinomiale et ordinale .....	507
<b>svyhist</b> (survey)	
Données pondérées .....	465
<b>svykm</b> (survey)	
Analyse de survie .....	651
<b>svymean</b> (survey)	
Données pondérées .....	465
<b>svymultinom</b> (svrepmisc)	
Régression logistique binaire, multinomiale et ordinale .....	507

<b>svyolr</b> (survey)	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>svyplot</b> (survey)	
Données pondérées . . . . .	465
<b>svyquantile</b> (survey)	
Données pondérées . . . . .	465
<b>svyranktest</b> (survey)	
Comparaisons (moyennes et proportions) . . . . .	485
<b>svyratio</b> (survey)	
Données pondérées . . . . .	465
<b>svytable</b> (survey)	
Comparaisons (moyennes et proportions) . . . . .	485
Données pondérées . . . . .	465
<b>svytotals</b> (survey)	
Données pondérées . . . . .	465
<b>svyttest</b> (survey)	
Comparaisons (moyennes et proportions) . . . . .	485
Données pondérées . . . . .	465
<b>svyvar</b> (survey)	
Données pondérées . . . . .	465
<b>switch</b> (base)	
Structures conditionnelles . . . . .	945
 <b>T</b>	
<b>t.test</b> (stats)	
Comparaisons (moyennes et proportions) . . . . .	485
Intervalles de confiance . . . . .	477
<b>table</b> (base)	
Comparaisons (moyennes et proportions) . . . . .	485
Conditions et comparaisons . . . . .	925
Données pondérées . . . . .	465
Facteurs et vecteurs labellisés . . . . .	117
Formules . . . . .	929
Intervalles de confiance . . . . .	477
Premier travail avec des données . . . . .	33
R Markdown : les rapports automatisés . . . . .	979
Recodage de variables . . . . .	197
Statistique bivariée . . . . .	365
Statistique univariée . . . . .	339
<b>tagged_na</b> (haven)	
Import de données . . . . .	145
<b>tail</b> (utils)	
Listes et Tableaux de données . . . . .	87
Premier travail avec des données . . . . .	33
<b>tally</b> (dplyr)	
Manipuler les données avec dplyr . . . . .	233
<b>tapply</b> (base)	
Comparaisons (moyennes et proportions) . . . . .	485
Données pondérées . . . . .	465
Sous-ensembles . . . . .	261
Statistique bivariée . . . . .	365
Vectorisation . . . . .	953
<b>tbl</b> (dplyr)	
Import de données . . . . .	145
<b>tbl_cross</b> (gtsummary)	
Statistique bivariée . . . . .	365
<b>tbl_df</b> (dplyr)	
Analyse de survie . . . . .	651
Import de données . . . . .	145
<b>tbl_merge</b> (gtsummary)	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>tbl_regression</b> (gtsummary)	
R Markdown : les rapports automatisés . . . . .	979
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>tbl_summary</b> (gtsummary)	
Comparaisons (moyennes et proportions) . . . . .	485
Données pondérées . . . . .	465
R Markdown : les rapports automatisés . . . . .	979
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Statistique bivariée . . . . .	365
Statistique univariée . . . . .	339
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>tbl_svysummary</b> (gtsummary)	
Comparaisons (moyennes et proportions) . . . . .	485
Données pondérées . . . . .	465
R Markdown : les rapports automatisés . . . . .	979
<b>terrain.colors</b> (grDevices)	
Couleurs et Palettes . . . . .	1021

<b>theme</b> (ggplot2)	
Exemples de graphiques avancés .....	839
Graphiques univariés et bivariés avec ggplot2 .....	417
<b>theme_minimal</b> (ggplot2)	
Exemples de graphiques avancés .....	839
<b>theme_tufte</b> (ggthemes)	
Étendre ggplot2 .....	793
<b>tibble</b> (tibble)	
Introduction au tidyverse .....	59
Manipuler les données avec dplyr .....	233
<b>tidy_plus_plus</b> (broom.helpers)	
Régression logistique binaire, multinomiale et ordinale .....	507
<b>tidy</b> (broom)	
Analyse de survie .....	651
Graphiques univariés et bivariés avec ggplot2 .....	417
Graphiques univariés et bivariés avec ggplot2 .....	417
Régression logistique binaire, multinomiale et ordinale .....	507
Régression logistique binaire, multinomiale et ordinale .....	507
Régression logistique binaire, multinomiale et ordinale .....	507
<b>tidy</b> (svrepmiss)	
Régression logistique binaire, multinomiale et ordinale .....	507
<b>tidy.hazard.msm</b> (JLutils)	
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>tidy_plus_plus</b> (broom.helpers)	
Régression logistique binaire, multinomiale et ordinale .....	507
<b>tiff</b> (grDevices)	
Export de graphiques .....	331
<b>time_length</b> (lubridate)	
Analyse de survie .....	651
Calculer un âge .....	1103
<b>to_character</b> (labelled)	
Recodage de variables .....	197
<b>to_factor</b> (labelled)	
Analyse de survie .....	651
Graphiques univariés et bivariés avec ggplot2 .....	417
Recodage de variables .....	197
Régression logistique binaire, multinomiale et ordinale .....	507
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>ToothGrowth</b> (datasets)	
lattice:graphiques et formules .....	879
<b>topo.colors</b> (grDevices)	
Couleurs et Palettes .....	1021
<b>trunc</b> (base)	
Analyse de survie .....	651
<b>txtProgressBar</b> (utils)	
Structures conditionnelles .....	945
<b>typeof</b> (base)	
Facteurs et vecteurs labellisés .....	117
<b>U</b>	
<b>unclass</b> (base)	
Recodage de variables .....	197
<b>ungroup</b> (dplyr)	
Manipuler les données avec dplyr .....	233
<b>unit</b> (scales)	
Mettre en forme des nombres .....	1019
<b>unite</b> (tidyverse)	
Réorganiser ses données avec tidyverse .....	299
<b>unlabelled</b> (labelled)	
Recodage de variables .....	197
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>unlist</b> (base)	
Vectorisation .....	953
<b>uname</b> (base)	
Vecteurs, indexation et assignation .....	67
<b>update.packages</b> (utils)	
Extensions (installation, mise à jour) .....	55
Installation de R et RStudio .....	11

<b>user_na_to_na</b> (labelled)	
Import de données .....	145
<b>V</b>	
<b>val_label</b> (labelled)	
Facteurs et vecteurs labellisés .....	117
<b>val_labels</b> (labelled)	
Facteurs et vecteurs labellisés .....	117
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>val_labels_to_na</b> (labelled)	
Facteurs et vecteurs labellisés .....	117
<b>vapply</b> (base)	
Vectorisation .....	953
<b>var</b> (base)	
Données pondérées .....	465
<b>var</b> (stats)	
Premier contact .....	13
<b>var.test</b> (stats)	
Comparaisons (moyennes et proportions) .....	485
<b>var_label</b> (labelled)	
Facteurs et vecteurs labellisés .....	117
Régression logistique binaire, multinomiale et ordinale .....	507
<b>Vectorize</b> (base)	
Vectorisation .....	953
<b>vglm</b> (VGAM)	
Régression logistique binaire, multinomiale et ordinale .....	507
<b>View</b> (utils)	
Listes et Tableaux de données .....	87
Premier travail avec des données .....	33
Visualiser ses données .....	175
<b>vif</b> (car)	
Multicolinéarité dans la régression .....	645

**W**

<b>wcClusterQuality</b> (WeightedCluster)	
Trajectoires de soins: un exemple de données longitudinales .....	705
<b>wcKMedoids</b> (WeightedCluster)	
Trajettoires de soins: un exemple de données longitudinales .....	705
<b>wcSilhouetteObs</b> (WeightedCluster)	
Trajettoires de soins: un exemple de données longitudinales .....	705
<b>weights</b> (survey)	
Données pondérées .....	465
<b>while</b> (base)	
Structures conditionnelles .....	945
<b>win.metafile</b> (grDevices)	
Export de graphiques .....	331
<b>write.csv</b> (utils)	
Export de données .....	327
<b>write.dbf</b> (foreign)	
Export de données .....	327
<b>write.dta</b> (foreign)	
Export de données .....	327
<b>write.foreign</b> (foreign)	
Export de données .....	327
<b>write.table</b> (utils)	
Export de données .....	327
<b>write.xlsx</b> (openxlsx)	
Export de données .....	327
<b>write.xlsx</b> (xlsx)	
Export de données .....	327
<b>write_csv</b> (readr)	
Export de données .....	327
<b>write_csv</b> (readr)	
Export de données .....	327
<b>write_delim</b> (readr)	
Export de données .....	327

<b>write_dta</b> (haven)	
Export de données .....	327
Export de données .....	327
<b>write_feather</b> (feather)	
Export de données .....	327
<b>write_sas</b> (haven)	
Export de données .....	327
<b>write_sav</b> (haven)	
Export de données .....	327
Export de données .....	327
<b>write_tsv</b> (reader)	
Export de données .....	327
<b>writeAsciiGrid</b> (maptools)	
Export de données .....	327
<b>writeLinesShape</b> (maptools)	
Export de données .....	327
<b>writePointsShape</b> (maptools)	
Export de données .....	327
<b>writePolyShape</b> (maptools)	
Export de données .....	327

**X****xlab** (ggplot2)

Graphiques univariés et bivariés avec ggplot2 .... 417

**xtabs** (stats)

Comparaisons (moyennes et proportions) .....	485
Formules .....	929
Graphiques univariés et bivariés avec ggplot2 ....	417
Recodage de variables .....	197
Statistique bivariée .....	365

**xyplot** (lattice)

lattice : graphiques et formules .... 879

**Y****ylab** (ggplot2)

Graphiques univariés et bivariés avec ggplot2 .... 417



# Index des extensions

## A

### **ade4**

Analyse des correspondances multiples (ACM) . . . . .	557
Classification ascendante hiérarchique (CAH) . . . . .	595
Données pondérées . . . . .	465
Extensions (installation, mise à jour) . . . . .	55

## B

### **base**

Analyse de survie . . . . .	651
Annotations mathématiques . . . . .	1097
Comparaisons (moyennes et proportions) . . . . .	485
Conditions et comparaisons . . . . .	925
Définir un plan d'échantillonnage complexe . . . . .	501
Données pondérées . . . . .	465
Export de données . . . . .	327
Extensions (installation, mise à jour) . . . . .	55
Facteurs et vecteurs labellisés . . . . .	117
Formules . . . . .	929
Fusion de tables . . . . .	269
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Import de données . . . . .	145
Intervalles de confiance . . . . .	477
lattice : graphiques et formules . . . . .	879
Listes et Tableaux de données . . . . .	87
Manipulations avancées avec data.table . . . . .	251
Manipuler du texte avec stringr . . . . .	289
Manipuler les données avec dplyr . . . . .	233
Organiser ses fichiers . . . . .	135
Où trouver de l'aide ? . . . . .	163
Premier contact . . . . .	13
Premier travail avec des données . . . . .	33
R Markdown : les rapports automatisés . . . . .	979
Recodage de variables . . . . .	197
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Sous-ensembles . . . . .	261
Statistique bivariée . . . . .	365
Statistique univariée . . . . .	339
Structures conditionnelles . . . . .	945
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
Tris . . . . .	257
Vecteurs, indexation et assignation . . . . .	67
Vectorisation . . . . .	953
Visualiser ses données . . . . .	175

**beepR**

Structures conditionnelles ..... 945

**breakDown**

Effets d’interaction dans un modèle ..... 617

**broom**

Analyse de survie ..... 651

Graphiques univariés et bivariés avec ggplot2 ..... 417

**Modèles à effets aléatoires (modèles mixtes et GEE)**

Régression logistique binaire, multinomiale et ordinale ..... 507

**broom.helpers**

Régression logistique binaire, multinomiale et ordinale ..... 507

**C****car**

Multicolinéarité dans la régression ..... 645

Régression logistique binaire, multinomiale et ordinale ..... 507

**cartography**

Cartes ..... 895

**circlize**

Autres extensions graphiques ..... 897

**cluster**

Analyse de séquences ..... 679

Classification ascendante hiérarchique (CAH) ..... 595

**codebook**

Visualiser ses données ..... 175

**cowplot**

Combiner plusieurs graphiques ..... 825

Régression logistique binaire, multinomiale et ordinale ..... 507

**D****data.table**

Analyse de survie ..... 651

Fusion de tables ..... 269

Manipulations avancées avec data.table ..... 251

Manipuler les données avec dplyr ..... 233

Recodage de variables ..... 197

Réorganiser ses données avec tidyr ..... 299

Sous-ensembles ..... 261

Trajectoires de soins: un exemple de données longitudinales ..... 705

Tris ..... 257

Vectorisation ..... 953

**DataExplorer**

Visualiser ses données ..... 175

**dataMaid**

Visualiser ses données ..... 175

**datasets**

lattice : graphiques et formules ..... 879

**DBI**

Import de données ..... 145

**dbplyr**

Import de données ..... 145

**dendextend**

Classification ascendante hiérarchique (CAH) ..... 595

**devtools**

Analyse de séquences ..... 679

Analyse des correspondances multiples (ACM) ..... 557

Calculer un âge ..... 1103

Classification ascendante hiérarchique (CAH) ..... 595

Combiner plusieurs graphiques ..... 825

Données pondérées ..... 465

Extensions (installation, mise à jour) ..... 55

Intervalles de confiance ..... 477

**DiagrammeR**

Autres extensions graphiques ..... 897

Trajectoires de soins: un exemple de données longitudinales ..... 705

**distill**

R Markdown : les rapports automatisés ..... 979

**doBy****Modèles à effets aléatoires (modèles mixtes et GEE)**

<b>doMC</b>	
Vectorisation . . . . .	953
<b>doSMP</b>	
Vectorisation . . . . .	953
<b>dplyr</b>	
Analyse de survie . . . . .	651
Définir un plan d'échantillonnage complexe . . . . .	501
Données pondérées . . . . .	465
Extensions (installation, mise à jour) . . . . .	55
Facteurs et vecteurs labellisés . . . . .	117
Fonctions à fenêtre . . . . .	287
Formules . . . . .	929
Fusion de tables . . . . .	269
Import de données . . . . .	145
Introduction au tidyverse . . . . .	59
Listes et Tableaux de données . . . . .	87
Manipulations avancées avec data.table . . . . .	251
Manipuler du texte avec stringr . . . . .	289
Manipuler les données avec dplyr . . . . .	233
Recodage de variables . . . . .	197
Réorganiser ses données avec tidyr . . . . .	299
Scraping . . . . .	317
Sous-ensembles . . . . .	261
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
Tris . . . . .	257
Vectorisation . . . . .	953
Visualiser ses données . . . . .	175
<b>DT</b>	
R Markdown : les rapports automatisés . . . . .	979
<b>dtplyr</b>	
Manipulations avancées avec data.table . . . . .	251
Manipuler les données avec dplyr . . . . .	233
<b>E</b>	
<b>ecdf</b>	
Statistique univariée . . . . .	339
<b>effects</b>	
Effets d'interaction dans un modèle . . . . .	617
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>esquisse</b>	
ggplot2 et la grammaire des graphiques . . . . .	789
<b>explor</b>	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>extrafont</b>	
Étendre ggplot2 . . . . .	793
Exemples de graphiques avancés . . . . .	839
<b>F</b>	
<b>factoextra</b>	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>FactoInvestigate</b>	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>FactoMineR</b>	
Analyse des correspondances multiples (ACM) . . . . .	557
Classification ascendante hiérarchique (CAH) . . . . .	595
Données pondérées . . . . .	465
<b>Factoshiny</b>	
Analyse des correspondances multiples (ACM) . . . . .	557
<b>fastcluster</b>	
Classification ascendante hiérarchique (CAH) . . . . .	595
<b>feather</b>	
Export de données . . . . .	327
<b>final_fit</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>finalfilt</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>finalfit</b>	
Régression logistique binaire, multinomiale et ordinale . . . . .	507
<b>flextable</b>	
R Markdown : les rapports automatisés . . . . .	979
<b>forcats</b>	
Exemples de graphiques avancés . . . . .	839
Extensions (installation, mise à jour) . . . . .	55
Introduction au tidyverse . . . . .	59
Manipuler du texte avec stringr . . . . .	289
Manipuler les données avec dplyr . . . . .	233
Recodage de variables . . . . .	197
Trajectoires de soins: un exemple de données longitudinales . . . . .	705

**foreign**

- Export de données ..... 327  
 Import de données ..... 145

**forestmodel**

- Régression logistique binaire, multinomiale et ordinale ..... 507

**formattable**

- R Markdown : les rapports automatisés ..... 979

**G****gapminder**

- Étendre ggplot2 ..... 793  
 Réorganiser ses données avec tidyverse ..... 299

**gee**

- Modèles à effets aléatoires (modèles mixtes et GEE)

**geepack**

- Modèles à effets aléatoires (modèles mixtes et GEE)  
 Trajectoires de soins: un exemple de données longitudinales ..... 705

**ggalluvial**

- Autres extensions graphiques ..... 897

**GGally**

- Analyse de survie ..... 651  
 Autres extensions graphiques ..... 897  
 Effets d’interaction dans un modèle ..... 617  
 Exemples de graphiques avancés ..... 839  
 Graphiques univariés et bivariés avec ggplot2 ..... 417  
 Introduction à ggplot2, la grammaire des graphiques ..... 395  
 Régression logistique binaire, multinomiale et ordinale ..... 507  
 Trajectoires de soins: un exemple de données longitudinales ..... 705

**ggalt**

- Étendre ggplot2 ..... 793  
 Graphiques univariés et bivariés avec ggplot2 ..... 417

**ganimate**

- Étendre ggplot2 ..... 793

**ggcharts**

- Étendre ggplot2 ..... 793

**ggdendro**

- Autres extensions graphiques ..... 897  
 Classification ascendante hiérarchique (CAH) ..... 595

**ggeffect**

- Régression logistique binaire, multinomiale et ordinale ..... 507

**ggeffects**

- Effets d’interaction dans un modèle ..... 617  
 Régression logistique binaire, multinomiale et ordinale ..... 507  
 Trajectoires de soins: un exemple de données longitudinales ..... 705

**ggendro**

- Autres extensions graphiques ..... 897

**ggfortify**

- Introduction à ggplot2, la grammaire des graphiques ..... 395

**gghighlight**

- Étendre ggplot2 ..... 793

**gghighlight**

- Étendre ggplot2 ..... 793

**ggmap**

- Cartes ..... 895  
 Introduction à ggplot2, la grammaire des graphiques ..... 395

**ggmosaic**

- Étendre ggplot2 ..... 793  
 Graphiques univariés et bivariés avec ggplot2 ..... 417

**ggmulti**

- Effets d’interaction dans un modèle ..... 617

**ggpirate**

- Étendre ggplot2 ..... 793  
 Graphiques univariés et bivariés avec ggplot2 ..... 417

<b>ggplot2</b>	
Analyse de séquences . . . . .	679
Analyse de survie . . . . .	651
Analyse des correspondances multiples (ACM) . . . . .	557
Annotations mathématiques . . . . .	1097
Autres extensions graphiques . . . . .	897
Cartes . . . . .	895
Classification ascendante hiérarchique (CAH) . . . . .	595
Combiner plusieurs graphiques . . . . .	825
Couleurs et Palettes . . . . .	1021
Données pondérées . . . . .	465
Effets d'interaction dans un modèle . . . . .	617
Étendre ggplot2 . . . . .	793
Exemples de graphiques avancés . . . . .	839
Export de graphiques . . . . .	331
Extensions (installation, mise à jour) . . . . .	55
Formules . . . . .	929
ggplot2 et la grammaire des graphiques . . . . .	789
Graphiques interactifs . . . . .	875
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
Introduction au tidyverse . . . . .	59
lattice : graphiques et formules . . . . .	879
R Markdown : les rapports automatisés . . . . .	979
Recodage de variables . . . . .	197
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Réorganiser ses données avec tidyr . . . . .	299
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
<b>ggplotAssist</b>	
ggplot2 et la grammaire des graphiques . . . . .	789
<b>ggpubr</b>	
Autres extensions graphiques . . . . .	897
Étendre ggplot2 . . . . .	793
<b>grepel</b>	
Étendre ggplot2 . . . . .	793
<b>ggridges</b>	
Étendre ggplot2 . . . . .	793
<b>gsosci</b>	
Couleurs et Palettes . . . . .	1021
<b>ggThemeAssist</b>	
ggplot2 et la grammaire des graphiques . . . . .	789
<b>ggthemes</b>	
Couleurs et Palettes . . . . .	1021
Étendre ggplot2 . . . . .	793
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
<b>gvvis</b>	
Graphiques interactifs . . . . .	875
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
<b>gfski</b>	
Étendre ggplot2 . . . . .	793
<b>glmulti</b>	
Effets d'interaction dans un modèle . . . . .	617
<b>glue</b>	
Manipuler du texte avec stringr . . . . .	289
<b>googlesheets</b>	
Import de données . . . . .	145
<b>gpplot2</b>	
Couleurs et Palettes . . . . .	1021
<b>graphics</b>	
Analyse des correspondances multiples (ACM) . . . . .	557
Comparaisons (moyennes et proportions) . . . . .	485
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
lattice : graphiques et formules . . . . .	879
Premier travail avec des données . . . . .	33
Statistique bivariée . . . . .	365
Statistique univariée . . . . .	339
<b>grDevices</b>	
Annotations mathématiques . . . . .	1097
Couleurs et Palettes . . . . .	1021
Export de graphiques . . . . .	331
Statistique univariée . . . . .	339
<b>grid</b>	
lattice : graphiques et formules . . . . .	879
<b>gridExtra</b>	
Modèles à effets aléatoires (modèles mixtes et GEE)	
<b>gt</b>	
Couleurs et Palettes . . . . .	1021
R Markdown : les rapports automatisés . . . . .	979

**gtsummary**

Comparaisons (moyennes et proportions) . . . . .	485
Données pondérées . . . . .	465
Effets d'interaction dans un modèle . . . . .	617
Formules . . . . .	929
R Markdown : les rapports automatisés . . . . .	979
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Statistique bivariée . . . . .	365
Statistique univariée . . . . .	339
Trajectoires de soins: un exemple de données longitudinales . . . . .	705

**H****haven**

Analyse de survie . . . . .	651
Export de données . . . . .	327
Extensions (installation, mise à jour) . . . . .	55
Facteurs et vecteurs labellisés . . . . .	117
Import de données . . . . .	145
Recodage de variables . . . . .	197

**highcharter**

Autres extensions graphiques . . . . .	897
----------------------------------------	-----

**Hmisc**

Facteurs et vecteurs labellisés . . . . .	117
Formules . . . . .	929

**hrbrthemes**

Couleurs et Palettes . . . . .	1021
Étendre ggplot2 . . . . .	793

**J****JLutils**

Analyse de séquences . . . . .	679
Analyse des correspondances multiples (ACM) . . . . .	557
Classification ascendante hiérarchique (CAH) . . . . .	595
Combiner plusieurs graphiques . . . . .	825
Intervalles de confiance . . . . .	477
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Trajectoires de soins: un exemple de données longitudinales . . . . .	705

**K****kableExtra**

R Markdown : les rapports automatisés . . . . .	979
Vectorisation . . . . .	953

**khroma**

Couleurs et Palettes . . . . .	1021
Exemples de graphiques avancés . . . . .	839

**knitr**

Fusion de tables . . . . .	269
R Markdown : les rapports automatisés . . . . .	979
Régression logistique binaire, multinomiale et ordinale . . . . .	507

**L****labelled**

Facteurs et vecteurs labellisés . . . . .	117
-------------------------------------------	-----

**labelled**

Analyse de survie . . . . .	651
Effets d'interaction dans un modèle . . . . .	617
Facteurs et vecteurs labellisés . . . . .	117
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Import de données . . . . .	145
Listes et Tableaux de données . . . . .	87
Recodage de variables . . . . .	197
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Statistique univariée . . . . .	339
Trajectoires de soins: un exemple de données longitudinales . . . . .	705
Visualiser ses données . . . . .	175

**lattice**

Formules . . . . .	929
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
lattice : graphiques et formules . . . . .	879
Modèles à effets aléatoires (modèles mixtes et GEE)	

**latticeExtra**

lattice : graphiques et formules . . . . .	879
Modèles à effets aléatoires (modèles mixtes et GEE)	

**lcmm**

Trajectoires de soins: un exemple de données longitudinales . . . . .	705
-----------------------------------------------------------------------	-----

<b>lemon</b>	Combiner plusieurs graphiques ..... 825
<b>lemon</b>	Combiner plusieurs graphiques ..... 825 Étendre ggplot2 ..... 793
<b>LexisPlotR</b>	Diagramme de Lexis ..... 1111
<b>lme4</b>	Formules ..... 929 Modèles à effets aléatoires (modèles mixtes et GEE)
<b>lubridate</b>	Analyse de survie ..... 651 Calculer un âge ..... 1103 Extensions (installation, mise à jour) ..... 55 Gestion des dates ..... 285 Import de données ..... 145 Scraping ..... 317
 <b>M</b>	
<b>magrittr</b>	Manipuler les données avec dplyr ..... 233
<b>maptools</b>	Export de données ..... 327 Import de données ..... 145
<b>mapview</b>	Cartes ..... 895
<b>MASS</b>	Analyse des correspondances multiples (ACM) ..... 557 Graphiques univariés et bivariés avec ggplot2 ..... 417 Régression logistique binaire, multinomiale et ordinale ..... 507 Statistique bivariée ..... 365
<b>mctest</b>	Multicolinéarité dans la régression ..... 645
<b>memisc</b>	Facteurs et vecteurs labellisés ..... 117
<b>mlogit</b>	Régression logistique binaire, multinomiale et ordinale ..... 507
<b>mosaic</b>	Comparaisons (moyennes et proportions) ..... 485
 <b>N</b>	
<b>msm</b>	Trajectoires de soins: un exemple de données longitudinales ..... 705
<b>msSurv</b>	Trajectoires de soins: un exemple de données longitudinales ..... 705
<b>multgee</b>	Trajectoires de soins: un exemple de données longitudinales ..... 705
 <b>O</b>	
<b>openxlsx</b>	Export de données ..... 327 Import de données ..... 145
<b>ordinal</b>	Régression logistique binaire, multinomiale et ordinale ..... 507 Trajectoires de soins: un exemple de données longitudinales ..... 705

**P****packrat**

Organiser ses fichiers ..... 135

**paletteer**

Couleurs et Palettes ..... 1021

**pander**

R Markdown : les rapports automatisés ..... 979

**patchwork**

Combiner plusieurs graphiques ..... 825

**phia**

Effets d’interaction dans un modèle ..... 617

**plotly**

Graphiques interactifs ..... 875

lattice : graphiques et formules ..... 879

**plyr**

Fusion de tables ..... 269

Vectorisation ..... 953

**printr**

R Markdown : les rapports automatisés ..... 979

**productplots**

Graphiques univariés et bivariés avec ggplot2 ..... 417

**progress**

Structures conditionnelles ..... 945

**purrr**

Extensions (installation, mise à jour) ..... 55

Introduction au tidyverse ..... 59

**Q****question**

Statistique bivariée ..... 365

**questionr**

Analyse de séquences .....	679
Analyse de survie .....	651
Analyse des correspondances multiples (ACM) .....	557
Classification ascendante hiérarchique (CAH) .....	595
Comparaisons (moyennes et proportions) .....	485
Conditions et comparaisons .....	925
Données pondérées .....	465
Effets d’interaction dans un modèle .....	617
Exemples de graphiques avancés .....	839
Facteurs et vecteurs labellisés .....	117
Formules .....	929
Fusion de tables .....	269
Graphiques univariés et bivariés avec ggplot2 .....	417
Import de données .....	145
Intervalles de confiance .....	477
Listes et Tableaux de données .....	87
Multicolinéarité dans la régression .....	645
Premier travail avec des données .....	33
R Markdown : les rapports automatisés .....	979
Recodage de variables .....	197
Régression logistique binaire, multinomiale et ordinale .....	507
Réorganiser ses données avec tidyr .....	299
Sous-ensembles .....	261
Statistique bivariée .....	365
Statistique univariée .....	339
Trajectoires de soins : un exemple de données longitudinales .....	705
Tris .....	257
Visualiser ses données .....	175

**R****raster**

Analyse spatiale .....	787
Import de données .....	145

**RColorBrewer**

Analyse des correspondances multiples (ACM) .....	557
Classification ascendante hiérarchique (CAH) .....	595
Couleurs et Palettes .....	1021
Introduction à ggplot2, la grammaire des graphiques .....	395

**reader**

Export de données .....	327
-------------------------	-----

**readr**

Export de données . . . . .	327
Extensions (installation, mise à jour) . . . . .	55
Formules . . . . .	929
Import de données . . . . .	145
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
Introduction au tidyverse . . . . .	59
Scraping . . . . .	317

**readxl**

Extensions (installation, mise à jour) . . . . .	55
Import de données . . . . .	145
Introduction au tidyverse . . . . .	59

**reshape2**

Extensions (installation, mise à jour) . . . . .	55
--------------------------------------------------	----

**revealjs**

R Markdown : les rapports automatisés . . . . .	979
-------------------------------------------------	-----

**riverplot**

Autres extensions graphiques . . . . .	897
----------------------------------------	-----

**rmarkdown**

R Markdown : les rapports automatisés . . . . .	979
-------------------------------------------------	-----

**rmformats**

R Markdown : les rapports automatisés . . . . .	979
-------------------------------------------------	-----

**rmdshower**

R Markdown : les rapports automatisés . . . . .	979
-------------------------------------------------	-----

**RSQLite**

Import de données . . . . .	145
-----------------------------	-----

**rvest**

Scraping . . . . .	317
--------------------	-----

**S****sankeyNetwork**

Autres extensions graphiques . . . . .	897
----------------------------------------	-----

**scales**

Étendre ggplot2 . . . . .	793
Exemples de graphiques avancés . . . . .	839
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Mettre en forme des nombres . . . . .	1019
Trajectoires de soins: un exemple de données longitudinales . . . . .	705

**sf**

Analyse spatiale . . . . .	787
Cartes . . . . .	895

**sjPlot**

R Markdown : les rapports automatisés . . . . .	979
-------------------------------------------------	-----

**skimr**

Visualiser ses données . . . . .	175
----------------------------------	-----

**sp**

Analyse spatiale . . . . .	787
Cartes . . . . .	895
Import de données . . . . .	145

**srvyr**

Définir un plan d'échantillonnage complexe . . . . .	501
Données pondérées . . . . .	465
Manipuler les données avec dplyr . . . . .	233

**stats**

Analyse de séquences . . . . .	679
Analyse de survie . . . . .	651
Analyse des correspondances multiples (ACM) . . . . .	557
Classification ascendante hiérarchique (CAH) . . . . .	595
Comparaisons (moyennes et proportions) . . . . .	485
Données pondérées . . . . .	465
Effets d'interaction dans un modèle . . . . .	617
Formules . . . . .	929
Graphiques univariés et bivariés avec ggplot2 . . . . .	417
Intervalles de confiance . . . . .	477
lattice : graphiques et formules . . . . .	879
Manipulations avancées avec data.table . . . . .	251
Premier contact . . . . .	13
Premier travail avec des données . . . . .	33
Recodage de variables . . . . .	197
Régression logistique binaire, multinomiale et ordinale . . . . .	507
Statistique bivariée . . . . .	365
Statistique univariée . . . . .	339

**stringi**

Manipuler du texte avec stringr . . . . .	289
-------------------------------------------	-----

**stringr**

Analyse de séquences . . . . .	679
Expressions régulières . . . . .	977
Extensions (installation, mise à jour) . . . . .	55
Introduction au tidyverse . . . . .	59
Manipuler du texte avec stringr . . . . .	289
Scraping . . . . .	317
Trajectoires de soins: un exemple de données longitudinales . . . . .	705

**survey**

Analyse de survie . . . . .	651
Comparaisons (moyennes et proportions) . . . . .	485
Définir un plan d'échantillonnage complexe . . . . .	501
Données pondérées . . . . .	465
Intervalles de confiance . . . . .	477
Manipuler les données avec dplyr . . . . .	233
Régression logistique binaire, multinomiale et ordinale . . . . .	507

**survival**

Analyse de survie . . . . .	651
Trajectoires de soins: un exemple de données longitudinales . . . . .	705

**survminer**

Analyse de survie . . . . .	651
-----------------------------	-----

**svrepmiss**

Régression logistique binaire, multinomiale et ordinale . . . . .	507
-------------------------------------------------------------------	-----

**T****tables**

R Markdown : les rapports automatisés . . . . .	979
-------------------------------------------------	-----

**tibble**

Extensions (installation, mise à jour) . . . . .	55
Introduction au tidyverse . . . . .	59
Listes et Tableaux de données . . . . .	87
Manipulations avancées avec data.table . . . . .	251
Manipuler les données avec dplyr . . . . .	233
Visualiser ses données . . . . .	175

**tidyrr**

Analyse de séquences . . . . .	679
Exemples de graphiques avancés . . . . .	839
Extensions (installation, mise à jour) . . . . .	55
Introduction au tidyverse . . . . .	59
Manipuler du texte avec stringr . . . . .	289
Réorganiser ses données avec tidyrr . . . . .	299
Trajectoires de soins: un exemple de données longitudinales . . . . .	705

**tidyselect**

Réorganiser ses données avec tidyrr . . . . .	299
-----------------------------------------------	-----

**tidyverse**

Autres extensions graphiques . . . . .	897
Exemples de graphiques avancés . . . . .	839
Extensions (installation, mise à jour) . . . . .	55
Fusion de tables . . . . .	269
Introduction au tidyverse . . . . .	59
Manipuler du texte avec stringr . . . . .	289
Manipuler les données avec dplyr . . . . .	233
Recodage de variables . . . . .	197
Réorganiser ses données avec tidyrr . . . . .	299
Trajectoires de soins: un exemple de données longitudinales . . . . .	705

**tmap**

Cartes . . . . .	895
------------------	-----

**TraMineR**

Analyse de séquences . . . . .	679
Classification ascendante hiérarchique (CAH) . . . . .	595
Trajettoires de soins: un exemple de données longitudinales . . . . .	705

**TraMineRextras**

Analyse de séquences . . . . .	679
--------------------------------	-----

**U****utils**

Conditions et comparaisons . . . . .	925
Export de données . . . . .	327
Extensions (installation, mise à jour) . . . . .	55
Formules . . . . .	929
Import de données . . . . .	145
Installation de R et RStudio . . . . .	11
Introduction à ggplot2, la grammaire des graphiques . . . . .	395
Listes et Tableaux de données . . . . .	87
Où trouver de l'aide? . . . . .	163
Premier contact . . . . .	13
Premier travail avec des données . . . . .	33
Statistique univariée . . . . .	339
Structures conditionnelles . . . . .	945
Visualiser ses données . . . . .	175

**V****vcd**

Statistique bivariée . . . . .	365
--------------------------------	-----

**VGAM**

Régression logistique binaire, multinomiale et ordinale . . . . . 507

**viridis**

Autres extensions graphiques . . . . . 897  
 Couleurs et Palettes . . . . . 1021  
 Trajectoires de soins: un exemple de données longitudinales . . . . . 705

**W****waffle**

Étendre ggplot2 . . . . . 793

**WeightedCluster**

Analyse de séquences . . . . . 679  
 Classification ascendante hiérarchique (CAH) . . . . . 595  
 Trajectoires de soins: un exemple de données longitudinales . . . . . 705

**X****xlsx**

Export de données . . . . . 327  
 Import de données . . . . . 145

**xtable**

R Markdown: les rapports automatisés . . . . . 979

**Y****yarr**

Statistique bivariée . . . . . 365

**yarrr**

Statistique bivariée . . . . . 365