STATISTIQUES DESCRIPTIVES avec R

Christophe Ambroise

2025-03-21

Table of contents

1 Le langage R de base (R base)

1.1 Qu'est-ce que R ?

En bref

R est un logiciel de développement scientifique spécialisé dans le calcul et l'**analyse statistique**

Mais aussi

- un langage interprété,
- un environnement de développement,
- un projet open source (projet **GNU**),
- un logiciel multi-plateforme (Linux, Mac, Windows),
- la 18ième lettre de l'alphabet

1.2 Principales fonctionnalités

1. Gestionnaire de données - Lecture, manipulation, stockage.
2. Algèbre linéaire - Opérations classiques sur vecteurs, tableaux et matrices
3. Statistiques et analyse de données - Dispose d'un *grand* nombre de méthodes d'analyse de données (des plus anciennes et aux plus récentes)
4. Moteur de sorties graphiques - Sorties écran ou fichier
5. Système de modules - Alimenté par la communauté (+ de 2000 extensions!)
6. Interface facile avec C/C++, Fortran

2 Historique de R

2.1 Chronologie

Année	Événement
1976	Développement du langage **S** au **Bell Labs** par John Chambers et son équipe.

Année	Événement
1988	Développement de **S+**, version commerciale de S, par **Statistical Sciences Inc.** (plus tard AT&T).
1993	Création de **R**, inspiré de S, par **Robert Gentleman** et **Ross Ihaka** à l'Université d'Auckland.
1995	Publication des **codes sources de R** sous licence **GNU/GPL**.
1997	Élargissement du groupe de développement avec la participation de **Martin Mächler, Douglas Bates et Luke Tierney**.
2000	Sortie de **R 1.0.0**, première version stable officielle.
2002	Création de la **R Foundation for Statistical Computing**
2010	Lancement de **RStudio**, un environnement de développement intégré (IDE) dédié à R.
2015	Introduction du **tidyverse**, une collection de packages modernisant la manipulation de données et la visualisation.

2.2 Se tenir informé

1. **La page web de la Fondation R**
   - Statuts, liens et références.
   - https://www.r-project.org/foundation/
2. **La page web du CRAN (Comprehensive R Archive Network)**
   - Binaires d'installation, packages, documentations, etc.
   - https://cran.r-project.org/
3. **Conférences des utilisateurs de R**
   - **useR! 2025** : La conférence internationale des utilisateurs
     - https://user2025.r-project.org/
   - **Rencontres R 2025** : Les 11èmes Rencontres R auront lieu du 19 au 21 mai 2025 à l'Université de Mons (UMONS) en Belgique.
     - https://rr2025.sciencesconf.org/
4. **The R Journal**
   - Propose des articles sur de nouvelles extensions, des applications.
   - https://journal.r-project.org/

2.3 Qualités et défauts de R

Avantages

- **Libre et gratuit** : accessible à tous sans coût.
- **Écosystème riche** : des milliers de packages pour la statistique, le machine learning et la visualisation.
- **Large communauté** : support actif et nombreuses ressources en ligne.
- **Langage polyvalent** : adapté à l'analyse de données, la bioinformatique et la finance.
- **Puissant pour les statistiques** : leader dans les méthodes avancées d'analyse.
- **Visualisation avancée** : graphiques élégants (ggplot2…).
- **Reproductibilité** : grâce à R Markdown, Quarto et knitr.
- **Intégration facile** : avec Python, SQL, Spark, et d'autres langages.

2.4 Qualités et défauts de R

Inconvénients

- **Performance** : peut être lent sur des très grands volumes de données.
- **Gestion de la mémoire** : moins optimisée que d'autres langages comme Julia ou Rust.
- **Courbe d'apprentissage** : syntaxe parfois déroutante pour les débutants.
- **Support de l'IA et du deep learning** : moins développé que Python avec TensorFlow ou PyTorch.
- **Personnalisation graphique** : ggplot2 est puissant mais nécessite des ajustements parfois lourds.

2.5 Caractéristiques

R est un langage fonctionnel interprété (tout comme Python)

Rappel

Compilation classique:

- Le code source est entièrement traduit en code machine avant exécution.

Interprétation:

- Chaque ligne ou instruction est analysée, vérifiée, puis immédiatement exécutée par l'interpréteur sans génération préalable d'un fichier exécutable.

2.6 Compilation

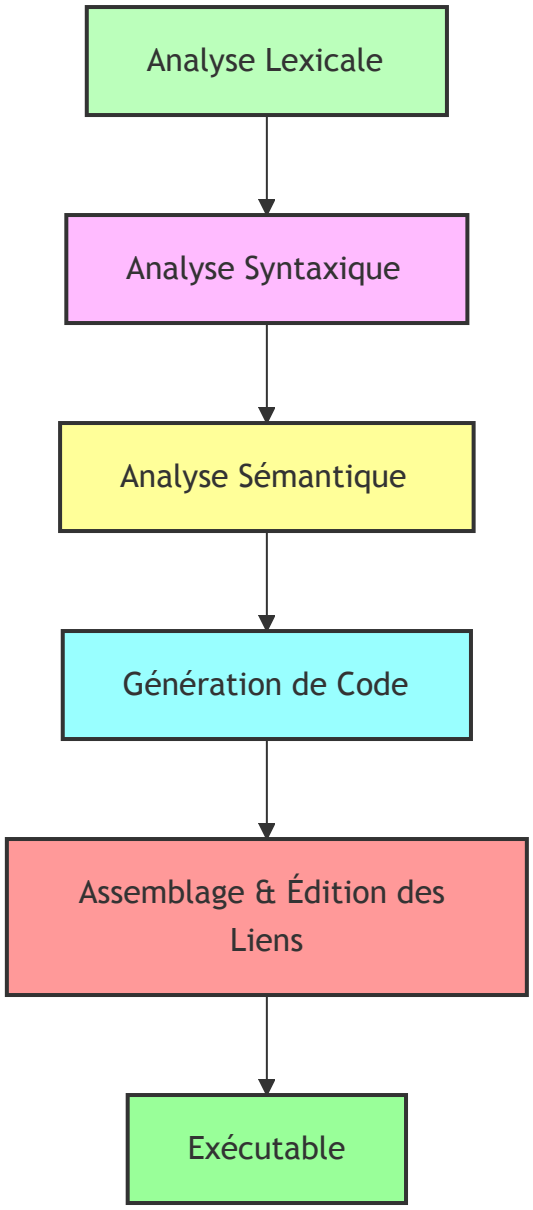


Figure 1: Compilation

2.7 Concurrents directs et alternatives

Domaine	Logiciels concurrents
Algèbre linéaire	Matlab (Mathworks), Scilab (INRIA), Octave (GNU)
Statistiques	SAS (SAS Inc.), SPSS (IBM), Stata
Calcul symbolique	Mathematica (Wolfram), Maple (Maplesoft), Maxima (GNU)
Autres alternatives	Python (le plus sérieux concurrent), Julia (rapide et moderne)

R reste un outil incontournable en **science des données** et **statistiques**, bien que Python et Julia gagnent en popularité pour le machine learning et le calcul haute performance.

2.8 Pléthore de livres sur R

Ouvrages de référence

- **Anciens ouvrages de référence** : Liste complète
- **R base** : Un livre de base pour débuter en R - Lire en PDF
- **Introduction à R** (par l'équipe du CRAN) - Lire en ligne
- **The R Book** (Michael J. Crawley) - Détails

Data Science et visualisation

- **ggplot2: Elegant Graphics for Data Analysis** (Hadley Wickham) - Lire sur Bookdown
- **R for Data Science** (Hadley Wickham & Garrett Grolemund) - Lire en ligne
- **Tidyverse Skills** (Hadley Wickham et al.) - Explorer sur Bookdown

2.9 Pléthore de livres sur R

R Markdown, Quarto et reporting

- **R Markdown: The Definitive Guide** (Yihui Xie, J.J. Allaire, Garrett Grolemund) - Lire sur Bookdown
- **Quarto for R Users** - Documentation officielle
- **Dynamic Documents with R and knitr** (Yihui Xie) - Détails

Statistiques et Machine Learning

- **Modern Applied Statistics with S** (Venables & Ripley) - Détails
- **An Introduction to Statistical Learning** (James, Witten, Hastie, Tibshirani) - Lire gratuitement
- **Elements of Statistical Learning** (Hastie, Tibshirani, Friedman) - Accès libre
- **Hands-On Machine Learning with R** (Brad Boehmke & Brandon Greenwell) - Détails

3 RStudio : une interface puissante pour R

3.1 Fonctionnalités principales

RStudio est un **environnement de développement intégré (IDE)** pour R, offrant :

- Un **éditeur de script intégré** avec coloration syntaxique.
- Une compatibilité **multi-plateforme** (Windows, macOS, Linux, serveurs).
- Une **gestion de projet et versionning** facilitée (intégration avec Git).
- Un accès direct aux **packages, graphiques et documentation**.

3.2 Organisation des panneaux dans RStudio

Lorsque vous ouvrez RStudio, l'interface est divisée en **quatre panneaux** :

Panneau	Fonction
Console (en bas à gauche)	Exécution des commandes R en direct.
Environnement / Historique (en haut à droite)	Liste des variables et historique des commandes.
Fichiers / Graphiques / Packages / Aide (en bas à droite)	Gestion des fichiers, affichage des graphiques, installation de packages et accès à la documentation.
Éditeur de script (en haut à gauche, une fois un fichier ouvert)	Écriture et exécution de scripts R.

3.3 Interagir avec R dans RStudio

Il existe **deux principales façons** d'interagir avec R :

1. **Via la console** : idéale pour tester des commandes rapidement.
2. **Via des fichiers de script** (`.R`) : recommandé pour un travail structuré et reproductible.

3.4 La console R (panneau en bas à gauche)

- C'est l'endroit où **R exécute directement les commandes**.
- Les résultats s'affichent immédiatement après chaque commande.
- **Attention** : les commandes tapées ici sont perdues après la fermeture de la session.

3.5 Éditeur de script (panneau en haut à gauche)

- Permet d'**écrire du code R et de le sauvegarder**.
- Exécuter du code **sans le retaper** à chaque session.
- Envoyer une ligne de code à la console avec **`Ctrl + Entrée`** (Windows/Linux) ou **`Cmd + Entrée`** (Mac).

3.6 Raccourcis utiles

Action	Raccourci
Passer de la console à l'éditeur	`Ctrl + 1`
Passer de l'éditeur à la console	`Ctrl + 2`
Exécuter une ligne de code	`Ctrl + Entrée` (Windows/Linux) ou `Cmd + Entrée` (Mac)

3.7 Commentaires

- Utilisez # signes pour commenter.
- Commentez libéralement dans vos scripts R.
- Tout ce qui se trouve à droite d'un # est ignoré par R.

3.8 Astuce: Exécution de segments de votre code

- RStudio vous offre une grande souplesse dans l'exécution du code depuis l'éditeur fenêtre. Il existe des boutons, des choix de menus et des raccourcis clavier. Pour exécuter la ligne en cours, vous pouvez

1. cliquez sur le bouton `Run` situé au-dessus du panneau de l'éditeur,
2. sélectionnez "Run Lines" dans le menu "Code", ou
3. appuyez sur Ctrl-Entrée dans Windows ou Linux ou sur Commande-Entrée sur OS X. (Ce raccourci peut également être vu en survolant la souris sur le bouton).

3.9 Bloc de code

- Pour lancer un bloc de code, sélectionnez-le, puis Exécuter.
- Si vous avez modifié une ligne de code dans un bloc de code que vous venez d'exécuter, il n'est pas nécessaire de resélectionner la section et `Run`, vous pouvez utiliser le bouton suivant,
- Ré-exécuter la région précédente Cela exécutera le bloc de code précédent inculquer les modifications que vous avez apportées.

3.10 Utiliser R comme calculatrice

La chose la plus simple que vous puissiez faire avec R est l'arithmétique:

```
1 + 100
```

```
[1] 101
```

Et R imprimera la réponse, avec un précédent "[1]". Ne t'inquiète pas pour ça pour l'instant, nous l'expliquerons plus tard. Pour l'instant, considérez-le comme indiquant une sortie.

3.11 Commande incomplète

Comme bash, si vous tapez une commande incomplète, R vous attendra pour complèter l'entrée:

```
> 1 +
```

```
+
```

Chaque fois que vous appuyez sur Entrée et que la session R affiche un "+" au lieu d'un ">", signifie qu'il attend que vous complétiez la commande. Si vous voulez annuler une commande que vous pouvez simplement appuyer sur "Esc" et RStudio vous rendra le ">" rapide.

3.12 Astuce: Annulation des commandes

- Si vous utilisez R depuis la ligne de commande plutôt que depuis RStudio,
- vous devez utiliser `Ctrl + C` au lieu de`Esc` pour annuler la commande. Ce s'applique également aux utilisateurs de Mac!
- Annuler une commande n'est pas seulement utile pour tuer des commandes incomplètes:
- vous pouvez aussi l'utiliser pour dire à R d'arrêter d'exécuter du code (par exemple si
- en prenant beaucoup plus de temps que prévu, ou pour se débarrasser du code que vous êtes en train d'écrire.

3.13 Ordre des opérations

Lorsque vous utilisez R comme calculatrice, l'ordre des opérations est le même que vous aurait appris à l'école.

De la plus haute à la plus basse préséance:

- Parenthèses: `(`, `)`
- Exposants: `^` ou `**`
- Diviser: `/`
- Multiplier: `*`
- Ajouter: `+`
- Soustraire: `-`

```
3 + 5 * 2
```

```
[1] 13
```

Utilisez des parenthèses pour regrouper les opérations afin de forcer l'ordre de évaluation si elle diffère du défaut, ou pour préciser ce que vous avoir l'intention

```
(3 + 5) * 2
```

```
[1] 16
```

Cela peut devenir compliqué lorsque cela n'est pas nécessaire, mais clarifie vos intentions. Rappelez-vous que d'autres peuvent lire votre code ultérieurement.

```
(3 + (5 * (2 ^ 2))) # difficile à lire
3 + 5 * 2 ^ 2 # si vous vous souvenez des règles
3 + 5 * (2 ^ 2) # Si vous oubliez certaines règles, cela pourrait vous aider
```

Le texte après chaque ligne de code s'appelle un "commentaire". Tout ce qui suit le symbole de hachage (ou octothorpe) "#" est ignoré par R lorsqu'il exécute du code.

Les nombres vraiment petits ou grands ont une notation scientifique:

```
2/10000
```

```
[1] 2e-04
```

Ce qui est un raccourci pour “multiplié par`10 ^ XX`”. Donc `2e-4` est un raccourci pour `2 * 10 ^ (-4)`.

Vous pouvez aussi écrire des nombres en notation scientifique:

```
5e3 # Notez l'absence de moins ici
```

```
[1] 5000
```

3.14 Fonctions mathématiques

R a beaucoup de fonctions mathématiques intégrées. Pour appeler une fonction, nous tapons simplement son nom, suivi par des parenthèses ouvertes et fermantes. Tout ce que nous tapons à l’intérieur des parenthèses s’appelle la fonction arguments:

```
sin (1) # fonctions trigonométriques
```

```
[1] 0.841471
```

```
log (1) # logarithme naturel
```

```
[1] 0
```

```
log10 (10) # base-10 logarithme
```

```
[1] 1
```

```
exp (0.5) # e ^ (1/2)
```

```
[1] 1.648721
```

3.15 Se souvenir

Ne vous souciez pas d'essayer de vous souvenir de toutes les fonctions de R:

- rechercher sur Google,
- ou si vous vous souvenez de la début du nom de la fonction, utilisez *complétion* dans RStudio.

C'est un avantage que RStudio sur R , il dispose de *capacités d'auto-complétion* qui vous permettent plus facilement rechercher des fonctions, leurs arguments et les valeurs qu'ils prendre.

Taper un `?` Avant le nom d'une commande ouvrira la page d'aide pour cette commande. En plus de fournir

- une description la page d'aide affichera généralement
- une collection d'exemples

3.16 Comparer les choses

Nous pouvons également faire la comparaison en R:

```
1 == 1 # égalité (noter deux signes égaux, lire comme "est égal à")
```

```
[1] TRUE
```

```
1 != 2 # inégalité (lire comme "n'est pas égal à")
```

```
[1] TRUE
```

```
1 <2 # moins que
```

```
[1] TRUE
```

```
1 <= 1 # inférieur ou égal à
```

```
[1] TRUE
```

```
1> 0 # plus grand que
```

```
[1] TRUE
```

```
1 >= -9 # supérieur ou égal à
```

```
[1] TRUE
```

3.17 Astuce: Comparer les nombres

- Un mot d'avertissement sur la comparaison des chiffres: vous devriez ne jamais utiliser == pour comparer deux nombres à moins qu'ils ne soient entiers (un type de données pouvant représenter spécifiquement uniquement des nombres entiers).
- Les ordinateurs ne peuvent représenter que des nombres décimaux avec un un certain degré de précision, donc deux nombres qui semblent le même lorsqu'il est imprimé par R, peut effectivement avoir différentes représentations sous-jacentes et donc être différent par une petite marge d'erreur (appelée Machine tolérance numérique).
- Au lieu de cela, vous devez utiliser la fonction `all.equal`.
- Lectures complémentaires: http://floating-point-gui.de/

3.18 Variables et affectation

Stocker des valeurs dans des variables en utilisant l'opérateur d'affectation <-

```
x <- 1/40
```

```
x
```

```
[1] 0.025
```

Plus précisément, la valeur stockée est une approximation * décimale * de cette fraction appelée [nombre à virgule flottante] (http://en.wikipedia.org/wiki/Floating_point).

Recherchez l'onglet `Environment` dans l'un des volets de RStudio, et vous verrez que `x` et sa valeur est apparu. Notre variable `x` peut être utilisée à la place d'un nombre dans tout calcul qui attend un nombre:

```
log (x)
```

```
[1] -3.688879
```

Notez également que les variables peuvent être réaffectées:

```r
x <- 100
```

`x` contenait la valeur 0.025 et a maintenant la valeur 100.

Les valeurs d'affectation peuvent contenir la variable affectée à:

```r
x <- x + 1 #notice comment RStudio met à jour sa description de x en haut à droite
```

Le côté droit de l'affectation peut être toute expression R valide. Le côté droit est *entièrement évalué* avant que l'affectation ait lieu.

3.19 Noms de variables

Les noms de variables peuvent contenir des lettres, des chiffres, des traits de soulignement et des points. Ils ne peut pas commencer avec un nombre ni contenir des espaces du tout. Différentes personnes utilisent différentes conventions pour les noms de variables longues, celles-ci comprennent

- periods.between.words
- souligne_entre_mots
- camelCaseToSeparateWords

Ce que vous utilisez dépend de vous, mais **soyez cohérent**.

3.20 Affectation (suite)

Il est également possible d'utiliser l'opérateur = pour l'affectation:

```r
x = 1/40
```

Mais c'est beaucoup moins courant parmi les utilisateurs de R.

Donc, la recommandation est d'utiliser `<-`.

3.21 Gérer votre environnement

Il existe quelques commandes utiles que vous pouvez utiliser pour interagir avec la session R.

`ls` listera toutes les variables et fonctions stockées dans l'environnement global (votre session de travail):

```
ls()
```

```
[1] "x"
```

3.22 Astuce: objets cachés

- Comme dans le shell, `ls` cachera toutes les variables ou fonctions commençant avec un "." par défaut.
- Pour lister tous les objets, tapez `ls (all.names = TRUE)`

Remarque

Notez ici que nous n'avons donné aucun argument à `ls`, mais nous avons quand même nécessaire de donner aux parenthèses de dire à R d'appeler la fonction.

3.23 Contenu d'une fonction

Si vous tapez `ls` par lui-même, R imprimera le code source de cette fonction!

```
ls
```

```
function (name, pos = -1L, envir = as.environment(pos), all.names = FALSE,
    pattern, sorted = TRUE)
{
    if (!missing(name)) {
        pos <- tryCatch(name, error = function(e) e)
        if (inherits(pos, "error")) {
            name <- substitute(name)
            if (!is.character(name))
                name <- deparse(name)
            warning(gettextf("%s converted to character string",
                sQuote(name)), domain = NA)
            pos <- name
        }
    }
    all.names <- .Internal(ls(envir, all.names, sorted))
    if (!missing(pattern)) {
        if ((ll <- length(grep("[", pattern, fixed = TRUE))) &&
            ll != length(grep("]", pattern, fixed = TRUE))) {
```

```
            if (pattern == "[") {
                pattern <- "\\["
                warning("replaced regular expression pattern '[' by  '\\\\['")
            }
            else if (length(grep("[^\\\\]\\[<-", pattern))) {
                pattern <- sub("\\[<-", "\\\\\\[<-", pattern)
                warning("replaced '[<-' by '\\\\[<-' in regular expression pattern")
            }
        }
        grep(pattern, all.names, value = TRUE)
    }
    else all.names
}
<bytecode: 0x140f212a0>
<environment: namespace:base>
```

3.24 Supression

Vous pouvez utiliser `rm` pour supprimer des objets dont vous n'avez plus besoin:

```
rm (x)
```

Si vous avez beaucoup de choses dans votre environnement et souhaitez les supprimer toutes, vous pouvez transmettre les résultats de `ls` à la fonction `rm`:

```
rm(list=ls())
```

Dans ce cas, nous avons spécifié que les résultats de `ls` devraient être utilisés pour le L'argument `list` dans `rm`. Lorsque vous attribuez des valeurs aux arguments par nom, vous *devez* utilisez l'opérateur = !!

Si au lieu de cela nous utilisons `<-`, il y aura des effets secondaires inattendus, ou vous pourriez avoir un message d'erreur:

```
rm(list <- ls ())
```

```
Error in rm(list <- ls()): ... must contain names or character strings
```

3.25 Astuce: Avertissements vs erreurs

- Quand R fait quelque chose d'inattendu! Les erreurs, comme ci-dessus, sont émises lorsque R ne peut pas procéder à un calcul.
- En revanche, les avertissements signifient généralement que la fonction a été exécutée, mais cela n'a probablement pas fonctionné comme prévu.
- Dans les deux cas, le message que R imprime vous donne généralement des indices sur la manière de résoudre un problème.

3.26 R Packages

Il est possible d'ajouter des fonctions à R en écrivant un paquet, ou par obtenir un paquet écrit par quelqu'un d'autre. Au moment de l'écriture, il y a sont plus de 7 000 paquets disponibles sur CRAN (l'archive complète de R réseau). R et RStudio ont des fonctionnalités pour gérer les paquets:

- Vous pouvez voir quels paquets sont installés en tapant

```
installed.packages ()
```

- Vous pouvez installer des paquets en tapant

`install.packages (" packagename ")`, où `packagename` est le nom du package, entre guillemets.

- Vous pouvez mettre à jour les paquets installés en tapant `update.packages ()`
- Vous pouvez supprimer un paquet avec `remove.packages (" packagename ")`
- Vous pouvez rendre un paquet disponible pour être utilisé avec `library (packagename)`

3.27 Exercices

1. Nommer toutes les panels de R studio
2. Réaliser une addition et une multiplication dans la console
3. Sauver un script R avec votre code d'addition

4 Chercher de l'aide

4.1 Lire les fichiers d'aide

R, et chaque paquet, fournissent des fichiers d'aide pour les fonctions. Pour chercher de l'aide sur un fonction d'une fonction spécifique qui est dans un package chargé

```
?nom_fonction
help(nom_fonction)
```

Cela chargera une page d'aide dans RStudio (ou du texte brut dans R seul).

4.2 Opérateurs spéciaux

Pour demander de l'aide sur des opérateurs spéciaux, utilisez des guillemets:

```
?"+"
```

4.3 Obtenir de l'aide sur les packages

De nombreux paquets contiennent des "vignettes": des tutoriels et des exemples de documentation. Sans aucun argument, `vignette()` listera toutes les vignettes pour tous les paquets installés;

`vignette (package =" nom-du-paquet ")`

listera toutes les vignettes disponibles pour

`package-name` et `vignette ("vignette-name")`

ouvriront la vignette spécifiée.

Si un paquet ne contient aucune vignette, vous pouvez généralement trouver de l'aide en tapant

`help("nom-du-paquet")`.

4.4 Quand vous vous souvenez de la fonction

Si vous ne savez pas exactement dans quel paquet est une fonction ou comment elle est spécifiquement orthographiée, vous pouvez effectuer une recherche floue:

```
??nom_fonction
```

5 Mélanger code et texte avec `knitr`

5.1 Motivations: reproductibilité de la recherche

Principe de Claerbout (Géophysicien, Stanford)

> An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.

5.2 En bref

`knitr`

pour générer des rapports qui combinent le texte, le code et les résultats.

Markdown

pour mettre en forme le texte

> Markdown est un langage de balisage léger créé par John Gruber en 2004. Son but est d'offrir une syntaxe facile à lire et à écrire. Un document balisé par Markdown peut être lu en l'état sans donner l'impression d'avoir été balisé ou formaté par des instructions particulières. — Wikipedia

Lien Wikipedia

Code Chunks

code dans des blocs délimités par des guillemets triples suivis de `{r}`.

5.3 Référence

Rmardown est un univers extensible, si vous voulez continuez, lisez

https://rmarkdown.rstudio.com/

5.4 Installer `knitr`

Dans la console

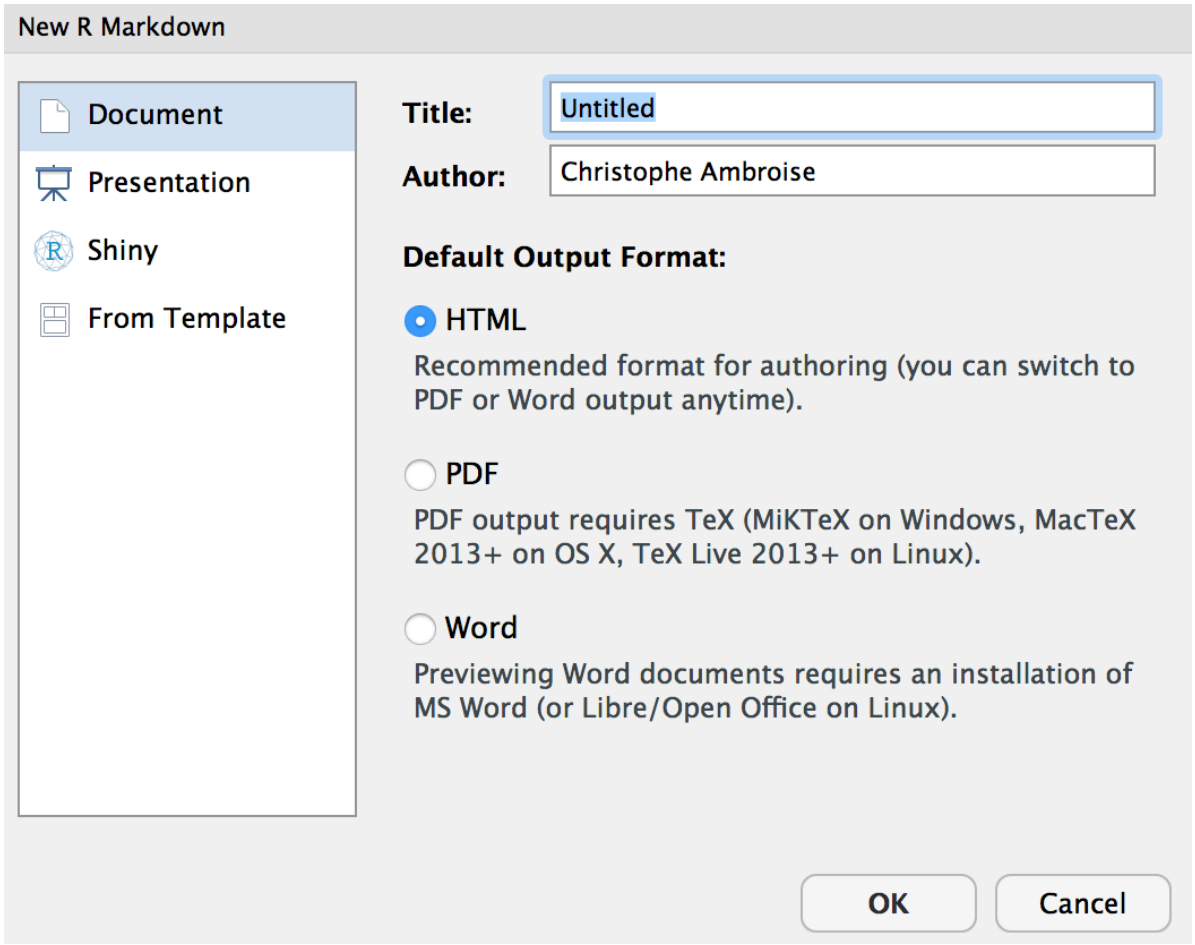
```
install.packages('knitr')
```

Par le menu

Tools -> Install Packages

5.5 Créer un de type fichier Rmarkdown (.Rmd)

Dans R Studio, cliquez sur Fichier → Nouveau fichier → R Markdown et vous obtiendrez une boîte de dialogue du type



5.6 YAML

Vous créez un fichier avec une entête dite yaml du type

```
---
title: "Initial R Markdown document"
author: "Karl Broman"
date: "April 23, 2015"
output: html_document
---
```

qui précise comment peut être transformé le fichier

5.7 Markdown

Markdown est un langage à balise

- écrivez **en gras** en utilisant deux astérisques, comme ceci: `**gras**`,
- écrivez en *italics* en utilisant des traits de soulignement, comme ceci: `_italics_`.

Vous pouvez créer une liste à puces en écrivant une liste avec des tirets ou astérisques, comme ceci:

```
* gras avec double astérisque
* italiques avec des soulignés
* police de type code avec backticks
```

5.8 Markdown

ou comme ça:

```
- gras avec double astérisque
- italiques avec des soulignés
- police de type code avec backticks
```

Vous pouvez créer une liste numérotée en utilisant simplement des chiffres. Vous pouvez utiliser le même nombre encore et encore si vous voulez:

```
1. gras avec double astérisque
1. italiques avec des soulignés
1. police de type code avec backticks
```

Cela apparaîtra comme:

1. gras avec double astérisque
2. italiques avec des soulignés
3. police de type code avec backticks

5.9 Markdown

Vous pouvez créer des en-têtes de section de différentes tailles en initiant une ligne avec un certain nombre de symboles `#`:

```
# Titre
## Section principale
### Sous-section
#### Sous-sous-section
```

5.10 Compilation

Vous *compilez* le document R Markdown en cliquant sur le “Knit HTML” en haut à gauche.

5.11 Un peu plus de Markdown

- Vous pouvez créer un hyperlien comme celui-ci:

```
[texte à afficher] (http://the-web-page.com).
```

- Vous pouvez inclure un fichier image comme ceci:

```
![Caption](http://url/for/file)
```

Vous pouvez faire des indices (par exemple, F_2) avec `F~2~` et des exposants (par exemple, F^2) avec `F^2^`.

5.12 Un peu plus de Markdown

Si vous savez écrire des équations dans [LaTeX] (http://www.latex-project.org/), vous serez heureux de savoir que vous pouvez utiliser `$ $` et `$$ $$` pour insérer des équations mathématiques, comme `$E = mc^2$` $E = mc^2$ et

```
$$y = \mu + \sum_ {i = 1}^p \beta_i x_i + \epsilon$$
```

$$y = \mu + \sum_{i=1}^{p} \beta_i x_i + \epsilon$$

5.13 Morceaux de code

Markdown est intéressant et utile, mais le plus grand intérêt vient du mélange du texte balisé avec des morceaux de code R.

- Quand traité, le code R sera exécuté; s’ils produisent des valeurs, figures, ceux-ci seront insérés dans le document final.

Les morceaux de code ressemblent à ceci:

````
```{r load_data}
gapminder <- read.csv("~/Desktop/gapminder.csv")
```
````

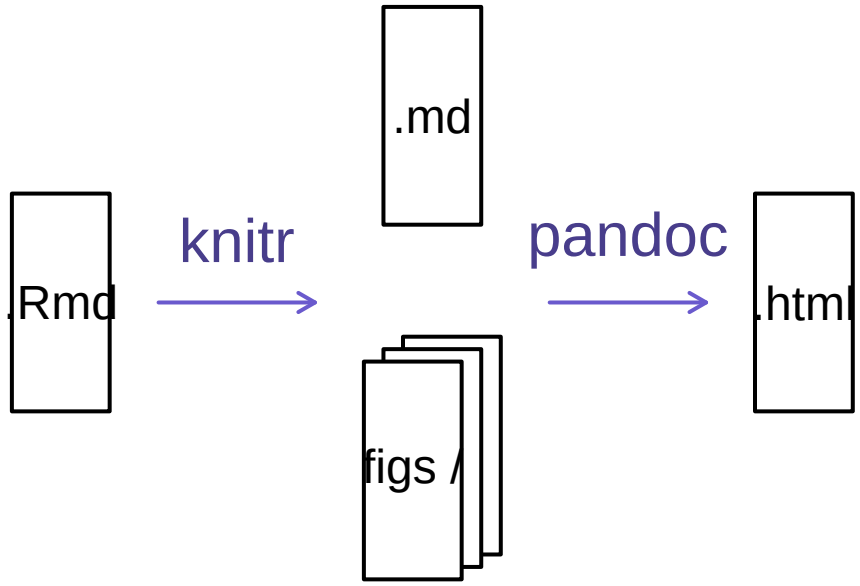
C'est une bonne idée de donner chaque morceau un nom, car ils vous aideront à corriger les erreurs et, si des graphiques sont produit, les noms de fichiers sont basés sur le nom du bloc de code les a produites.

5.14 Morceaux de code

Lorsque vous appuyez sur le bouton "Knit HTML", le document R Markdown est traité par [knitr] (http://yihui.name/knitr) et un simple Markdown document est produit (ainsi que, potentiellement, un ensemble de fichiers de figure): le code R est exécuté et remplacé à la fois par l'entrée et la sortie; si les chiffres sont produits, des liens vers ces chiffres sont inclus.

Les documents Markdown et figure sont ensuite traités par l'outil pandoc, qui convertit le fichier Markdown en fichier fichier html, avec les chiffres incorporés.

5.15 Knitr en diagramme



5.16 Options de blocs

Il y a une variété d'options pour affecter la façon dont les morceaux de code sont traités.

- Utilisez `echo = FALSE` pour éviter que le code lui-même ne soit affiché.
- Utilisez `results ="hide"` pour éviter d'imprimer des résultats.
- Utilisez `eval = FALSE` pour que le code soit affiché mais pas évalué.

- Utilisez `warning = FALSE` et `message = FALSE` pour masquer les avertissements ou messages produits.
- Utilisez `fig.height` et `fig.width` pour contrôler la taille des figures produit (en pouces).

5.17 Resources

Voici une sélection de ressources essentielles pour utiliser R Markdown et knitr :

- **Knitr in a Knutshell Tutorial** : Introduction rapide à `knitr` par Karl Broman - Lire ici
- **Dynamic Documents with R and knitr** (Yihui Xie) : Livre détaillé sur la génération de rapports dynamiques - Détails
- **R Markdown Documentation** : Guide officiel pour R Markdown - Consulter ici
- **R Markdown Cheat Sheet** : Fiche récapitulative des principales commandes - Télécharger
- **Quarto Documentation** : Guide officiel pour Quarto, l'évolution de R Markdown - Lire ici

Ces ressources vous aideront à **exploiter pleinement R Markdown et knitr** pour créer des documents dynamiques et reproductibles !

5.18 Exercice

1. Créer un fichier Rmarkdown qui produira une sortie html avec un code chunk de votre choix

5.19 Pour aller plus loin : Quarto vs Jupyter Notbooks

Knitr et Rmarkdown sont relativement spécifiques à R.

Quarto et les Jupyter Notebooks sont deux outils puissants utilisés pour la création de documents reproductibles mêlant code, texte et visualisations.

5.20 Jupyter Notebooks

- **Nature Interactive** : Interface web interactive permettant d'exécuter du code cellule par cellule, facilitant l'exploration et l'analyse de données en temps réel.
- **Format de Fichier** : Extension `.ipynb`, stockant le code, le texte et les sorties (graphiques) dans un fichier JSON.
- **Langages Supportés** : Principalement Python, mais supporte aussi R, Julia, et d'autres via des noyaux (kernels).
- **Cas d'Utilisation** : Idéal pour le prototypage rapide, l'analyse exploratoire de données et les démonstrations éducatives.

5.21 Quarto

- **Polyvalence** : Plateforme de publication scientifique et technique permettant de créer des articles, blogs, livres et présentations.
- **Indépendance du Langage** : Supporte nativement Python, R, Julia et JavaScript.
- **Contrôle de Version Amélioré** : Fichiers `.qmd` en texte simple, mieux adaptés au contrôle de version avec Git.
- **Rendu et Publication** : Utilise Pandoc pour exporter vers divers formats (HTML, PDF, Word). Exécute le code lors du rendu pour garantir des sorties à jour.

5.22 Différences Clés

- **Interactivité vs. Publication** : Jupyter favorise l'expérimentation en temps réel, tandis que Quarto est conçu pour créer des documents publiables.
- **Gestion du Code et des Sorties** : Jupyter intègre les sorties dans le fichier, rendant le contrôle de version difficile. Quarto exécute le code à chaque rendu sans stocker les sorties.
- **Flexibilité des Formats** : Quarto prend en charge de nombreux formats de sortie et propose des fonctionnalités avancées (références croisées, bibliographies, mises en page personnalisées).

6 Structures de données

6.1 Vecteurs: définition

Propriétés

- objet le plus *élémentaire* sous R,
- collection d'entités *de même nature*,
- (ou type) défini par la nature des entités qui le composent.

Les modes possibles

1. numérique
2. caractère
3. logique

6.2 Quelques vecteurs typés

1. Numérique

```
x0 <- 0
x1 <- c(-1,23,98.7)
mode(x0)
```

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

2. Caractère

```
y0 <- "bonjour"
y1 <- c("Pomme","Flore","Alexandre")
mode(y1)
```

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

3. Logique

```
z0 <- TRUE
z1 <- c(FALSE,TRUE,FALSE,TRUE,TRUE)
z2 <- c(T,F,F)
mode(z2)
```

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

6.3 Remarques sur l'affectation

Affectation

```
C'est l'opération qui consiste à *attribuer une valeur* à une variable.
```

En R, plusieurs choix sont possibles:

l'opérateur usuel est <- (signe inférieur suivi du signe moins)

```r
jo <- "l'indien"
jo
```

```
[1] "l'indien"
```

l'opérateur = peut être utilisé la plupart du temps

```r
nb.max.d.annees.pour.faire.une.these = 3
nb.max.d.annees.pour.faire.une.these
```

```
[1] 3
```

la commande `assign` permet cette opération (d'où l'anglicisme *assignation*)

```r
assign("x", c(8,9,-pi,sqrt(2)))
```

6.4 Valeurs spéciales

Variables réservées par R

- `NA` est le code R pour les valeurs manquantes (absentes des données),
- `NaN` est le code de R pour signifier un résultat numérique aberrant ,
- `Inf` et `-Inf` sont les valeurs réservées pour plus et moins ∞,
- `NULL` est l'objet nul.

```r
c(4,2,NA,5)
```

```
[1]  4  2 NA  5
```

```r
0/0
```

```
[1] NaN
```

```
1/0
```

```
[1] Inf
```

```
names(1)
```

```
NULL
```

6.5 Opérations arithmétiques

Les opérations sur les vecteurs s'effectuent terme-à-terme

Soient `x`,`y` tels que

```
x<-c(1,2,-3,-4)
y<-c(-5,-6,9,0)
```

+

addition des éléments de deux vecteurs

```
x+y
```

```
[1] -4 -4  6 -4
```

-

soustraction des éléments de deux vecteurs

```
x-y
```

```
[1]   6   8 -12  -4
```

*****

multiplication des éléments de deux vecteurs

```
x*y
```

```
[1]  -5 -12 -27   0
```

\

division des éléments de deux vecteurs

```
x/y
```

```
[1] -0.2000000 -0.3333333 -0.3333333       -Inf
```

6.6 Recyclage des éléments du vecteur

Lors d'une opération entre vecteurs, les vecteurs trop courts sont ajustés pour atteindre la taille du plus grand vecteur en recyclant les données.

⇝ souvent pratique mais **attention aux effets de bords!**

6.7 Opérateurs mathématiques

Fonctions numériques élémentaires

`floor,ceiling,round`

```
floor(2/3)
```

```
[1] 0
```

```
ceiling(2/3)
```

```
[1] 1
```

```
round(2/3,3)
```

```
[1] 0.667
```

6.8 Fonctions arithmétiques élémentaires

`` `\^{},\%\%,\%/\%,abs,log,exp,log10,sqrt,cos,tan,sin...` ``

s'appliquent toutes terme-à-terme.

```
log10(c(10,100,1000))
```

```
[1] 1 2 3
```

```
cos(c(pi/2,pi))^2 + sin(c(pi/2,pi))^2
```

```
[1] 1 1
```

6.9 Fonctions spécifiques à un vecteur

prod, sum, max, min, range, which.min, which.max, length

```
x <- c(-8,1.5,3)
prod(x)
```

```
[1] -36
```

```
sum(x)
```

```
[1] -3.5
```

```
length(x)
```

```
[1] 3
```

```
max(x)
```

```
[1] 3
```

```
min(x)
```

```
[1] -8
```

```
range(x)
```

```
[1] -8  3
```

```
which.max(x)
```

```
[1] 3
```

```
which.min(x)
```

```
[1] 1
```

Pour le minimum / maximum terme-à-terme: `pmin,pmax`.

6.10 Fonctions appliquées le long du vecteur

`` `cumsum, cumprod, cummin, cummax` ``

```
x <- c(-2,1,-3,2)
cumprod(x)
```

```
[1] -2 -2  6 12
```

```
cumsum(x)
```

```
[1] -2 -1 -4 -2
```

```
cummax(x)
```

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

```
cummin(x)
```

```
[1] -2 -2 -3 -3
```

6.11 Opérateurs ensemblistes

Fonctionnent pour tous les modes

`unique,intersect,union,setdiff,setequal,is.element`

```
unique(c("banane","citron","banane"))
```

```
[1] "banane" "citron"
```

```
intersect(c("banane","citron"),c("orange","banane"))
```

```
[1] "banane"
```

```
union(c("banane","citron"),c("orange","banane"))
```

```
[1] "banane" "citron" "orange"
```

```
setequal(c("banane","citron"),c("orange","banane"))
```

```
[1] FALSE
```

```
is.element(1,sample(c(1,2,3),2))
```

```
[1] TRUE
```

6.12 Génération de vecteurs

Il existe de nombreuse manière de générer des vecteur de manière plus ou moins automatique

R étant un langage vectoriel savoir générer le vecteur que l'on veut représente un atout important pour programmer en R

6.13 L'opérateur :

```
from:to

Génère une séquence par pas de un depuis le nombre `from`
jusqu'à `to` (si possible).
```

```
-5:5
```

```
 [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

```
5:-5
```

```
 [1]  5  4  3  2  1  0 -1 -2 -3 -4 -5
```

```
pi:6
```

```
[1] 3.141593 4.141593 5.141593
```

```
1:6/2
```

```
[1] 0.5 1.0 1.5 2.0 2.5 3.0
```

```
1:(6/2)
```

```
[1] 1 2 3
```

6.14 La commande seq

Plusieurs schémas possibles

- seq(from,to)
- seq(from,to,by=)
- seq(from,to,length.out=)

Exemple

```r
seq(1,10)
```

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

```r
seq(2,10,by=2)
```

```
[1]  2  4  6  8 10
```

```r
seq(2,10,length.out=6)
```

```
[1]  2.0  3.6  5.2  6.8  8.4 10.0
```

6.15 La commande `rep`

Fonctionne pour tous les modes

- `rep(x,times)`, où `times` peut être un vecteur,
- `rep(x,each)`.

Exemple

```r
rep(1,3)
```

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

```r
rep("Mercy",3)
```

```
[1] "Mercy" "Mercy" "Mercy"
```

```r
rep(c("A","B","C"),c(3,2,4))
```

```
[1] "A" "A" "A" "B" "B" "C" "C" "C" "C"
```

```r
rep(c(TRUE,FALSE), each=2)
```

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

6.16 Génération de vecteurs logiques

Obtenus par conditions avec

- les opérateurs logiques '<', '<=', '>', '>=', '==' '!='
- le ET, le OU, NON, OU exclusif: '\&' (intersection), '|' (union), '!' (négation), xor.

```r
note1 <- c(8,9,14,3,17.5,11)
note2 <- c("C","B","A","B","E","B")
admis <- (note1 >= 10) & (note2 == "A" | note2 == "B")
mention <- (note1 >= 15) & (note2 == "A")
admis
```

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

```r
sum(admis)
```

```
[1] 2
```

```r
sum(mention)
```

```
[1] 0
```

6.17 Par concaténation

Avec 'c()'

```
L'opérateur `'c()'` peut s'appliquer à n'importe quoi pourvu que
l'on concatène des vecteurs de même type.
```

```r
c( c(1,2), c(3,4))
```

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

```r
round(c(seq(-pi,pi,len=4),rep(c(1:3),each=2),0),2)
```

```
 [1] -3.14 -1.05  1.05  3.14  1.00  1.00  2.00  2.00  3.00  3.00  0.00
```

remarque

Dans le second exemple, les entiers composants `c(1:3)` ont été forcés au typage flottant.

Avec `paste`

```
Concaténation  de chaînes de  caractères. Convertit  en caractères
les éléments passés en argument avant toute opération.
```

```r
paste("R","c'est","bien")
```

```
[1] "R c'est bien"
```

```r
paste(2:4,"ieme")
```

```
[1] "2 ieme" "3 ieme" "4 ieme"
```

```r
paste("A",1:5, sep="")
```

```
[1] "A1" "A2" "A3" "A4" "A5"
```

```r
paste("A",1:5, sep="",collapse="")
```

```
[1] "A1A2A3A4A5"
```

6.18 Indexation des vecteurs

Principe

- Permet la **sélection d'un sous-ensemble** du vecteur `x`.
- Le sous-ensemble est spécifié **entre crochets** `x[subset]`.

L'objet `subset` peut prendre 4 types différents

1. **un vecteur logique**, qui doit être de la même taille que le vecteur `x`;
2. **un vecteur numérique aux composantes positives**, qui spécifie les valeurs à inclure;
3. **un vecteur numérique aux composantes négatives**, qui spécifie les valeurs à exclure;
4. **un vecteur de chaînes de caractères**, qui spécifie les noms des éléments de `x` à conserver.

6.19 Indexation des vecteurs: exemples

Vecteurs logiques

```
x <- c(3,6,-2,9,NA,sin(-pi/6))
x[x > 0]
```

```
[1]  3  6  9 NA
```

```
x[!is.na(x)]
```

```
[1]  3.0  6.0 -2.0  9.0 -0.5
```

```
x[!is.na(x) & x>0]
```

```
[1] 3 6 9
```

```
mean(x,na.rm=TRUE)
```

```
[1] 3.1
```

```
x[x <= mean(x,na.rm=TRUE)]
```

```
[1]  3.0 -2.0   NA -0.5
```

Vecteurs aux composantes positives ou négatives

```
x
```

```
[1]  3.0  6.0 -2.0  9.0   NA -0.5
```

```r
x[2]
```

```
[1] 6
```

```r
x[1:5]
```

```
[1]  3  6 -2  9 NA
```

```r
x[-c(1,5)]
```

```
[1]  6.0 -2.0  9.0 -0.5
```

```r
x[-(1:5)]
```

```
[1] -0.5
```

Vecteurs de chaînes de caractères

```r
names(x) <- c("var1","var2","var3","var4","var5","var6")
x
```

```
var1 var2 var3 var4 var5 var6
 3.0  6.0 -2.0  9.0   NA -0.5
```

```r
x[c("var1","var3")]
```

```
var1 var3
   3   -2
```

6.20 Autres commandes d'indexation et de sélection

1. **Classer**

- `sort` renvoie le vecteur classé par ordre croissant ou décroissant,
- `order` renvoie les indices d'ordre des éléments par ordre croissant ou décroissant,

2. **Extraire**

- `which` renvoie les indices de `x` vérifiant une condition;

3. **Échantillonner**

- `sample` échantillonne aléatoirement dans un vecteur `x`, avec ou sans remise.

6.21 Exemples

```
x <- -5:5
y <- sample(x)
sort(y)
```

```
 [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

```
order(y)
```

```
 [1]  3  1  2  9 10  6  7  5  8  4 11
```

```
y[order(y)]
```

```
 [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

```r
y[order(y,decreasing=TRUE)]
```

```
 [1]  5  4  3  2  1  0 -1 -2 -3 -4 -5
```

```r
which(sample(x,4) > 0)
```

```
[1] 1
```

6.22 Facteurs

Facteur

Un **facteur** est un vecteur de **variables catégorielles** .
Les **niveaux** du facteur peuvent être ordonnés ou pas.

Utilisation les facteurs s'utilisent pour

catégoriser les données d'un vecteur (ce qui s'avère très utile pour la gestion des variables qualitatives).

⇝ un facteur est souvent associé à d'autres vecteurs pour en définir une **partition**.

6.23 Création, manipulation

Création: la fonction `factor`

```r
factor(sample(1:3,10,replace=TRUE))
```

```
 [1] 1 3 1 1 2 1 3 3 1 2
Levels: 1 2 3
```

```r
factor(sample(1:3,10,replace=TRUE),levels=1:5)
```

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

Gestion: `nlevels,levels,table`

```
x <- factor(sample(c("thésard","CR","MdC"),15,replace=TRUE))
cat(nlevels(x),"niveaux:",levels(x))
```

```
3 niveaux: CR MdC thésard
```

```
table(x)
```

```
x
     CR     MdC thésard
      5       7       3
```

6.24 Un exemple de facteur associé à un vecteur

Un exemple de facteur associé à un vecteur

Données

```
Chacun me donne son âge et son grade^[sauf un qui refuse :'( ]
```

```
age <- c(25,35,32,27,32,40,26,25,26,28,30,NA,36,30,30)
grd <- c("thd","CR","MdC","thd","thd","MdC","MdC","thd","thd","MdC","CR","MdC","CR","thd","t
```

Question : nombre d'individus par catégorie ?

```
table(grd)
```

```
grd
 CR MdC thd
  3   5   7
```

6.25 La fonction `tapply`

Un autre point fort de R

Utilisation

```
Applique une fonction sur un vecteur partitionné en groupes.
```

Question : âge moyen / écart-type par catégorie ?

```
tapply(age,grd,mean,na.rm=TRUE)
```

```
      CR      MdC      thd
33.66667 31.50000 27.85714
```

```
tapply(age,grd,sd,na.rm=TRUE)
```

```
      CR      MdC      thd
3.214550 6.191392 2.794553
```

6.26 Matrices (et tableaux)

Les matrices et tableaux sont la des structure de stockage très courantes

6.27 Tableau: définition

objet `array`

```
Un tableau est un  vecteur muni d'un attribut dimension (`dim`),
lui même défini par un vecteur. Il est défini par la commande

`array(data,dim,dimnames=)`
```

Exemple

```
array(1:8,c(2,2,2))
```

```
, , 1

     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

6.28 Matrice: définition

objet `matrix`

Une matrice est un tableau à deux dimensions. Elle est définie par la commande

`` `matrix(data,nrow=,ncol=,byrow)` ``

En conséquence

- Un objet `array` à deux dimensions est automatiquement converti en `matrix`
- Un vecteur auquel on ajoute un atribut dimension est automatiquement converti en `matrix`

Exemple

```
class(array(1:4,c(2,2)))
```

```
[1] "matrix" "array"
```

```
x <- c(1,2,3,4)
dim(x) <- c(2,2)
class(x)
```

```
[1] "matrix" "array"
```

6.29 Remarques importantes

- R range les éléments d'une matrice par défaut par **colonne**.

```
matrix(1:6,nrow=2)
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
matrix(1:6,nrow=2,byrow=TRUE)
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

- Lors de la création d'une matrice, R **recycle** les éléments jusqu'à ce que les contraintes de dimension soient vérifiées.

6.30 Matrices: opérateurs élémentaires

Étant donné qu'une matrice est un vecteur pourvu d'une dimension, on a la proposition suivante:

```
**La   plupart   des   opérateurs   vectorielles   s'appliquent**
(arithmétiques/mathématiques, ensemblistes, d'indexation).
```

```
a <- matrix(sample(-4:4,9),3,3)
cat(max(a),sum(a),prod(a))
```

```
4 0 0
```

```
which(a > 0)
```

```
[1] 3 4 8 9
```

```
cumsum(a[a > 0])
```

```
[1]  2  5  6 10
```

```
order(a)
```

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

```
round(exp(a),4)
```

```
       [,1]    [,2]    [,3]
[1,] 0.0183 20.0855  0.0498
[2,] 0.3679  0.1353  2.7183
[3,] 7.3891  1.0000 54.5982
```

6.31 Manipulation de matrices

Opérateurs matriciels usuels

- `+,/,*,\^{}` sont les opérateurs usuels terme-à-terme,
- `\%*\%` est le produit matriciel,
- `crossprod()` est le produit scalaire,
- `t()` transpose une matrice,
- `diag()` extrait / spécifie la diagonale.

Exemples

```
a <- matrix(sample(-4:4,9),3,3)
b <- matrix(sample(a),3,3)
diag(a)
```

```
[1] -3  1  3
```

```
diag(a) <- diag(b) <- 1
diag(a)
```

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

```
a + t(b) %*% b
```

```
     [,1] [,2] [,3]
[1,]   27   18  -15
[2,]   21   27   -5
[3,]  -11  -11    7
```

6.32 Concaténation de matrices

Trois fonctions selon l'effet voulu:

- `c()` concatène les éléments de plusieurs matrices en un vecteur,
- `cbind()` empile **horizontalement** plusieurs matrices,
- `rbind()` empile **verticalement** plusieurs matrices.

Exemples

```
a <- matrix(1,2,3)
b <- matrix(2,2,3)
c(a,b)
```

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

```
cbind(a,b)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    1    1    2    2    2
[2,]    1    1    1    2    2    2
```

```
rbind(a,b)
```

```
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    2    2    2
[4,]    2    2    2
```

6.33 Algèbre linéaire élémentaire

Résolution de systèmes linéaires, inversion matricielle

La commande `solve` résout

$$\mathbf{A}x = \mathbf{b},$$

```
A <- matrix(c(4,2,8,-3),2,2)
b <- c(2,3)
solve(A,b)
```

```
[1]  1.0714286 -0.2857143
```

ou inverse une matrice:

```
round(solve(A) %*% A,8)
```

```
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

6.34 Commandes avancées d'algèbre linéaire

Utile pour l'analyse numérique

R dispose des outils classiques d'algèbre linéaire

- `det`: calcule le **déterminant** d'une matrice;
- `chol`: factorisation de **Cholesky** ($A = C^t C$, avec A symétrique, C triangulaire supérieure);
- `qr`: factorisation **QR** ($A = QR$ avec Q orthogonale, R triangulaire supérieure);
- `eigen`: calcule valeurs propres et **vecteurs propres** d'une matrice;
- `svd`: calcule la décomposition en **valeurs singulières**.
- …

6.35 Liste: définition

objet `list`

```
Une  liste est une  **collection d'objets  hétérogènes**. Elle
est  définie  par la  commande  `list(el1=,  el2=,  ...)`.  Les
éléments d'une liste peuvent posséder un nom.
```

```
list(c(1,2,3),c("robert","johnson"),matrix(rnorm(4),2,2))
```

```
[[1]]
[1] 1 2 3

[[2]]
[1] "robert"  "johnson"

[[3]]
           [,1]      [,2]
[1,] 0.81638552 0.5294257
[2,] 0.05985947 0.8633613
```

```
list(numero = c(1,2,3), noms = c("robert","johnson"), mat = matrix(rnorm(4),2,2))
```

```
$numero
[1] 1 2 3

$noms
[1] "robert"  "johnson"

$mat
          [,1]     [,2]
[1,] 1.1503739 1.452064
[2,] 0.9415722 1.231840
```

6.36 Accéder aux éléments

Deux situations

- Les éléments de la liste **ne sont pas nommés**: on accède au ieme élément par indexation `nom_liste[[i]]` uniquement.

- Les éléments de la liste **sont nommés**: on peut y accéder comme ci-dessus ou en utilisant le nom de l'élément `nom_liste$nom_elt`.

```
maliste <- list(numero = c(1,2,3), noms = c("robert","johnson"), mat = matrix(rnorm(4),2,2))
maliste$nom
```

```
[1] "robert"  "johnson"
```

```
maliste$nom[2]
```

```
[1] "johnson"
```

```
maliste[[2]]
```

```
[1] "robert"  "johnson"
```

```
maliste[[2]][2]
```

```
[1] "johnson"
```

[containsverbatim,allowframebreaks] ## Manipulation de listes

Sélectionner des éléments

```
Fonctionne (presque) comme pour les vecteurs
```

```
l1 <- list(1:2,c("a","c","g","t"))
l1[[-2]]
```

```
[1] 1 2
```

Commande `lapply`

```
Applique une fonction à chaque élément d'une liste
```

```
lapply(maliste,length)
```

```
$numero
[1] 3

$noms
[1] 2

$mat
[1] 4
```

Commande `c()`

Permet de concaténer deux listes.

```
c(list(1:2,c("a","c","g","t")),list(rnorm(3),"yop"))
```

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

[[2]]
[1] "a" "c" "g" "t"

[[3]]
[1] 0.8668089 0.1342110 0.6061756

[[4]]
[1] "yop"
```

6.37 Tableau de données: définition

Un autre point fort de R

objet `data.frame`

C'est une liste à laquelle on impose certaines contraintes^[que je vous épargne], afin de rassembler vecteurs et facteurs sous la forme d'un tableau de données.

- Pratiquement, **un tableau de données est une matrice dont les colonnes sont de mode différent**,
- C'est l'objet idéal pour la **manipulation de données** (**forcez-vous** à l'utiliser).

6.38 Création de tableau de données

Syntaxe

On peut spécifier le nom des colonnes par le vecteur `row.names`
ou directement comme pour une liste:

`data.frame(e1=,e2=, ...,row.names=)`

Exemple

```
age <- c(25,35,32,27,32,40,26,25,26,28,30,NA,36,30,30)
grd <- c("thd","CR","MdC","thd","thd","MdC","MdC","thd","thd","MdC","CR","MdC","CR","thd","t
sex <- factor(sample(c(rep("M",3),rep("F",12))))
donnees <- data.frame(age=age,grade=grd,sexe=sex)
head(donnees)
```

```
  age grade sexe
1  25   thd    M
2  35    CR    F
3  32   MdC    F
4  27   thd    F
5  32   thd    F
6  40   MdC    F
```

6.39 Manipulation des éléments du tableau de données

- Comme une liste !
- les commandes `attach()` `detach` placent ôtent les éléments du tableaux de données dans l'itinéraire de recherche.

```
donnees$age
```

```
 [1] 25 35 32 27 32 40 26 25 26 28 30 NA 36 30 30
```

```
attach(donnees,warn.conflicts=FALSE)
grade
```

```
 [1] "thd" "CR"  "MdC" "thd" "thd" "MdC" "MdC" "thd" "thd" "MdC" "CR"  "MdC"
[13] "CR"  "thd" "thd"
```

```
detach(donnees)
```

6.40 Travailler avec les tableaux de données

- beaucoup de fonctions prédéfinies
- penser aux fonctions `tapply` (ou `by`)

```
summary(donnees)
```

```
      age           grade           sexe
 Min.   :25.00   Length:15          F:12
 1st Qu.:26.25   Class :character   M: 3
 Median :30.00   Mode  :character
 Mean   :30.14
 3rd Qu.:32.00
 Max.   :40.00
 NA's   :1
```

```
attach(donnees,warn.conflicts=FALSE)
by(age,sexe,mean,na.rm=TRUE)
```

```
sexe: F
[1] 30.63636
------------------------------------------------------------
sexe: M
[1] 28.33333
```

```
by(age,grade,mean,na.rm=TRUE)
```

```
grade: CR
[1] 33.66667
------------------------------------------------------------
grade: MdC
[1] 31.5
------------------------------------------------------------
grade: thd
[1] 27.85714
```

```
detach(donnees)
```

6.41 Structures de contrôle

Les structure de contrôle sont les blocs d'un programme

6.42 Regrouper les expressions

Syntaxe

{expr_1; expr_2; …; expr_n }

ou

{
expr_1 …
expr_n

}

Remarques sur les groupes

- La dernière valeur du groupe est retournée;
- un groupe d'expressions peut être passé à une fonction, réutilisé dans une expression plus grande, etc.

6.43 Exécution conditionnelle: `if,if/else,ifelse`

Syntaxe

```
if (condition) {
  expr_1
 else {
  expr_2
```

ou {

```
ifelse(condition, a, b)
```

Remarques

- `condition` est une valeur logique: penser à `\&,|,!`, …
- le `else` est optionnel,
- `elseif` permet d'imbriquer les conditionnements.

6.44 Exécution répétée: boucle `for`

Syntaxe

```
for (var in set) {
  expr(var)
}
```

ou

```
for (var in set)
  expr(var)
```

à fuir pour éviter les effets de bords sournois!

Remarques sur la boucle `for`

- `var` est la variable incrémentée,
- `set` est un vecteur définissant les valeurs successives,
- **lente** comparée aux opérateurs matriciels.

6.45 Exécution répétée: boucles `while` et`repeat`‘

Syntaxe

```
while (condition) {
  expr

}
```

ou

```
repeat {
  expr
}
```

Remarque

- Comme pour `for`, éviter les imbrications sources de lenteur.

6.46 Contrôle des boucles: `break, next`

Exemples d'utilisation

```
repeat {
  expr
  if (condition) {break}
}
```

ou

```
while (condition1){
  expr_1
  if (condition2) {next}
  expr_2

}
```

Remarque

- `break` est la seule manière d'interrompre une boucle `repeat`.

6.47 Les fonctions

Permettent de factoriser des lignes de codes

6.48 Définir une fonction

Syntaxe

```
nom_de_la_fonction <- function(arg1,arg2, ...) {
  expression

  return(var)
}
```

Remarques

- `return` peut être omis (à éviter): dans ce cas la dernière valeur calculée est renvoyée.
- peut être tapée directement dans l'interpréteur ou dans un fichier externe `functions.R`, chargé par `source`.

6.49 Un exemple simple

Moyenne empirique d'un vecteur

Avec suppression des valeurs manquantes !

```
moyenne <- function(x){
  ## suppression des valeurs manquantes
  x.not.na <- x[!is.na(x)]
  ## moyenne empirique
  resultat <- sum(x.not.na) / length(x.not.na)
  return(resultat)
}
```

Tests

```
moyenne(rnorm(100))
```

```
[1] -0.08150755
```

```
moyenne(c(1,-5,3,NA,8.7))
```

```
[1] 1.925
```

6.50 Les arguments, leurs valeurs par défaut

Encore un point fort de R

Propriétés

```
-    les   arguments    peuvent   être   passés   dans   le
  **désordre** s'ils sont **nommés**: `var=object`,
-   on peut définir  une valeur  par défaut  pour n'importe
  quel argument lors de la définition de la fonction: `var=10`.
-  en cas de **sorties multiples**, les sorties doivent être
  renvoyées sous forme de liste.
```

Remarques

- Les valeurs par défaut rendent la lecture des fonctions beaucoup plus aisée pour l'utilisateur: **imposer peu d'arguments obligatoires}**.
- Les noms des éléments de la liste définie dans la fonction sont conservés à l'extérieur de la fonction.

6.51 Un exemple (un tout petit peu) plus avancé

Résumé numérique d'un vecteur

```r
resume <- function(x,na.rm=TRUE,affiche=FALSE) {
  mu <- mean(x,na.rm=na.rm)
  s2 <- var(x,na.rm=na.rm)
  if (affiche) {
    cat("\nMoyenne:",mu,"et variance:",s2)
  }
  return(list(moyenne = mu, variance = s2))
}
```

```r
out <- resume(rnorm(100))
out$variance
```

```
[1] 1.030765
```

```r
out <- resume(affiche=TRUE,x=rexp(100,0.5))
```

```
Moyenne: 2.027861 et variance: 3.537367
```

6.52 Les packages

Les packages sont des codes qui peuvent être inclus pour étendre les fonctionnalités de R

6.53 Motivations: reproductibilité de la recherche

Principe de Claerbout (Géophysicien, Stanford)

An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.

Démarche

- Proposer une méthode et exposer dans un article ses propriétés,
- Écrire et déposer un package sur CRAN,
- Publier un article dans « journal of statistical software » ou une note dans « Bioinformatics ».

6.54 Simplicité de la création d'un package

Définir un objectif

`Par exemple, `SIMoNe` : construire un graphe des interactions significatives entre gènes à partir de données du transcriptome.`

Organisation type

- Fichier DESCRIPTION
- Répertoire R : fonctions R (fonction `inferGraph(data)`)
- Répertoire man : documentation des fonctions
- Répertoire data : données
- (Répertoire src : pour les fichiers à compiler, header etc.)

6.55 R vs Python

Critère	R	Python
Philosophie	Spécialisé en statistiques et analyse exploratoire	Langage polyvalent et généraliste
Syntaxe	Fonctionnelle	Orientée objet
Structures principales	vecteurs, matrices, dataframes, listes	listes, dictionnaires, ensembles, tuples, dataframes
Bibliothèques principales	tidyverse, ggplot2, caret, Shiny	NumPy, pandas, matplotlib, scikit-learn
Gestion des données	Puissante, avec des dataframes natifs	Très performante, via pandas
Visualisation	ggplot2 (simple et élégant)	matplotlib, seaborn (flexibles mais plus complexes)
Performance	Moyenne (optimisable avec Rcpp)	Bonne (optimisable avec bibliothèques natives)
Machine learning	Bibliothèques spécialisées, moins variées	Bibliothèques avancées variées (scikit-learn, TensorFlow, PyTorch)
Utilisation typique	Statistiques, recherche académique	Industrie, recherche, web, IA
Communauté et popularité	Forte communauté académique (statistiques)	Très large communauté diversifiée

6.56 Intégration de Python dans R: Reticulate

Voir

- Reticulate website,
- Rbloggers about the subject

```
# Set the RETICULATE_PYTHON environment variable to your Python executable
Sys.setenv(RETICULATE_PYTHON = "~/python3-machine-learning/bin/python")

# Load the reticulate package
library(reticulate)

# Verify the Python configuration
py_config()
```

```
python:         /Users/cambroise/python3-machine-learning/bin/python
libpython:      /opt/homebrew/opt/python@3.13/Frameworks/Python.framework/Versions/3.13/lib/p
pythonhome:     /Users/cambroise/python3-machine-learning:/Users/cambroise/python3-machine-le
version:        3.13.2 (main, Feb  4 2025, 14:51:09) [Clang 16.0.0 (clang-1600.0.26.6)]
numpy:          /Users/cambroise/python3-machine-learning/lib/python3.13/site-packages/numpy
numpy_version:  2.2.1

NOTE: Python version was forced by RETICULATE_PYTHON
```

Installation d'un package

```
py_install("pandas")
```

```
# Python code block
import sys
print(f"Python version: {sys.version}")
```

```
Python version: 3.13.2 (main, Feb  4 2025, 14:51:09) [Clang 16.0.0 (clang-1600.0.26.6)]
```

6.57 Exemple

```
import seaborn as sns

df = sns.load_dataset('iris')  # ou 'tips', 'titanic', 'penguins', etc.
print(df.head())
```

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

Dans R il est possible de récupérer les variables de python:

```
library(ggplot2)
ggplot(py$df,aes(sepal_length,petal_length,color=species)) + geom_point()
```

```
module '__main__' has no attribute 'df'
```

7 De l'intérêt des statistiques descriptives: coefficient de Gini

7.1 Le capital au 21ème siècle

- Livre d'économie publié en 2013 et écrit par Thomas Piketty, paru aux éditions du Seuil
- Bestseller aux états unis

Figure 2: Le capital au 21ème siècle

7.2 Quelle thèse ?

- les inégalités de revenu, les inégalités de patrimoine et le rapport capital/revenu dans les pays développés suivent chacun une courbe en U
- on retrouve au début du xxie siècle des niveaux d'inégalités comparables aux niveaux d'inégalités du xixe siècle et du début du xxe siècle.

7.3 Quelles données, quel résumé ?

- Si l'on se concentre sur le revenu des personnes appartenant à des ménages fiscaux
- A partir des impôts on peut accéder revenu de chaque ménage du pays avant et après impôt.
- L'INSEE utilise une mesure du revenu par unité de consommation, à l'aide d'une échelle d'équivalence. L'échelle la plus utilisée actuellement consiste à décompter
  - 1 unité de consommation (UC) pour le premier adulte du ménage,
  - 0,5 UC pour les autres personnes de 14 ans ou plus,
  - 0,3 UC pour les enfants de moins de 14 ans.

7.4 Coefficient de Gini

- Le coefficient ou indice de Gini porte le nom du statisticien et démographe italien Corrado Gini (1884–1965).

- indicateur de dispersion permettant principalement d'apprécier les inégalités dans la distribution des richesses d'un territoire
- varie entre zéro et un,
- zéro étant la situation d'égalité parfaite (chaque citoyen est exactement aussi riche que son voisin),
- un étant la situation d'inégalité parfaite (un citoyen possède toutes les richesses, les autres aucune).

Les pays du monde s'ordonnent ainsi entre 0,25 (pays d'Europe scandinave et centrale) et 0,70 (pays émergents d'Amérique latine, d'Afrique centrale)

7.5 Coefficient de Gini

Courbe de Lorenz

$$L(F(x) = p) = \frac{\int_{-\infty}^{x} t\, f(t)\, dt}{\int_{-\infty}^{\infty} t\, f(t)\, dt} = \frac{\int_{-\infty}^{x} t\, f(t)\, dt}{\mu}$$

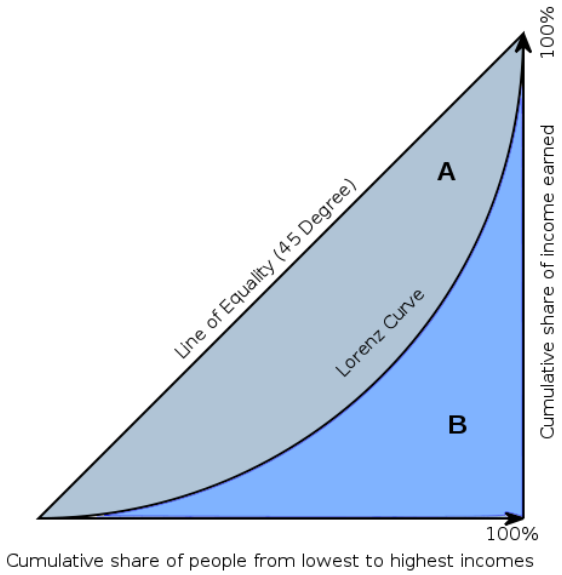


Figure 3: Coefficient de Gini

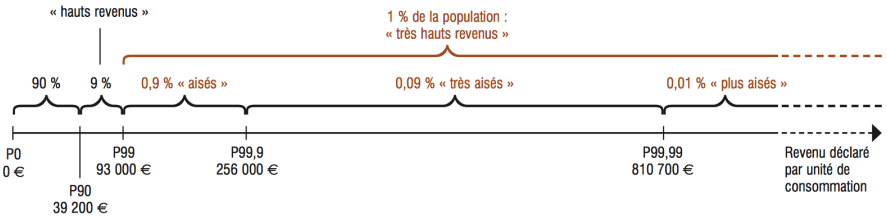
$$G = 2A = 1 - 2B$$

7.6 Coefficient de Gini

Si x_i est la richesse ou le revenu de la personne i, et qu'il y a n personnes, alors le coefficient de Gini G est donné par :

$$G = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} |x_i - x_j|}{2 \sum_{i=1}^{n} \sum_{j=1}^{n} x_j} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} |x_i - x_j|}{2n^2 \bar{x}}$$

7.7 Revenu en France en 2011



Champ : France métropolitaine, personnes appartenant à des ménages fiscaux dont le revenu déclaré par unité de consommation est strictement positif.
Lecture : 0,01 % de la population a un revenu déclaré par unité de consommation supérieur à 810 700 euros pour l'année 2011.
Sources : DGFiP, exhaustif fiscal 2011, calculs Insee.

Figure 4: Revenu en France en 2011

7.8 Evolution du coefficient de Gini en France

	1996	1999	2002	2005	2008	2009	2010	2010[1]	2011[1]
Seuils de niveau de vie (en milliers d'euros 2011)									
Niveau de vie médian (D5)	16,7	17,3	18,5	18,7	19,7	19,8	19,7	19,6	19,6
Premier décile de niveau de vie (D1)	8,9	9,5	10,3	10,3	10,9	10,8	10,6	10,6	10,5
Neuvième décile de niveau de vie (D9)	31,2	32,8	35,1	34,6	36,9	37,2	37,0	36,7	37,5
Rapports interdéciles									
D9/D1	3,4	3,5	3,4	3,4	3,4	3,4	3,5	3,5	3,6
D9/D5	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9
D5/D1	1,9	1,8	1,8	1,8	1,8	1,8	1,8	1,8	1,9
Masses de niveau de vie détenues									
S20 (en %)	9,0	9,1	9,3	9,0	9,0	8,9	8,7	8,7	8,6
S50 (en %)	31,0	30,9	31,1	31,0	30,9	30,7	30,2	30,1	29,8
S80 (en %)	63,0	62,3	62,3	62,0	61,6	61,8	61,0	60,7	60,5
(100-S80)/S20	4,1	4,1	4,1	4,2	4,3	4,3	4,5	4,5	4,6
Indice de Gini	**0,279**	**0,284**	**0,281**	**0,286**	**0,289**	**0,290**	**0,299**	**0,303**	**0,306**

1. À partir de 2010, les estimations de revenus financiers mobilisent l'enquête Patrimoine 2010 (*encadré 1*).
Champ : France métropolitaine, personnes vivant dans un ménage dont le revenu déclaré au fisc est positif ou nul et dont la personne de référence n'est pas étudiante.
Lecture : les 20 % les plus modestes disposent en 2011 de 8,6 % de la somme des revenus disponibles par UC (S20), les 20 % les plus aisés perçoivent 39,5 % de la somme des revenus disponibles par UC (complément à 100 de S80), soit 4,6 fois plus.
Sources : Insee, enquêtes Revenus fiscaux et sociaux rétropolées de 1996 à 2004, enquêtes Revenus fiscaux et sociaux 2005 à 2011 ; DGI ; DGFiP ; Cnaf ; Cnav ; CCMSA.

Figure 5: Evolution du coefficient de Gini en France

7.9 Coefficient de Gini dans le monde

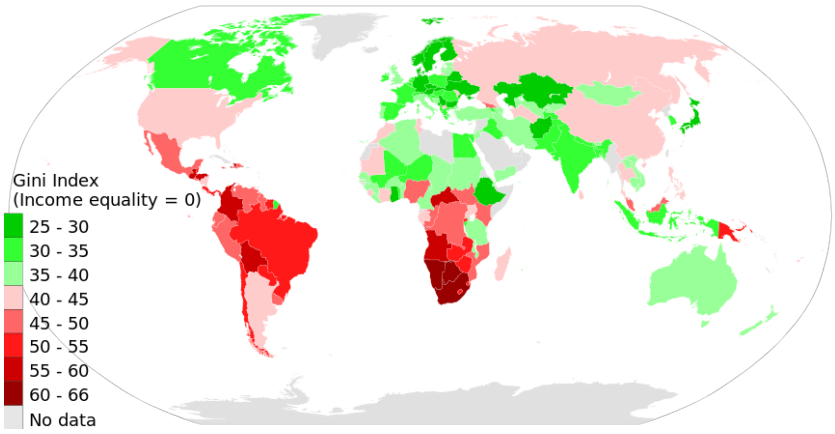


Figure 6: Coefficient de Gini dans le monde

8 Concepts fondamentaux

8.1 Qu'est ce que la statistique ?

Activité qui consiste dans le recueil, le traitement et l'interprétation de données d'observation issues d'une population:

- une branche des mathématiques appliquées
- une méthode

- un ensemble de techniques

8.2 Population

- population, ensemble d'entités objet de l'investigation statistique
- individus, définis comme les éléments d'une certaine population

Différentes notion de population

- dans certain cas la population de référence est **finie** et ses éléments peuvent être explicitement dénombrés
- la notion de population revêt parfois une signification plus abstraite (exemple population de malades)

- parfois la notion de population s'identifie avec celle de procédure de génération de données (données d'expression)

8.3 Variables, distributions

Variables

Chaque individu est décrit par un ensemble de variables:

- **qualitative** (sexe, nationalité, état matrimonial, ...): les valeurs prises par le caractère sont les modalités
  - ordinale (notion d'ordre): modalités intrinsèquement ordonnées
  - nominale : pas de structure d'ordre : par exemple le sexe.
- **quantitative** (taille, poids, …)
  - discrète
  - continue

8.4 Modes d'étude d'une population

- Étude exhaustive Dite par recensement. L'étude d'une population de grande taille est souvent difficile voire impossible
- Échantillon. Le processus de sélection d'un échantillon est l'échantillonnage. Seule solution d'une le cas d'une population **infinie**

8.5 Inférence statistique

Processus visant à

- déduire de conclusions générales relative à la population totale
- à partir de la connaissance partielle relative à un nombre de cas particulier

8.6 Objectifs d'une étude statistique

1. **Statistique Descriptive ou Exploratoire** : synthétiser, résumer, structurer l'information
2. **Statistique inférentielle** : formuler ou valider des hypothèses relatives à la population totale

8.7 Le tableau de données

- n mesurés par p
- Tableau

$$X = (x_i^j) = \begin{pmatrix} x_1^1 & x_1^j & x_1^p \\ & \cdot & \cdot \\ x_i^1 & x_i^j & x_i^p \\ & \cdot & \cdot \\ x_n^1 & x_n^j & x_n^p \end{pmatrix}$$

- Chaque variable est représentée par le vecteur $x^j = (x_1^j, ..., x_n^j)'$
- Chaque individu est représenté par le vecteur $sx_i = (x_i^1, ..., , x_i^p)'$
- X : réalisation d'un échantillon de taille n du de dimension p:

$$X = (X^1, ..., X^p)'$$

9 Statistiques descriptives

9.1 Analyse statistique descriptive classique : des analyses univariées aux approches multivariées

1. Phase exploratoire et univariée
2. Approches bivariées
3. Approches multivariées
4. Interprétation des résultats

9.2 En pratique

- processus itératif :
  - analyses univariées
  - exploration de relations plus complexes entre les variables
- les méthodes dépendent des objectifs de l'analyse et des caractéristiques du jeu de données (nature des variables, relation entre variables, …)
- interprétation prudente: variables manquantes,limitations des méthodes employées (linéaires, …), …

9.3 Phase exploratoire et univariée

Distribution des variables

- Moyenne, médiane, mode, écart-type, etc. pour résumer les caractéristiques principales
- Histogrammes, diagrammes en boîtes, etc. pour visualiser la forme de la distribution

Analyse des valeurs aberrantes

- Identification des points qui s'écartent significativement de la distribution.
- Étude de leur influence sur les résultats et décision de les supprimer ou non.

9.4 Approches bivariées

Détecter la **Dépendance entre paires de variables**

- représentation graphique
- statistiques
- tests

9.5 Approches multivariées

Analyse factorielle

- Réduction de la dimensionnalité d'un ensemble de variables corrélées.
- Identification des facteurs explicatifs principaux.

9.6 Clustering

- Affectation des individus à des groupes pré-définis en fonction de leurs caractéristiques.
- Utilisation d'algorithmes comme k-means

9.7 Interprétation des résultats

- Interpréter les résultats de manière claire et concise.
- Tenir compte des limitations des méthodes utilisées.
- Présenter les résultats de manière visuelle et pédagogique.
- L'interprétation des résultats doit être faite avec prudence et en tenant compte des limitations des méthodes employées

9.8 Observations

Nous considérerons dans un premier temps une seule colonne du tableau de données, soient n observations:

$$x_1, ..., x_n$$

9.9 Variable quantitative discrète ou qualitative ordinale

La variable prend ses valeurs dans

$$V_x = \{\epsilon_1, ..., \epsilon_K\}$$

avec

$$\epsilon_1 < ... < \epsilon_K$$

Tableau de fréquence

- ϵ_k, la modalité
- n_k, l'effectif des observations ayant la valeur ϵ_k
- $f_k = \frac{n_k}{n}$, la fréquence
- $F_k = \sum_{j=1}^{k} f_j$, la fréquence relative cumulée

9.10 Variable qualitative nominale

- Même représentation sans l'ordre.
- Pas de fréquence cumulée.

9.11 Camembert, diagramme en barres, radar

Couplés à la commande `table` Le diagramme en barres et le graphe en camembert permettent de visualiser le découpage d'une population en donnée catégorielle.

```
  pop pepin.08 poids.08 volcm3.08
1  CE      1.0     0.89      7.70
2  CE      1.0     1.14      8.82
3  CE      1.2     1.26     10.20
4  CE      1.2     0.66        NA
5  CE      1.2     0.83        NA
6  CE      1.3     0.54      4.61
```

```r
attach(vigne)
par(mfrow=c(1,2))
pie(table(pop))
barplot(table(pop),las=3)
```

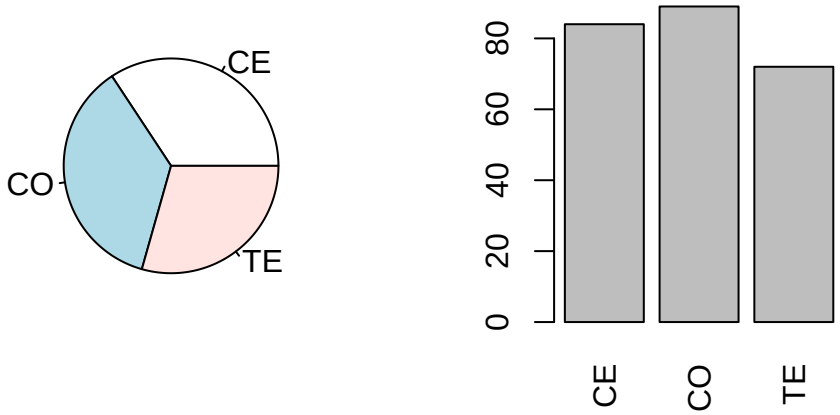
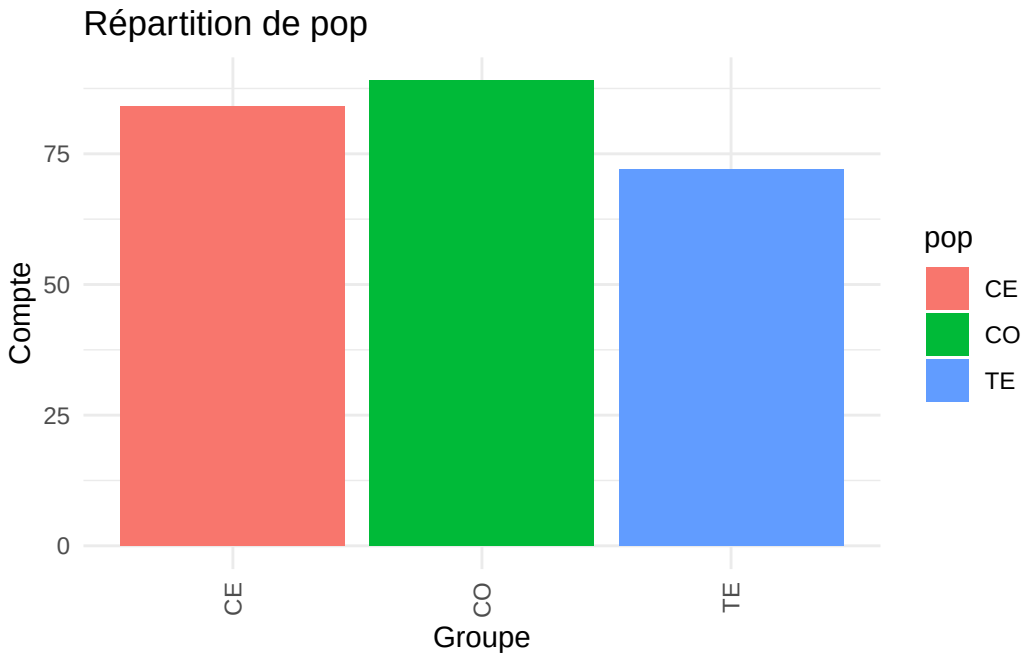


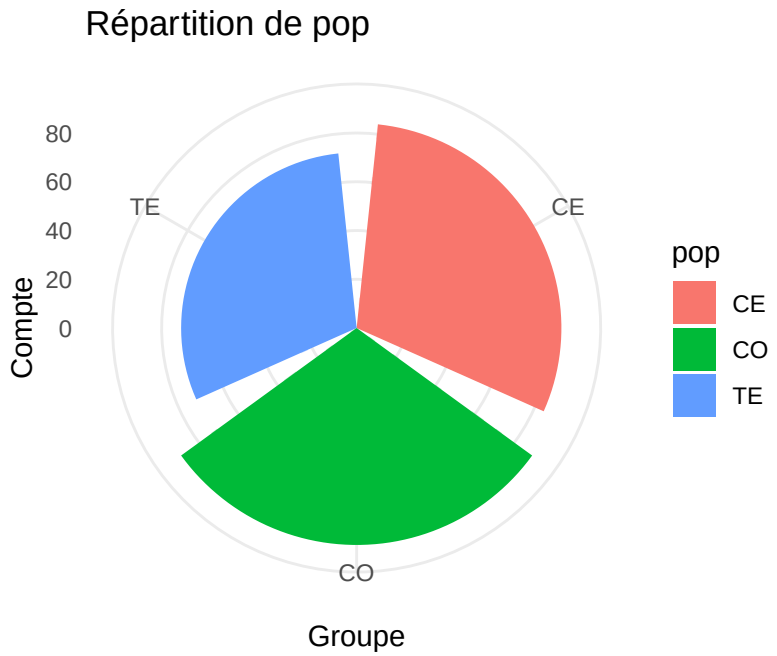
Figure 7: Camembert des différentes populations

9.12 En ggplot2

```r
ggplot(vigne, aes(x = pop,fill=pop)) +
  geom_bar() + # geom_bar() compte les occurrences par défaut
  theme_minimal() +
  labs(title = "Répartition de pop", x = "Groupe", y = "Compte") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) # Pour faire pivoter le
```




```
ggplot(vigne, aes(x = pop,fill=pop)) +
  geom_bar() + # geom_bar() compte les occurrences par défaut
  coord_polar()+
  theme_minimal() +
  labs(title = "Répartition de pop", x = "Groupe", y = "Compte")
```



9.13 Variable continue, résumés numériques

Tendance centrale

- Moyenne: $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$
- Remarque: la somme des écarts à la moyenne empirique est nulle

$$\sum_i (x_i - \bar{x}) = 0$$

- Inconvénient : problème des valeurs aberrantes
- Moyenne tronquée: $M_k = \frac{1}{n-2k}\sum_{i=k+1}^{n-k} x_{(i)}$ où $x_{(i)}$ est l'observation de rang i
- Médiane:

$$M = \begin{cases} x_{(n/2)} \text{ si } n \text{ est pair,} \\ x_{(\lfloor n/2 \rfloor + 1)} \text{ sinon} \end{cases}$$

- Fractile empirique d'ordre α

$$\hat{f}_\alpha = \begin{cases} x_{(n\alpha)} \text{ si } n\alpha \text{ est entier,} \\ x_{(\lfloor n\alpha \rfloor + 1)} \text{ sinon} \end{cases}$$

9.14 Indicateurs de dispersion

- la variance empirique: $s^2 = \frac{1}{n}\sum_i (x_i - \bar{x})^2$
- la variance empirique corrigée: $s^2 = \frac{1}{n-1}\sum_i (x_i - \bar{x})^2$
- l'étendue: $max_i x_i - min_i x_i$
- l'étendue interquartile $EI = Q_3 - Q_1 = \hat{f}_{3/4} - \hat{f}_{1/4}$

9.15 Reprenons l'exemple de la vigne

- Renommons les variables et considérons uniquement les données de 2008, pour une manipulation plus agréable.
- J'enlève également la colonne variété

9.16 Résumé statistique 'Commande summary

Le résumé numérique s'adapte selon la nature des variables (univariée, multivariée, factorielle)

```
summary(pepin.08)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  0.000   1.400   1.800   1.858   2.400   3.200      27
```

```
summary(pop)
```

```
CE CO TE
84 89 72
```

```
knitr::kable(summary(vigne))
```

pop	pepin.08	poids.08	volcm3.08
CE:84	Min. :0.000	Min. :0.390	Min. : 3.44
CO:89	1st Qu.:1.400	1st Qu.:0.820	1st Qu.: 7.14
TE:72	Median :1.800	Median :1.060	Median : 8.91
NA	Mean :1.858	Mean :1.212	Mean :10.47
NA	3rd Qu.:2.400	3rd Qu.:1.360	3rd Qu.:11.90
NA	Max. :3.200	Max. :3.750	Max. :33.80
NA	NA's :27	NA's :28	NA's :44

9.17 Résumé statistique `Package Hmisc`

```
library(Hmisc)
```

```
Attaching package: 'Hmisc'

The following objects are masked from 'package:base':

    format.pval, units
```

```
describe(vigne)
```

```
vigne

 4  Variables      245  Observations
--------------------------------------------------------------------------------
pop
       n  missing distinct
     245        0        3

Value         CE    CO    TE
Frequency     84    89    72
Proportion 0.343 0.363 0.294
--------------------------------------------------------------------------------
pepin.08
       n  missing distinct     Info     Mean  pMedian      Gmd      .05
     218       27       26    0.997    1.858      1.9    0.753      1.0
     .10      .25      .50      .75      .90      .95
     1.2      1.4      1.8      2.4      2.7      2.9

lowest : 0   0.5 0.8 1   1.1, highest: 2.8 2.9 3   3.1 3.2
--------------------------------------------------------------------------------
poids.08
       n  missing distinct     Info     Mean  pMedian      Gmd      .05
     217       28      122        1    1.212     1.11   0.6078    0.578
     .10      .25      .50      .75      .90      .95
   0.676    0.820    1.060    1.360    1.868    2.472

lowest : 0.39 0.4  0.41 0.42 0.49, highest: 3.31 3.39 3.43 3.47 3.75
--------------------------------------------------------------------------------
volcm3.08
       n  missing distinct     Info     Mean  pMedian      Gmd      .05
     201       44      188        1    10.47    9.485    5.306     4.71
     .10      .25      .50      .75      .90      .95
    5.98     7.14     8.91    11.90    16.58    23.06

lowest : 3.44  3.48  3.97  4.13  4.32 , highest: 26.97 27.59 28.29 28.54 33.8
--------------------------------------------------------------------------------
```

9.18 Tableau de fréquences

Dans le cas où la variable x est continue, la réalisation d'un tableau de fréquence nécessite un partitionnement préalable du domaine de définition en K classes de largeur

- constante
- ou variable

9.19 Fonction de répartition empirique

$$\hat{F} : \mathbb{R} \mapsto [0,1],\ x \mapsto \frac{1}{n} card\{i : x_i \leq x\}$$

- Le graphe de la fonction de répartition est une fonction en escalier appelé diagramme cumulatif

9.20 Graphe en tiges et feuilles

Un graphe en tige est feuille permet de visualiser le tableau des fréquences

```
stem(pepin.08)
```

```
  The decimal point is 1 digit(s) to the left of the |

   0 | 00000000
   2 |
   4 | 0
   6 |
   8 | 0
  10 | 00000000000
  12 | 0000000000000000000
  14 | 00000000000000000000000000000000000000
  16 | 000000000000000000000
  18 | 0000000000000000000000
  20 | 000000000000000000000
  22 | 0000000000000000
  24 | 00000000000000000000
  26 | 0000000000000000000
  28 | 00000000
  30 | 000
  32 | 0000000
```

9.21 Fonction de répartition empirique

Crée un objet qui peut être tracé avec .

```
par(mfrow=c(1,2))
plot(ecdf(poids.08[pop!="TE"]))
plot(ecdf(poids.08[pop=="TE"]))
```

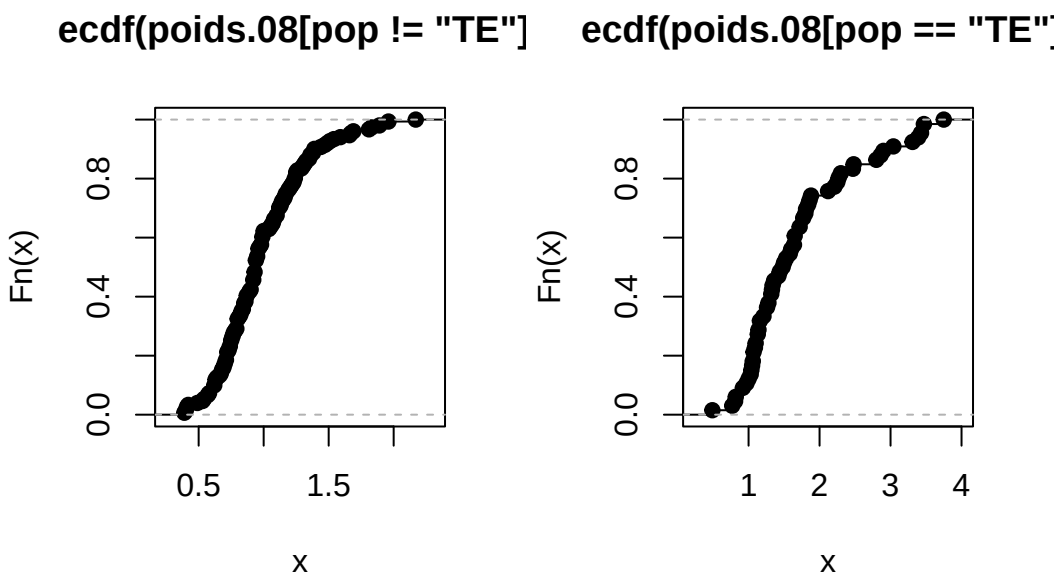


Figure 8: F.d.r empirique du poids pour les non TE et les TE

9.22 Comparaison de distribution

Pour comparer visuellemnt deux distributions, la manière la plus efficace est le graphe quantile/quantile (qui doivent correspondre si les distributions sont proches.)

```
qqplot(poids.08[pop=="CE"],poids.08[pop=="CO"])
```

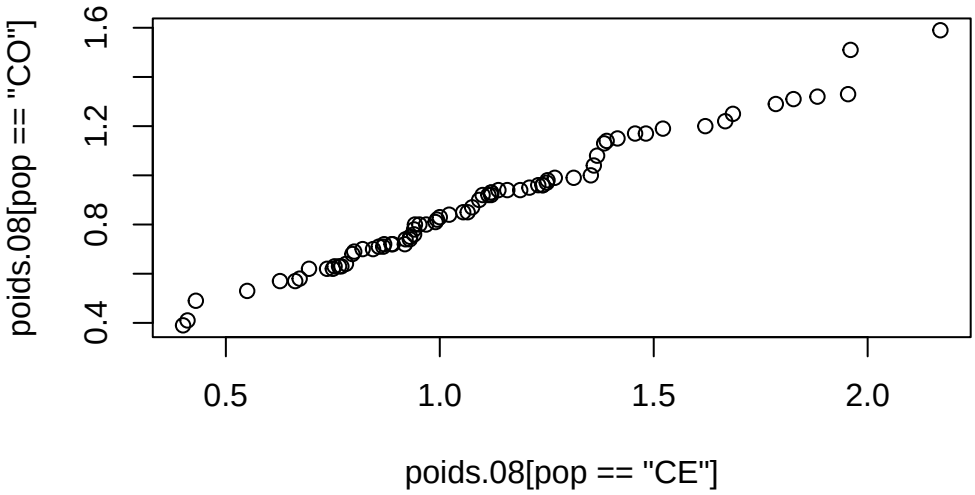


Figure 9: qqplot des distributions de poids de CE et CO

9.23 Comparaison de distribution en ggplot2

```
library(ggplot2)
# Séparation des données
poids_CE <- vigne$poids.08[vigne$pop == "CE"]
poids_CO <- vigne$poids.08[vigne$pop == "CO"]

# Générer le QQ plot avec qqplot mais sans l'afficher, pour récupérer les points
qq <- qqplot(poids_CE, poids_CO, plot.it = FALSE)

# Créer un dataframe avec les points
qq_data <- data.frame(x = qq$x, y = qq$y)

# Utiliser ggplot2 pour afficher le QQ plot
ggplot(qq_data, aes(x = x, y = y)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color = "red") +
  labs(title = "QQ plot de poids.08 pour CE vs CO",
       x = "Quantiles de CE",
       y = "Quantiles de CO") +
  theme_minimal()
```

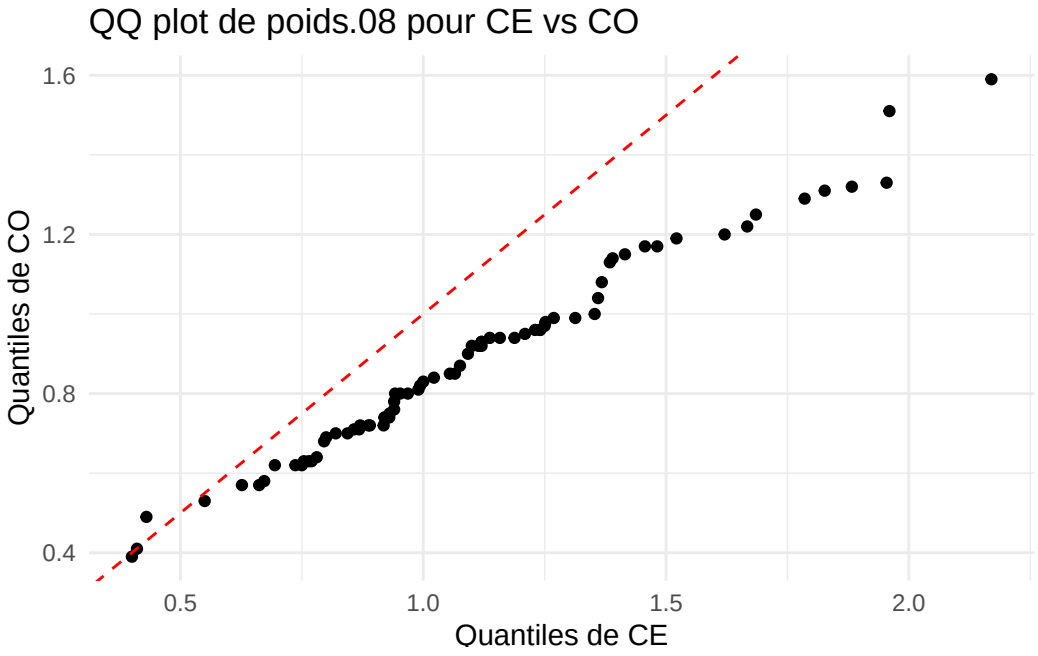


Figure 10: qqplot des distributions de poids de CE et CO

9.24 Histogramme et estimateur à noyaux

- Estimateur de la fonction de densité

$$\hat{f}_n(x) = \sum_i h_i \mathbb{I}_{[a_i, a_{i+1}[}(x) \quad a_1 < ... < a_{k+1}$$

- Découpage en intervalles
- Calcul de la fréquence
- Aire du rectangle proportionnel à la fréquence

$$\sum_i h_i(a_{i+1} - a_i) = 1 \text{ et } h_i(a_{i+1} - a_i) = \hat{P}_F(X \in [a_i, a_{i+1}[)$$

- Attention : hauteur proportionnelle à la fréquence si et seulement si les intervalles ont tous la même largeur
- Nombre d'intervalles :
  - Important
  - Réglage difficile
  - Règle empirique : règle de Sturges $1 + 10/3 * \log_{10}(n)$

```r
hist(pepin.08,nclass=25,prob=TRUE)
lines(density(pepin.08[!is.na(pepin.08)]),col="red")
```

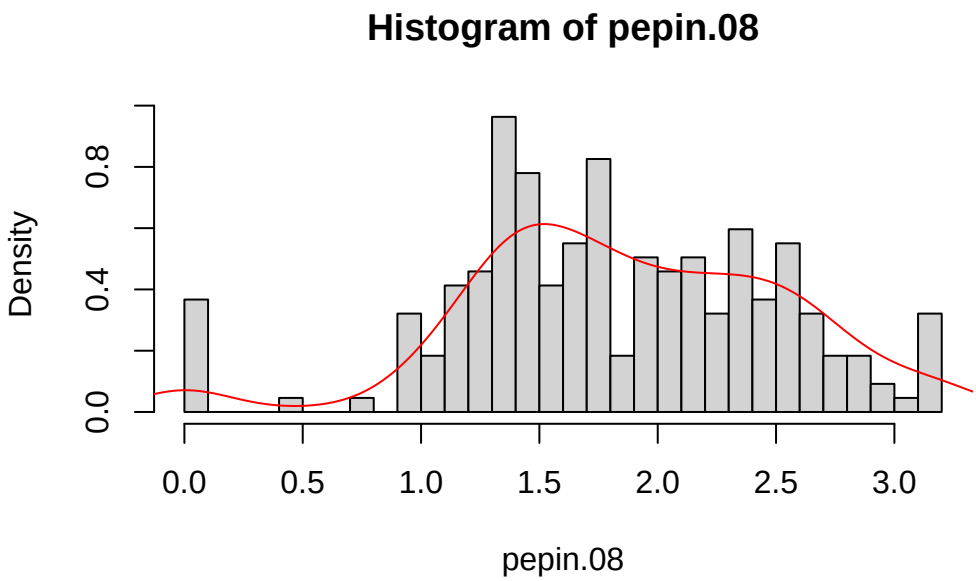


Figure 11: histogramme du nombre de pépins par baie en 2008

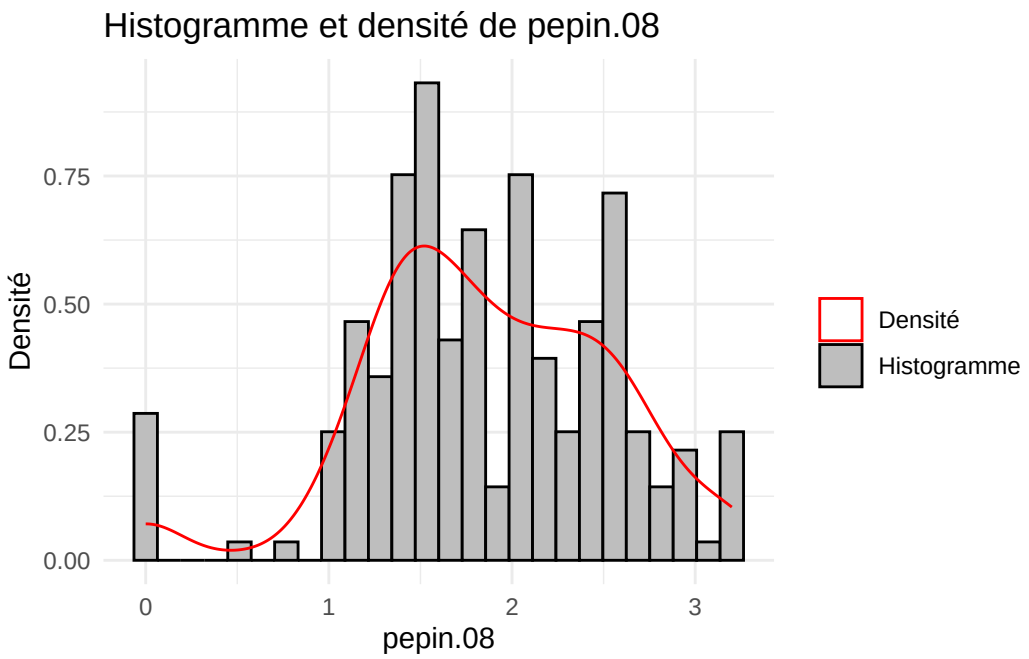
9.25 En ggplot2

```r
ggplot(data = vigne, aes(x = pepin.08)) +
  geom_histogram(aes(y = ..density.., color = "Histogramme"), binwidth = diff(range(vigne$pep
  geom_density(aes(color = "Densité")) +
  scale_color_manual(name = "", values = c("Histogramme" = "black","Densité" = "red"), labels
  labs(title = "Histogramme et densité de pepin.08",
       x = "pepin.08",
       y = "Densité") +
  theme_minimal()
```

```
Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
i Please use `after_stat(density)` instead.
```

```
Warning: Removed 27 rows containing non-finite outside the scale range
(`stat_bin()`).
```

```
Warning: Removed 27 rows containing non-finite outside the scale range
(`stat_density()`).
```



9.26 Boite à moustache

- Éléments atypiques (aberrants, *outliers*)
  - Notion arbitraire
  - Règle empirique assez souvent utilisée : valeurs situées à l'extérieur de $[q_1 - 1.5 \times Iqr, q_3 + 1.5 \times Iqr]$
- Définition : Graphique constitué
  - d'un rectangle délimité par les quartiles et partagé en deux par la médiane
  - d'une paire de moustaches : minimum et maximum de l'échantillon auquel on a ôté les éléments atypiques
  - des outliers eux-mêmes

9.27 Boîtes à moustaches

La boîte à moustache permet de visualiser les grands traits caractéristiques d'une distribution.

```r
boxplot(poids.08~pop,col=c("red","green3","steelblue"),notch=T)
```

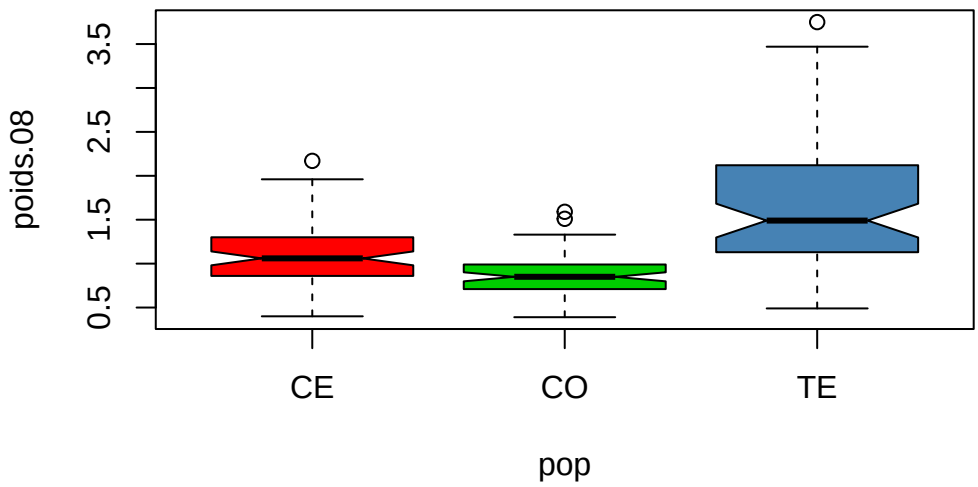
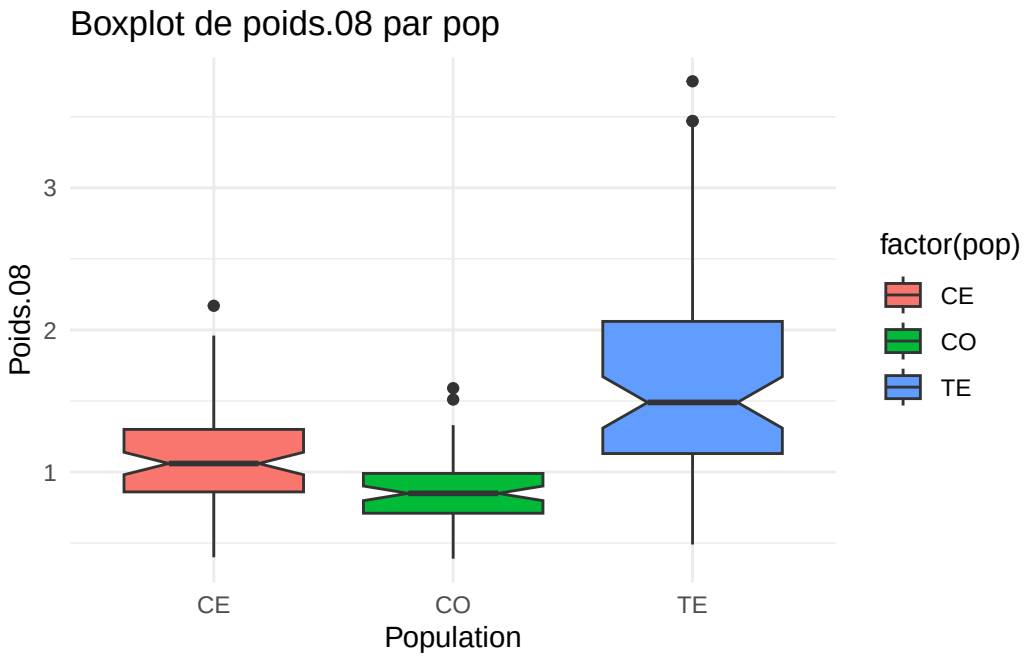


Figure 12: boîtes à moustaches du poids selon les populations

9.28 En ggplot

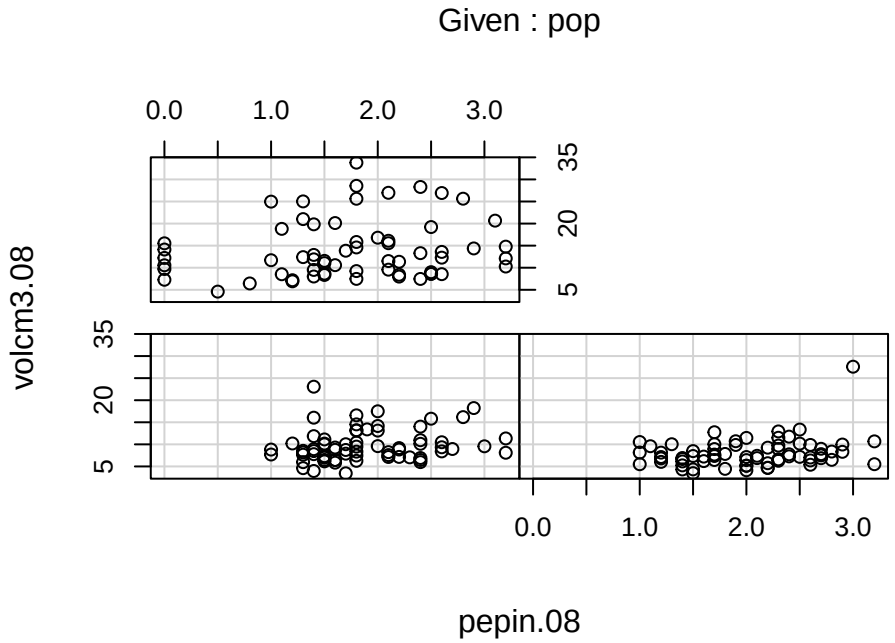
```r
ggplot(vigne, aes(x = pop, y = poids.08, fill = factor(pop))) +
  geom_boxplot(notch = TRUE) +
  labs(title = "Boxplot de poids.08 par pop",
       x = "Population",
       y = "Poids.08") +
  theme_minimal()
```

```
Warning: Removed 28 rows containing non-finite outside the scale range
(`stat_boxplot()`).
```



9.29 Graphe conditionné par une variable

```
coplot(volcm3.08~pepin.08 | pop,show.given=FALSE)
```

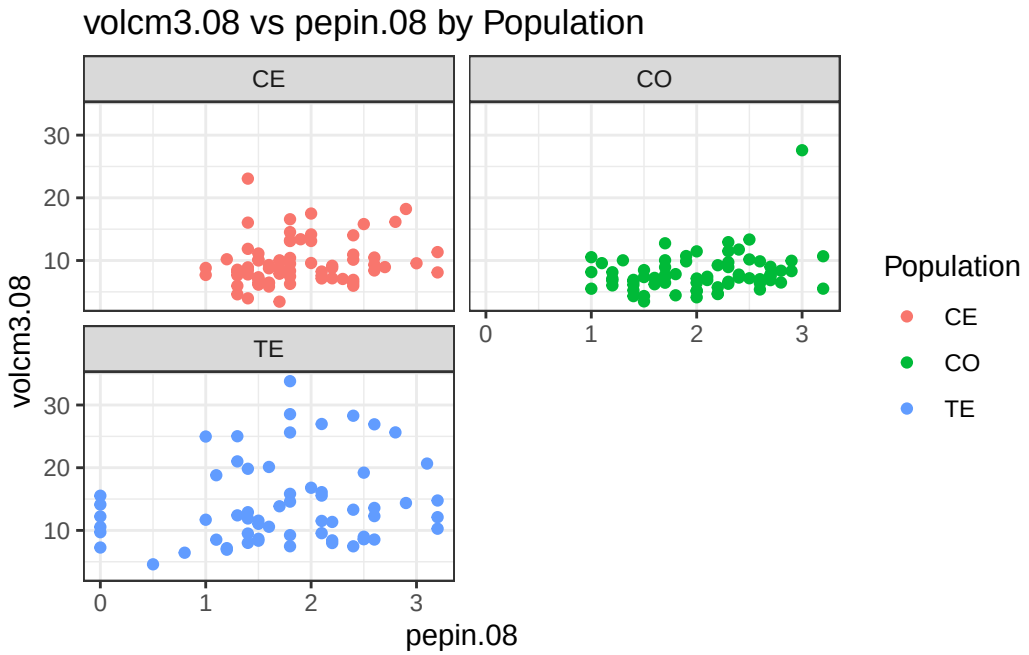


```
 Missing rows: 4, 5, 11, 19, 20, 21, 49, 71, 78, 79, 80, 81, 82, 83, 84, 99, 112, 129, 139,
```

9.30 Graphe conditionné par une variable (ggplot)

```
ggplot(data = vigne,
       aes(x = pepin.08, y = volcm3.08, color = pop)) +
  geom_point() +
  facet_wrap(~ pop, nrow = 2) +
  labs(title = "volcm3.08 vs pepin.08 by Population",
       x = "pepin.08",
       y = "volcm3.08",
       color = "Population") +
  theme_bw()
```

```
Warning: Removed 44 rows containing missing values or values outside the scale range
(`geom_point()`).
```



9.31 Conclusion

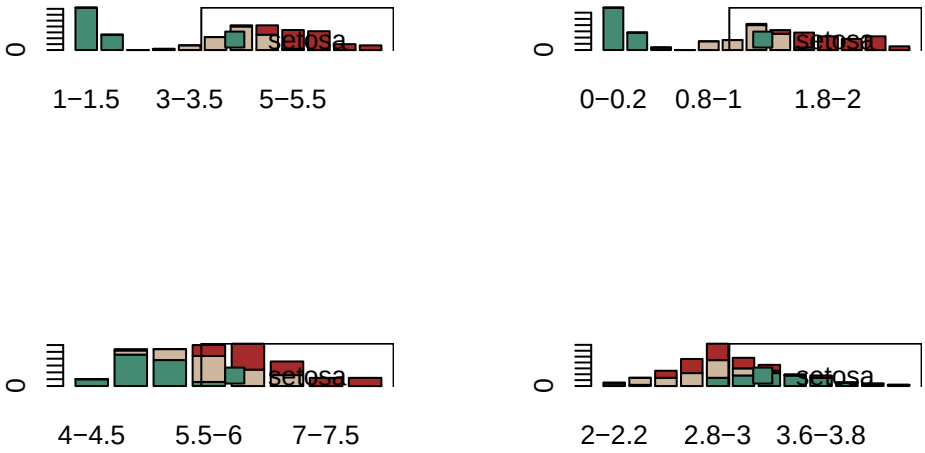
Mise en évidence de certaines caractéristiques :

- Présence de données atypiques
- Absence de symétrie de la distribution
- Présence de populations hétérogènes
- …

10 Description bidimensionnelle

10.1 Histogrammes et variable qualitative

```
histy<-function(x,y){
    mycolors<-colors()[seq(2,600,by=10)]
    hist(x,plot=FALSE)$breaks->breaks
    length(levels(y))->nbLevels
    z<-matrix(0,nbLevels,length(breaks)-1)
    for (l in 1:nbLevels){
        xl<-x[as.numeric(y)==l]
        inter<-matrix(c(breaks[-length(breaks)],breaks[-1]),ncol=2)
        apply(inter,1,function(interi) sum((interi[1]<xl) & (xl<=interi[2]))) -> z[l,]
        }
    as.table(z)->z
    dimnames(z)[2]<-list(apply(inter,1,function(x) toString(paste(x[1],x[2],sep="-"))))
    dimnames(z)[1]<-list(levels(y))
    barplot(z,col=mycolors[2:(nbLevels+2)])
    legend("topright",levels(y),fill= mycolors[2:(nbLevels+2)])
    invisible(z)
    }
par(mfrow=c(2,2))
histy(iris$Petal.Length,iris$Species)
histy(iris$Petal.Width,iris$Species)
histy(iris$Sepal.Length,iris$Species)
histy(iris$Sepal.Width,iris$Species)
```



10.2 Statistiques associées à un vecteur aléatoire

- Rappels
  - X réalisation d'un échantillon de taille n du vecteur aléatoire X
  - x_i réalisation de taille 1 de X
  - x^j réalisation d'un échantillon de taille n de X^j

10.3 Moyenne et variance

- Moyenne empirique

$$\overline{x} = (\overline{x}^1, \dots, \overline{x}^p)' \quad \text{où} \quad \overline{x}^j = \frac{1}{n}\sum_{i=1}^{n} x_i^j$$

- Variance empirique

$$s_j^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i^j - \overline{x}^j)^2$$

10.4 Covariance vs Corrélation empirique

- Covariance empirique

$$s_{jj'} = \frac{1}{n}\sum_{i=1}^{n}(x_i^j - \overline{x^j}).(x_i^j - \overline{x^{j'}})$$

- Corrélation empirique

$$r_{jj'} = \frac{s_{jj'}}{s_j s_{j'}}$$

10.5 Matrice de variance empirique

$$S = (s_{jj'}) = \frac{1}{n}(X - 1_n \overline{x})'(X - 1_n \overline{x}) = \frac{1}{n} Y'Y$$

où 1_n est la matrice de dimension $(n, 1)$ remplie de 1 et Y est la matrice centrée associée à X.

- empirique

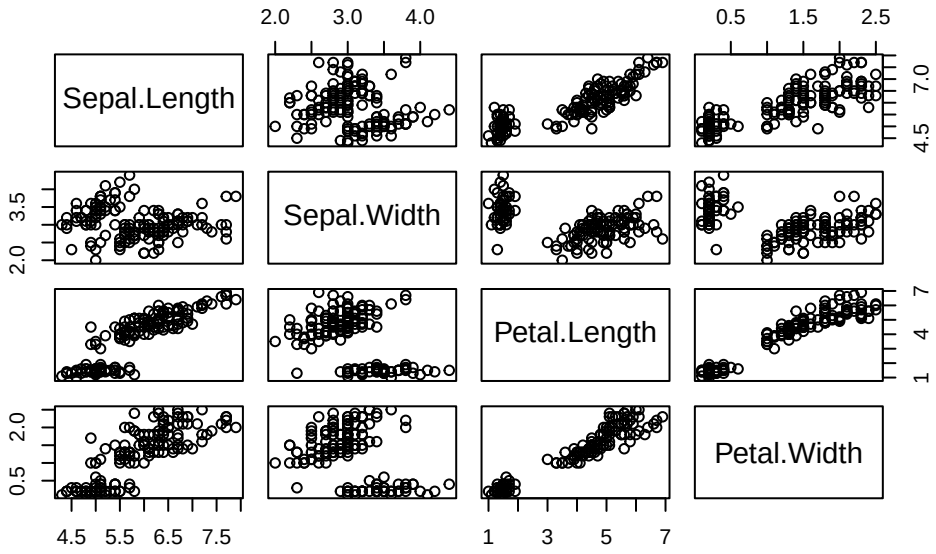
$$R = (r_{jj'}) = D_{1/s_j} S D_{1/s_j}$$

10.6 Graphique de dispersion

- Représentation de chaque individu i par le point du plan (x_i^1, x_i^2)
- Nuage de n points dans le plan
- Visualisation synthétique des données : permet de voir
  - les relations linéaires
  - les regroupements en classes homogènes

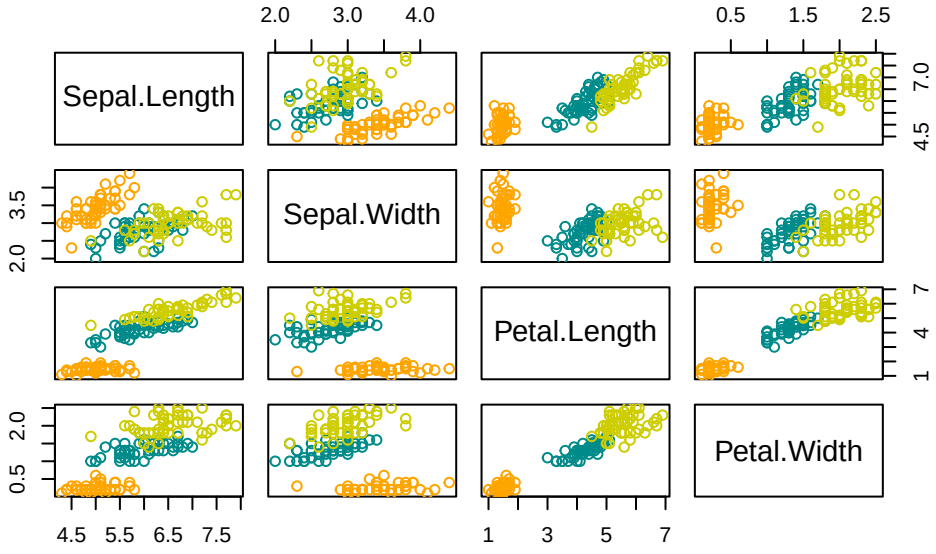
10.7 Les 5 variables des iris (pairs)

```
data(iris)
pairs(iris[,1:4])
```



10.8 Les 5 variables des iris en couleurs (pairs)

```
pairs(iris[,1:4],col=c('orange','cyan4','yellow3')[iris$Species])
```



10.9 Les 5 variables des iris en couleurs (ggpairs)

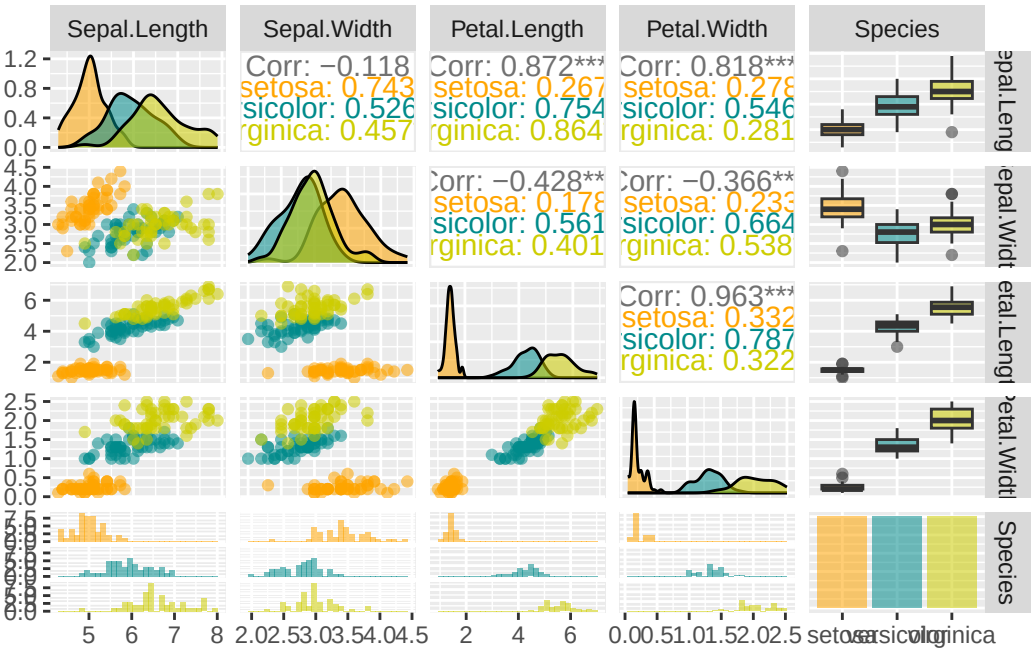
```
library(GGally)
```

```
Registered S3 method overwritten by 'GGally':
  method from
  +.gg   ggplot2
```

```
colors <- c("orange","cyan4",  "yellow3")
ggpairs(iris,aes(color = Species, alpha = 0.5))+
scale_color_manual(values = colors, aesthetics = c("colour", "fill"))
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



10.10 Covariance et corrélation

- 2 variables : covariance et corrélation empirique
- $>$ 2 variables : matrices de cov. et de corr. empiriques

Les iris

Table 7: Matrice de covariance

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	0.6856935	-0.0424340	1.2743154	0.5162707
Sepal.Width	-0.0424340	0.1899794	-0.3296564	-0.1216394
Petal.Length	1.2743154	-0.3296564	3.1162779	1.2956094
Petal.Width	0.5162707	-0.1216394	1.2956094	0.5810063

Table 8: Matrice de corrélation

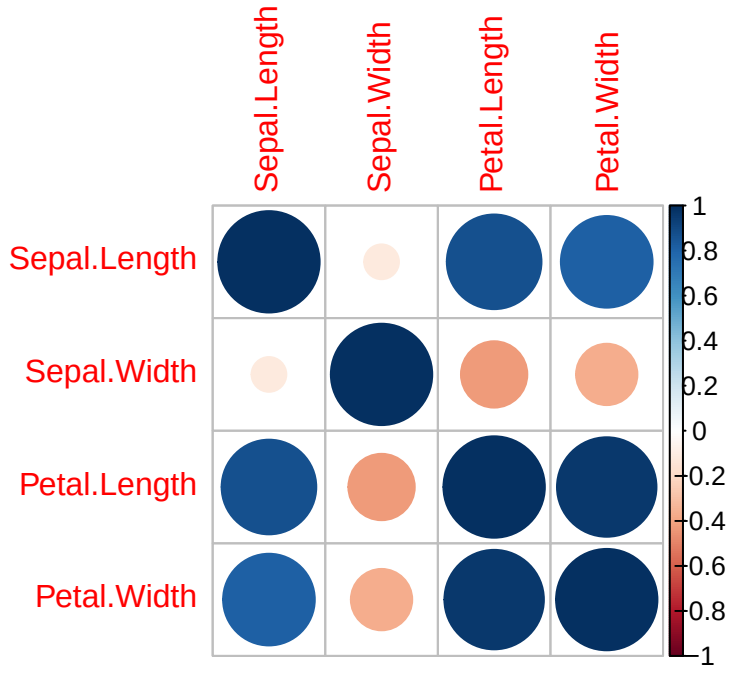
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	0.6856935	-0.0424340	1.2743154	0.5162707
Sepal.Width	-0.0424340	0.1899794	-0.3296564	-0.1216394
Petal.Length	1.2743154	-0.3296564	3.1162779	1.2956094
Petal.Width	0.5162707	-0.1216394	1.2956094	0.5810063

10.11 Représentation graphique (corrplot)

```
library(corrplot)
```

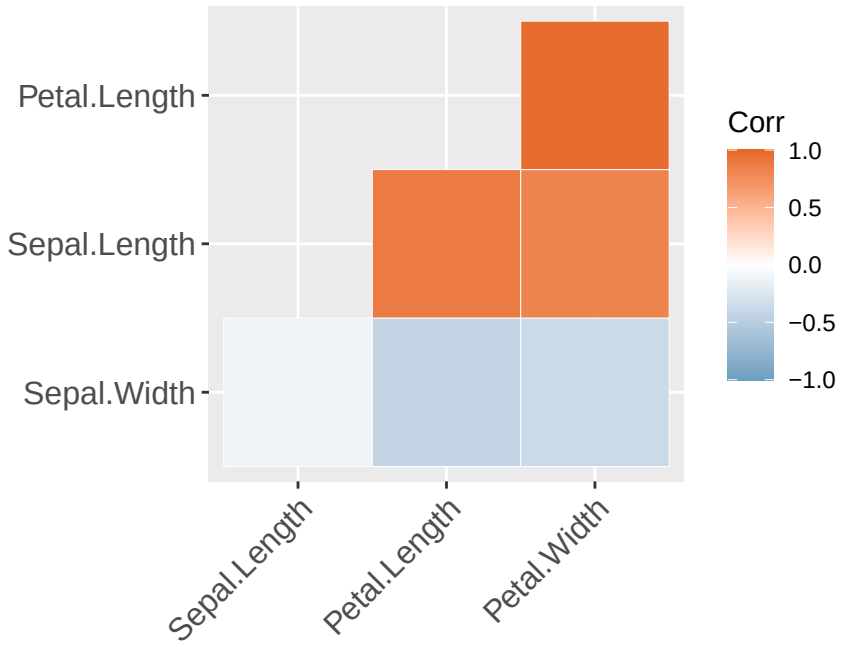
```
corrplot 0.95 loaded
```

```
data(iris)
corrplot(cor(iris[, 1:4]))
```



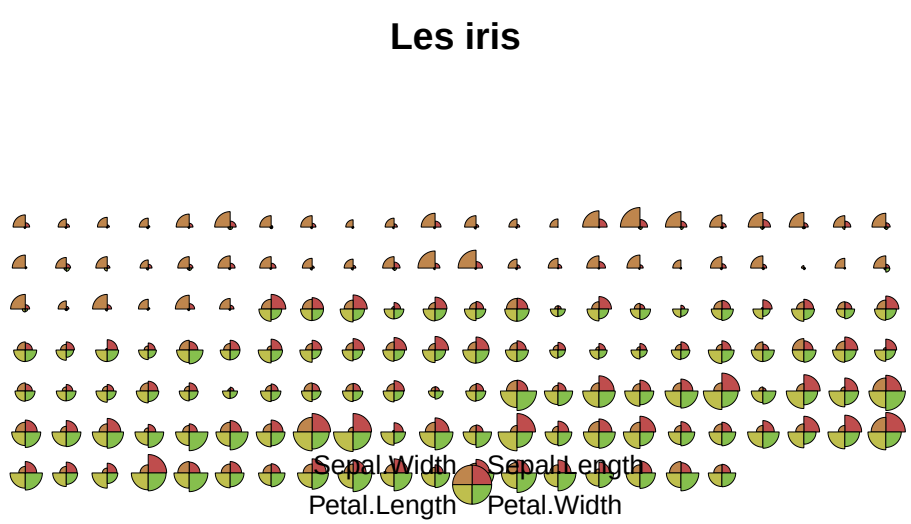
10.12 Représentation graphique (ggcorrplot)

```
library(ggcorrplot)
data(iris)
ggcorrplot(cor(iris[, 1:4]),
             hc.order = TRUE, type = "lower",
  outline.color = "white",
  ggtheme = ggplot2::theme_gray,
  colors = c("#6D9EC1", "white", "#E46726")
)
```



10.13 Description multidimensionnelle

Les diagrammes fleurs



10.14 Fléau de la dimension (curse of dimensionality)

- Espace de grande dimension

- Calculs similaires à ceux du plan
- Difficile de généraliser

Exemple 1 :

- Dans $\mathbb{R}$ - Pts uniformément répartis dans $[-1, +1]$ - % de points situées à 1 distance ≤ 0.75 de l'origine : 75%
- Dans $\mathbb{R}^{10}$ - Pts uniformément répartis dans $[-1, +1]^{10}$ - % de points situées à 1 distance ≤ 0.75 de l'origine : 5%

Exemple 2 :

on veut construire un histogramme en s'appuyant sur au moins une moyenne de 10 points par intervalle et 10 classes par variable

- $\mathbb{R}$: 10 classes $n = 100$
- $\mathbb{R}^2$: 100 classes $n = 1000$
- $\mathbb{R}^{10}$: 10^{10} classes $n = 10^{11} = 100 milliards$
- Si p assez grand, l'espace $\mathbb{R}^p$ est pratiquement vide et sauf si les données se situent au voisinage d'une variété de faible dimension, l'analyse des données n'apportera aucune information intéressante.
- Les points voisins d'un point donné sont tous très loin : difficultés dans l'emploi de méthodes du type k-plus proches voisins