Analyser des données avec R

Pierre-Yves de Müllenheim

2021-01-03

Contents

Introduction

Ce livre vise à servir de support à des enseignements auprès d'étudiants qui découvrent les analyses statistiques en même temps que le langage R et le logiciel RStudio. Au cours de la lecture, l'objectif est non seulement de découvrir les analyses statistiques qui permettent de répondre à des questions que l'on se pose à partir de données que l'on a en sa possession, mais aussi, en parallèle, de développer une certaine maîtrise du langage R avec le logiciel RStudio afin de réaliser les analyses désirées.

Ce livre n'est donc pas dédié exclusivement aux statistiques, ni exclusivement dédié à R. D'autres ouvrages proposent des contenus plus spécialisés, et donc plus poussés, et c'est vers ces ouvrages qu'il convient d'aller pour davantage maîtriser les fondamentaux des statistiques ou toutes les subtilités du langage R.

Bien que le langage R puisse être utilisé via le logiciel R en tant que tel, les procédures décrites dans ce livre sont relatives à l'utilisation du logiciel RStudio, qui est plus confortable en matière d'utilisation. Comme il le sera montré, R/RStudio n'est pas un logiciel "clique-bouton". Il s'agit d'un logiciel où il convient d'écrire des lignes de code pour obtenir le résultat souhaité. Si cela peut s'avérer plus complexe et/ou fastidieux à utiliser au départ par rapport à un logiciel classique où il suffit de cliquer sur des boutons pour obtenir une analyse particulière, cela vaut la peine de prendre le temps d'apprendre à utiliser R, au moins pour les raisons suivantes : les logiciels R et RStudio sont en accès libre sur internet ; il existe de nombreuses aides documentaires, notamment sur internet, qui permettent de faire probablement n'importe quelle analyse/manipulation de données, aussi sophistiquée soit elle ; il est possible de conserver les lignes de code pour refaire rapidement des analyses déjà effectuées, ou pour appliquer ces lignes de code à d'autres données, ou encore pour partager ces lignes de code avec d'autres personnes. De par sa gratuité et sa capacité à permettre le partage des analyses réalisées, R est ainsi une très bonne option pour embrasser la tendance actuelle, encore timide, de l'*open science*, consistant à permettre de savoir ce qui a été fait en matière d'analyses et à pouvoir reproduire ces analyses. Enfin, le langage R, notamment via l'utilisation de RStudio, permet de faire bien plus que des analyses de données (programmation, construction de site internet et de curriculum vitae, etc.), ce qui en fait un outil de travail

polyvalent et donc particulièrement intéressant.

Chapter 1

Prérequis

1.1 Installation de R et RStudio

R et RStudio sont deux logiciels en libre accès sur internet. Le logiciel R peut être utilisé indépendamment de l'utilisation du logiciel RStudio. En revanche, l'utilisation du logiciel RStudio requiert au préalable l'installation du logiciel R. En effet, RStudio est un logiciel qui permet d'utiliser les fonctionnalités de R tout en proposant une interface d'utilisation plus agréable et fonctionnelle que l'interface à l'origine proposée pour le logiciel R, qui est très basique. Les explications présentées au cours de ce document considèrent que l'utilisateur fonctionne avec Rstudio.

1.1.1 Installer R sur WINDOWS

- Aller sur le site https://cran.rstudio.com.
- Sur la page web qui s'affiche, l'encart du haut **Download and Install R** montre les différents liens de téléchargement possibles selon le système d'exploitation utilisé. Cliquer sur le lien **Download R for Windows**.
- Dans la nouvelle fenêtre qui vient de s'ouvrir, cliquer sur **install R for the first time**.
- Dans la nouvelle fenêtre qui vient de s'ouvrir, cliquer sur le premier lien en haut de la page : **Download R X.X.X. for Windows**. Exécuter le fichier s'il est proposé de le faire. Si ce n'est pas le cas, il est probable que le téléchargement du fichier ait été lancé automatiquement. Retrouver le fichier ainsi téléchargé sur votre PC (une fois son enregistrement terminé), puis exécuter le fichier.
- Suivre la procédure d'installation par défaut en cliquant à chaque fois sur **Suivant**.

- Une fois l'installation terminée, double-cliquer sur l'icône du bureau (**R x64 X.X.X**) pour vérifier que l'ouverture du logiciel R s'effectue correctement.

1.1.2 Installer R sur MAC

- Aller sur le site https://cran.rstudio.com.
- Sur la page web qui s'affiche, l'encart du haut **Download and Install R** montre les différents liens de téléchargement possibles selon le système d'exploitation utilisé. Cliquer sur le lien **Download R for (Mac) OS X**.
- Dans la nouvelle fenêtre qui s'ouvre, cliquer sur le lien qui correspond à votre version OS X (le clic entraînera le début du téléchargement du fichier).
- Sur le Mac, chercher dans le dossier **Téléchargements** le fichier téléchargé.
- Double-cliquer sur le fichier téléchargé pour lancer l'installation du logiciel R.
- Suivre la procédure par défaut et terminer l'installation.
- Lorsque l'installation est terminée, aller dans le dossier **Applications** du Mac pour rechercher le logiciel R. Double-cliquer sur l'icône pour ouvrir le logiciel et vérifier que l'ouverture se déroule correctement.

1.1.3 Installer RStudio sur WINDOWS ou MAC

- Aller sur le site https://www.rstudio.com.
- En haut de la page d'accueil, mettre le curseur de la souris sur **Products**, puis, dans le menu déroulant qui apparaît, cliquer sur **RStudio**.
- Dans la nouvelle fenêtre qui vient de s'ouvrir, faire défiler la page légèrement vers le bas jusqu'à voir apparaître l'option de téléchargement **DOWNLOAD RSTUDIO DESKTOP**, puis cliquer dessus.
- Dans la nouvelle fenêtre qui vient de s'ouvrir, faire défiler la page vers le bas puis cliquer sur l'option **DOWNLOAD** dans la colonne dédiée au téléchargement de RStudio Desktop.
- **Sur WINDOWS :** Exécuter le fichier s'il est proposé de le faire. Si le clic entraîne automatiquement le téléchargement du fichier, retrouver alors ce fichier sur le PC (une fois son enregistrement terminé), puis exécuter le fichier et suivre la procédure d'installation par défaut. Une fois le logiciel installé, retrouver le fichier d'exécution du logiciel sur votre PC (chemin d'accès possible : **Ordinateur > Windows (C:) > Programmes > RStudio > bin > rstudio.exe**). Créer un raccourci pour le fichier **rstudio.exe** (clic droit sur le fichier > Créer un raccourci) et mettre le raccourci sur le bureau du PC. Double-cliquer sur l'icône RStudio afin de vérifier si l'ouverture s'effectue correctement.

- **Sur MAC :** Le téléchargement débute lorsque vous cliquez sur le lien. Une fois le téléchargement terminé, sur votre MAC, réaliser l'installation en double-cliquant sur le fichier téléchargé et en suivant la procédure indiquée. Une fois l'installation terminée, aller dans le dossier **Applications** du MAC et double-cliquer sur l'icône de RStudio pour vérifier si le logiciel s'ouvre correctement.

1.2 Prise en main de RStudio

Le logiciel R et son interface RStudio se présentent en quelque sorte comme un super calculateur : on écrit une instruction (i.e., une ligne de code) dans une fenêtre, on lance cette instruction, et le logiciel nous donne le résultat, qu'il s'agisse d'un calcul, d'un graphique, d'une modification d'un jeu de données, etc. Quand on ouvre RStudio pour la première fois, la fenêtre principale qui se présente est la **Console**. Cette fenêtre permet d'y écrire des lignes de code et de les lancer en appuyant sur la touche **Entrée**. Lorsque l'on souhaite conserver les lignes de code que l'on a écrites, ou que l'on souhaite écrire des lignes de code sans forcément les lancer, il est possible d'utiliser une fenêtre **Script** (chemin d'accès : **File** > **New File** > **R Script**). Pour lancer les lignes de code qui sont écrites dans une fenêtre de script, il suffit de se placer n'importe où sur la ligne de code et de cliquer sur l'icône **Run** du logiciel (raccourci : **Ctrl** + **Entrée**). Une fois le code activé, celui-ci et son resultat sont montrés dans la Console.

1.2.1 Manipuler des objets (valeurs, vecteurs, et tableaux de données)

De manière basique, R permet d'effectuer des opérations simples avec des nombres, telles que des additions (avec le symbole `+`), des soustractions (avec le symbole `-`), des multiplications (avec le symbole `*`), des divisions (avec le symbole `/`) , des racines carrées (avec la fonction `sqrt()`), ou encore des élévations à la puissance (avec le symbole `^`).

```
(9 + 3 - 5) * 5 / 2 + sqrt(9) ^ 2
```

```
## [1] 26.5
```

De manière plus élaborée, R permet de créer des vecteurs (des suites de nombres), notamment grâce à la fonction `c()`, et de les manipuler avec différentes sortes d'opérations. Lorsqu'une opération ou une série d'opérations est appliquée à un vecteur, chaque valeur du vecteur subit les opérations spécifiées. Dans l'exemple ci-dessous, on voit par exemple que chaque valeur du vecteur a été multipliée par 2 et s'est vue ajouter la valeur 3.

```
c(0, 1, 2, 3, 4, 5) * 2 + 3
```

```
## [1]  3  5  7  9 11 13
```

Si des vecteurs peuvent contenir des nombres, ils peuvent également contenir des caractères, tels que de simples lettres ou des mots, ces vecteurs ne pouvant cependant pas, par nature, subir des opérations mathématiques. Pour pouvoir être crées, les caractères doivent être écrits à l'intérieur du vecteur avec des guillemets (`" "`).

```
c("a", "b", "c")
```

```
## [1] "a" "b" "c"
```

```
c("Pierre", "Marie", "Jean")
```

```
## [1] "Pierre" "Marie"  "Jean"
```

De manière encore plus élaborée, R permet de créer des tableaux de données à partir de vecteurs à l'aide de la fonction `data.frame()`. Dans l'exemple ci-dessous, les noms x, y, et z, marqués à gauche du signe `=`, sont les noms des vecteurs que contiendra le tableau de données. À droite du signe `=`, on retrouve la fonction `c()` qui permet de créer un vecteur avec des valeurs à l'intérieur.

```
data.frame(x = c(0, 1, 2, 3),
           y = c(3, 5, 7, 9),
           z = c("a", "b", "c", "d"))
```

```
##   x y z
## 1 0 3 a
## 2 1 5 b
## 3 2 7 c
## 4 3 9 d
```

1.2.2 Manipuler des objets assignés à des noms

L'une des particularités de R, c'est de permettre d'associer des objets (des valeurs, des vecteurs, ou encore des tableaux de données) à des noms. Pour ce faire, R utilise la fonction d'assignation `<-`. Cette fonction s'utilise en écrivant à droite de la flèche l'objet que l'on veut associer à un nom, et en écrivant à gauche de la flèche le nom désiré. (Attention : Toujours utiliser seulement des

caractères alphanumériques et des points . ou des tirets du bas _ pour écrire un nom ; ne pas commencer par un chiffre ; avoir à l'esprit que R est sensible à la casse, ce qui veut dire qu'un nom commençant par une majuscule sera un nom différent de celui qui a les mêmes lettres mais qui commence par une minuscule.) L'utilisation de noms associés à des objets permet de grandement faciliter les analyses par la suite. Lorsqu'on réalise une assignation, il est possible de voir le nouveau nom et l'objet associé dans la fenêtre **Environnement** de RStudio. Lorsqu'on lance le code permettant d'assigner un objet à un nom, R ne montre pas le contenu de l'objet. Pour le voir, il faut écrire le nom associé à l'objet dans une ligne de code et lancer la commande.

Il est donc possible d'associer à un nom un objet qui serait une valeur numérique...

```
a <- 9
a
```

```
## [1] 9
```

... ou encore une succession de caractères ...

```
Prenom <- "Pierre"
Prenom
```

```
## [1] "Pierre"
```

... ou encore un vecteur ...

```
Taille <- c(178, 191, 178, 182, 167, 151)
Taille
```

```
## [1] 178 191 178 182 167 151
```

```
Poids <- c(60, 89, 92, 67, 80, 70)
Poids
```

```
## [1] 60 89 92 67 80 70
```

```
Sexe <- c("M", "M", "F", "F", "M", "F")
Sexe
```

```
## [1] "M" "M" "F" "F" "M" "F"
```

... et même un tableau de données, qui aurait été soit conçu à la main, soit conçu à partir d'objets de type vecteurs qui auraient été crées auparavant, comme ci-dessous. (À noter que les vecteurs doivent être de la même longueur (i.e., ils doivent contenir le même nombre de valeurs) pour pouvoir être combinés dans un tableau de données avec la fonction `data.frame()`.)

```
df <- data.frame(Taille, Poids, Sexe)
df
```

```
##   Taille Poids Sexe
## 1    178    60    M
## 2    191    89    M
## 3    178    92    F
## 4    182    67    F
## 5    167    80    M
## 6    151    70    F
```

Lorsqu'un objet de type tableau de données est assigné à un nom, il est possible d'afficher le contenu d'une seule colonne de ce tableau à partir du nom associé au tableau, du symbole `$`, et du titre de la colonne désirée.

```
df$Taille
```

```
## [1] 178 191 178 182 167 151
```

Une fois que des objets sont liés à des noms, il est possible, comme montré initialement avec des valeurs, d'utiliser ces noms pour réaliser des opérations. Par exemple, via des noms, on peut manipuler des objets contenant simplement une valeur numérique...

```
a <- 7
b <- 3
c <- 2
(a + b) / c
```

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

ou alors des objets contenant un vecteur ...

```
vec1 <- c(0, 2, 4, 6, 8)
vec2 <- c(1, 4, 5, 9, 0)
vec1 * 10
```

```
## [1]  0 20 40 60 80
```

```
vec1 * vec2
```

```
## [1]  0  8 20 54  0
```

ou encore des objets contenant un tableau de données, en créant par exemple une variable à partir d'autres variables du tableau.

```
df$IMC <- df$Poids / (df$Taille/100) ^ 2
df
```

```
##   Taille Poids Sexe      IMC
## 1    178    60    M 18.93700
## 2    191    89    M 24.39626
## 3    178    92    F 29.03674
## 4    182    67    F 20.22703
## 5    167    80    M 28.68514
## 6    151    70    F 30.70041
```

Lorsque plusieurs objets ont été assignés à des noms, il est possible de vouloir supprimer certaines assignations, par exemple en raison du fait qu'un objet aurait été assigné par erreur. Pour supprimer une assignation, il est possible d'utiliser la fonction `rm()`. L'instruction `rm(list = ls())` supprime toutes les assignations qui ont été réalisées auparavant.

```
rm(vec1)
```

1.2.3 Utiliser des fonctions

Dans les exemples de code précédents, nous avons utilisé plusieurs fonctions : la fonction `sqrt()`, la fonction `c()`, la fonction `data.frame()`, et la fonction `rm()`. Dans la suite de ce document, nous serons amenés à voir comment l'on crée une fonction et comment l'on arrive finalement à n'avoir qu'une expression suivie de parenthèses à utiliser pour faire un ensemble d'actions automatiquement. Mais avant cela, il est important de savoir globalement comment une fonction s'utilise. Cela est important car il est rapidement possible de se rendre compte qu'utiliser R, c'est utiliser des fonctions. De plus, lorsque l'on souhaite réaliser une nouvelle analyse avec une fonction que l'on n'a jamais utilisée auparavant, il est nécessaire de pouvoir en comprendre la structure et d'être en mesure d'en comprendre le fonctionnement pour pouvoir l'utiliser.

Pour expliquer comment s'utilise une fonction, commençons directement par un exemple, cette fois avec la fonction `plot()` :

```
plot(x = iris$Sepal.Length, y = iris$Petal.Length)
```

Comme nous pouvons l'observer ci-dessus, pour utiliser une fonction, il faut d'abord écrire son nom, puis mettre des parenthèses pour qu'on puisse écrire des informations à l'intérieur. Ces informations, elles sont de deux natures. D'un côté il y a les **arguments** (qui sont `x` et `y` dans l'exemple ci-dessus), et d'un autre côté il y a les **valeurs** (qui sont les variables `Sepal.Length` et `Petal.Length` du jeu de données `iris` dans l'exemple ci-dessus). Notons ici que le concept de valeur est à prendre au sens général du terme. Dans ce cadre là, une valeur pourrait tout aussi bien désigner des nombres ou des lettres, des vecteurs, des jeux de données, etc. Avec certaines fonctions, nous aurions pu mettre le nom de la variable seul et mettre le nom du jeu de données en face d'un autre argument, mais cela n'était pas possible ici. Comme nous pouvons le voir également, l'argument et la valeur sont toujours mis en lien par le biais du signe `=`. (Attention : Nous verrons plus tard qu'avec certaines fonctions, l'écriture qui est à gauche du signe `=` est en fait le nom d'une nouvelle variable à créer, mais laissons cela de côté pour le moment.) Le nombre d'arguments dépend des fonctions. Certaines n'en n'ont qu'un, d'autres peuvent en avoir un très grand nombre. Dans une fonction, certains arguments doivent obligatoirement recevoir une valeur indiquée par nos soins, alors que d'autres arguments ne seront tout simplement pas utilisés si on ne leur associe pas une valeur particulière. Enfin, certains arguments prendront une valeur par défaut associée à la fonction si l'on écrit rien les concernant dans la fonction. Dans la fonction `plot()`, seul l'argument `x` doit obligatoirement recevoir une valeur en principe pour que la fonction puisse être utilisée. Dans notre exemple, le fait d'avoir en plus associer une valeur à l'argument `y` permet à la fonction de non pas montrer uniquement les données de la variable `x`, mais de réaliser un graphique en montrant les données de `y` en fonction de `x`, comme ci-dessous :

```
plot(x = iris$Sepal.Length, y = iris$Petal.Length)
```

Lorsque les arguments sont explictement précisés comme pour `x` et `y` de l'exemple ci-dessus, il est en réalité possible de les écrire dans l'ordre que l'on veut. Nous aurions par exemple très bien pu écrire les choses de la manière suivante sans que cela ne change rien au résultat de la commande :

```
plot(y = iris$Petal.Length, x = iris$Sepal.Length)
```

Ce changement d'ordre n'est possible que lorsque les arguments sont explictement précisés, car oui, il est possible de configurer une fonction sans avoir à écrire le nom des arguments. C'est possible de le faire car une fonction est conçue de telle sorte à avoir un ordre par défaut des arguments. Lorsque les noms des arguments ne sont pas précisés, R associe les valeurs mises entre parenthèses aux arguments de la fonction en suivant l'ordre par défaut des arguments qui

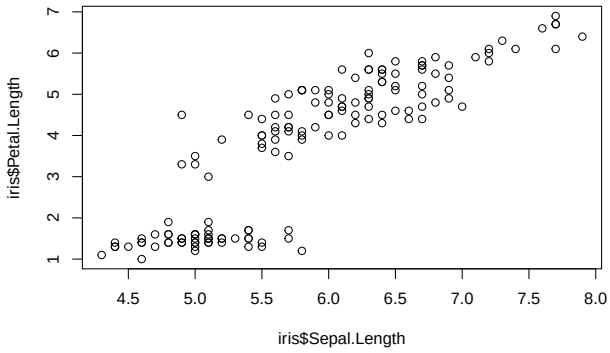


Figure 1.1: Graphique obtenu avec un paramétrage explicite de la fonction 'plot()'

est propre à la fonction. Ainsi, si nous voulons avoir, pour le graphique associé à notre exemple, la variable `Sepal.Length` en `x`, et la variable `Petal.Length` en `y`, nous aurions très bien pu écrire la fonction comme ceci :

```
plot(iris$Sepal.Length, iris$Petal.Length)
```

En revanche, si nous avions inversé l'ordre des variables sans préciser les noms des arguments, nous aurions eu un résultat différent du graphique précédent (dans l'exemple ci-dessous, les variables en x et en y ont été inversées par rapport au graphique précédent) :

```
plot(iris$Petal.Length, iris$Sepal.Length)
```

Dans la suite de ce document, les arguments configurés par nos soins seront en général précisés lorsque nous leur donnerons une valeur. Cela étant dit, beaucoup d'arguments seront aussi régulièrement laissés de côté lorsque cela ne sera pas nécessaire de les préciser pour l'exemple.

Au regard de ce qu'il vient d'être expliqué, il est donc une bonne pratique, avant d'utiliser une fonction, de connaître les arguments qu'elle contient, non seulement pour savoir comment configurer ses arguments, mais aussi pour savoir ce que font les arguments de la fonction lorsque l'on ne touche pas à leur configuration par défaut. En général, toute fonction utilisable et opérationnelle dans R dispose d'une aide consultable directement dans RStudio. Pour consulter l'aide associée à une fonction, il suffit d'écrire dans la Console le signe `?` suivi du nom de la fonction qui pose question, comme ci-dessous :

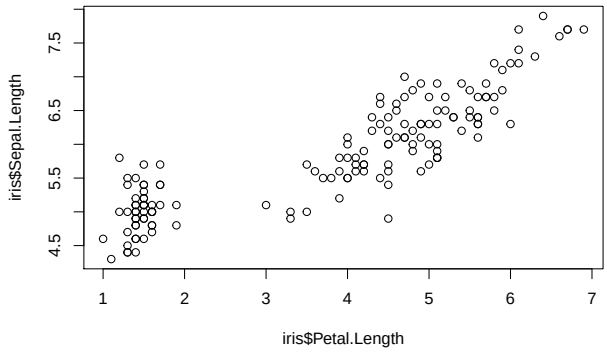


Figure 1.2: Graphique obtenu avec un paramétrage implicite de la fonction ‘plot()‘

```
?plot
```

Toutefois, paradoxalement, l'aide n'est pas toujours facile à comprendre pour ceux qui n'ont pas un niveau d'expertise avancé avec R. Pour bien comprendre comment on peut utiliser une fonction, internet peut être une excellente ressource car il regorge de sites et d'exemples crées par la communauté R. L'un des sites sur lesquels on tombe souvent lors d'une recherche internet est le site https://stackoverflow.com. Une bonne partie des problèmes de compréhension et d'utilisation des fonctions de R que l'on rencontre peuvent être résolus en consultant des exemples venant de ce site.

1.2.4 Installer et charger des packages

Le logiciel R fonctionne en bonne partie sur la base de fonctions qui permettent de réaliser automatiquement différents types de calculs. Ces fonctions sont regroupées dans des ensembles qu'on appelle des packages. La version d'installation du logiciel R dispose d'un ensemble de packages de base qui permettent de réaliser un très grand nombre d'analyses. Toutefois, la version de base de R impose parfois des manières d'écrire certaines instructions qui sont peu intuitives ou qui parfois ne permettent tout simplement pas de faire les analyses souhaitées. Pour palier ces problèmes, des packages sont régulièrement crées et actualisés par la communauté R. Pour pouvoir les utiliser, il est nécessaire de d'abord installer le package additionnel grâce à la fonction `install.packages()`. L'une des collections de packages les plus utiles pour manipuler des tableaux de données et effectuer des analyses statistiques et

graphiques est celle du `tidyverse`, qui a été pensée notamment pour faciliter l'écriture des lignes de code.

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

Une fois que le package a été installé (ou l'ensemble de packages s'il s'agit d'une collection comme dans le cas du `tidyverse`), une étape supplémentaire est nécessaire pour pouvoir utiliser les fonctions qu'il contient : il faut le charger dans l'environnement R. Pour cela, il est possible d'utiliser la fonction `library()`.

```
library(tidyverse)
```

Lorsque l'on charge la collection de packages `tidyverse`, on peut observer dans la Console que plusieurs packages sont chargés en même temps : `ggplot2`, pour la visualisation de données ; `dplyr`, pour la manipulation de données ; `tidyr`, pour l'organiation des tableaux de données ; `readr`, pour l'importation de jeux de données ; `purrr`, pour la programmation ; `tibble`, pour le formatage de tableaux de données ; `stringr`, pour la gestion des chaînes de caractères ; `forcats`, pour la gestion de variables qualitatives. Si nous avons téléchargé et chargé l'ensemble des packages du `tidyverse`, nous aurions pu aussi installé et chargé un seul de ces packages à la fois, comme pour la plupart des packages qui existent.

1.2.5 Divers

Au fur et à mesure que l'on écrit un script, une bonne pratique consiste à régulièrement créer des sections avec des titres et d'ajouter des commentaires pour certaines analyses. Afin de ne pas rendre activable les lignes de code qui ne serviraient qu'à écrire des titres ou des commentaires, il convient d'utiliser le symbole `#` devant l'écriture du code.

```
### Titre de section 1 ----------------------------------------------------------------
## Sous-titre 1
  # Commentaire 1
```

Étant donné que l'erreur est difficile à éviter à un moment donné ou à un autre lorsqu'on commence à écrire son propre code ou à utiliser un code qui vient de quelqu'un d'autre, il est utile de reconnaître les situations dans lesquelles une erreur est survenue. La situation la plus évidente est l'apparition d'un texte en rouge dans la Console. Lorsqu'il s'agit bel et bien d'une erreur (car il peut ne s'agir parfois que d'un message d'alerte ou d'information), le texte en rouge décrit l'erreur qui a été détectée et qui empêche le code d'être entièrement activé. De manière moins visible, il est possible parfois d'observer un `+` tout en bas de

la console. Cela survient lorsque le code lancé à l'instant est incomplet (e.g., une parenthèse a été oubliée). Si cela arrive, il vaut mieux appuyer sur **Echap**, trouver l'erreur dans le code, et relancer la commande. Avant de relancer la commande, il faut s'assurer que R donne effectivement la main pour lancer une nouvelle instruction. C'est le cas lorsque le symbole `>` est observé dans la Console. Enfin, RStudio permet d'utiliser un certain nombre de raccourcis clavier. Pour en avoir une vue d'ensemble, appuyer sur **Alt+Shift+K**.

1.3 Résumé

- R permet de faire des opérations sur des valeurs : additions, soustractions, multiplications, divisions, etc.
- R permet de faire des opérations sur des objets : valeurs uniques, vecteurs, tableaux de données, etc.
- Pour créer un vecteur, qu'il contienne des nombres, du texte, ou les deux, il est possible d'utiliser la fonction `c()`.
- Pour créer un tableau de données, il est possible d'utiliser la fonction `data.frame()`.
- Pour afficher une colonne particulière d'un tableau de données, utiliser le symbole `$`.
- Pour assigner un objet à un nom, utiliser la fonction d'assignation `<-`.
- Pour supprimer une assignation, utiliser la fonction `rm()`.
- L'utilisation de R repose sur l'utilisation de fonctions. Une fonction s'utilise en écrivant son nom, suivi de parenthèses à l'intérieur desquelles on peut préciser les arguments qui nous intéressent et indiquer les valeurs nécessaires selon les besoins des analyses.
- Pour demander l'aide de R à propos d'une fonction, écrire `?` suivi du nom de la fonction.
- Pour installer un package, utiliser la fonction `install.packages()`.
- Un code peut être écrit directement dans la **Console**, ou dans une fenêtre **Script**. Pour activer le code à partir d'un script, utiliser le bouton **Run** ou **Ctrl + Entrée**. Les noms et les objets associés apparaissent dans la fenêtre **Environnement**.
- Pour écrire des titres et des commentaires dans un script, utiliser un `#` avant l'écriture du code.
- Dans la Console, le symbole `>` signifie que le logiciel est prêt à lancer une nouvelle instruction. Le symbole `+` indique que l'instruction initialement lancée est incomplète. Mieux vaut alors faire **Echap**, modifier le code, et recommencer.

Chapter 2

Importation et manipulation d'une base de données

2.1 Comprendre ce qu'est une base de données

Lorsqu'on souhaite répondre à une question, la démarche scientifique classique consiste à effectuer une série de mesures ou d'observations selon un protocole qui a été conçu en cohérence avec la question posée. En principe, ces mesures ou observations donnent lieu à l'obtention de valeurs. Ces valeurs peuvent être de forme numérique (e.g., les valeurs de taille de différents individus) ou de forme littérale (e.g., les valeurs de sexe de différents individus). Quelle que soit leur forme, les valeurs que l'on obtient dans un contexte qui est connu, comme dans le cas d'un protocole de mesures, ont un sens bien défini car elles sont associées à des choses que l'on a cherché à caractériser. Lorsqu'une valeur est porteuse d'un sens bien défini, on peut alors considérer qu'il s'agit d'une **donnée**. Très souvent, pour répondre à une question, il est nécessaire d'acquérir plusieurs données qui seraient relatives à différentes choses que l'on a cherché à caractériser (e.g., la taille, la couleur, le poids, etc.), et qui seraient relatives également à différents individus chez qui l'on aurait souhaité caractériser ces choses. Afin de conduire les analyses qui permettraient de répondre à la question posée, il convient alors de répertorier toutes les données acquises dans un même document, et plus exactement dans un même fichier, qui serait la base de données, telle que présentée ci-dessous.

Table 2.1: Exemple de base de données

id	genre	taille	nb_victoires	niveau

1	H	1.80	45	1
2	H	1.93	90	3
3	H	1.50	100	4
4	F	1.95	43	1
5	F	1.52	34	2
6	H	1.87	67	2
7	H	1.83	79	3

La base de données prend donc la forme d'un tableau. Plusieurs principes sont à respecter en général lors de la création d'une base de données. Tout d'abord, les lignes de la base de données (qu'on appelle des **observations**) doivent correspondre le cas échéant à des individus bien identifiés. Ensuite, chaque colonne doit correspondre à une variable. L'ensemble des données contenues dans une même ligne correspond donc aux données relatives aux différentes variables (e.g., la taille, le poids, le sexe, etc.) qui auraient été obtenues chez un même individu. Dans le cas d'études où l'on évaluerait une ou plusieurs variables plusieurs fois chez un même individu (i.e., à différentes moments, dans différentes conditions), il peut convenir de créer autant de lignes que de fois où les variables auraient été évaluées. Par exemple, le tableau ci-dessous représente une base de données (certes très sommaire) contenant des données d'individus dont on aurait évalué le poids deux fois, avant et après un programme de prise en charge. On remarque alors qu'il y a deux lignes par individu qui correspondent aux deux temps d'évaluation. La taille, elle, n'aura été évaluée qu'une seule fois, en début de programme, mais pour éviter de laisser des cellules vides (ce qui est une bonne pratique), alors la valeur aura été reproduite dans la seconde ligne.

Table 2.2: Organisation d'une base de données avec des mesures répétées

id	taille	temps_eval	poids
1	1.75	pre	75
1	1.75	post	73
2	1.89	pre	90
2	1.89	post	88

En principe, les données de la base qui ont été obtenues selon la même procédure d'acquisition représentent le même type de choses. Ces choses sont appelées des **variables** car elles varient selon les individus qui ont été étudiés et les conditions de mesure qui ont été mises en oeuvre (dans le cas où il y en aurait plusieurs). Lorsque ces choses ne sont pas censées varier, on parle de **constantes**. Une base de données peut comporter plusieurs variables de natures différentes :

- Des **variables quantitatives**, qui comportent des nombres. Parmi les variables quantitatives, on distingue celles qui sont continues et celles

qui sont discrètes. Les variables quantitatives continues contiennent des données pouvant comporter théoriquement un nombre infini de décimales (e.g., la taille, le poids, etc.). Au contraire, les variables quantitatives discrètes ne peuvent contenir théoriquement que des nombres finis (e.g., le nombre de victoires sportives au cours d'une année). Certaines variables en théorie discrètes sont cependant souvent considérées comme continues tant le nombre de valeurs théoriquement possibles pour la variable est grand, tel que pour le nombre de globules blancs mesurés dans le sang (Labreuche, 2010).

- Des **variables qualitatives**, qui contiennent des valeurs désignant non pas des quantités mais des modalités. Ces modalités peuvent être exprimées sous forme littérale ou numérique. Parmi les variables qualitatives, on distingue celles qui sont nominales et celles qui sont ordinales. Les variables qualitatives nominales contiennent des modalités qui ne peuvent pas être ordonnées (e.g., les couleurs, les genres, etc.). Au contraire, les variables qualitatives ordinales contiennent des modalités qui peuvent être ordonnées (e.g., les niveaux de compétition sportive : départemental ; régional ; interrégional ; national ; international). Les variables qualitatives ordinales qui prendraient des valeurs numériques pour indiquer par exemple un niveau d'expertise (1, 2, 3, 4) se différencient des variables quantitatives discrètes par l'absence d'information sur la distance qui sépare les nombres de cette variable (Labreuche, 2010).

2.2 Fixer le répertoire de travail

Lorsque l'on souhaite réaliser l'analyse d'une base de données avec RStudio, il peut être utile et plus fonctionnel pour la suite de créer un dossier spécifique, sur l'ordinateur, où se trouveraient à la fois la base de données ainsi que toutes les créations (e.g., tableaux, figures, etc.) qui seraient produites au fur et à mesure des analyses. Une fois le dossier crée, il faut ensuite, dans RStudio, ouvrir un fichier **Script** où toutes les commandes seront écrites et enregistrables (chemin d'accès : **File > New File > R Script**). Une fois ouvert, il est possible d'enregistrer le script en appuyant sur **Ctrl+S**, en privilégiant le dossier où se trouve la base de données comme emplacement. La prochaine manipulation à faire est alors de **Fixer le répertoire**. L'enjeu est ici d'indiquer à RStudio un dossier à privilégier pour réaliser des importations de fichiers dans RStudio ou des exports de fichiers à partir de RStudio. Le dossier à considérer pour cela pourrait donc être celui où se trouve déjà la base de données. Plus concrètement, après avoir fait cela, dès lors que l'on voudra importer un tableau de données dans RStudio, au lieu de préciser dans la fonction d'importation tout le chemin d'accès définissant l'emplacement du fichier sur l'ordinateur, il suffira dans la fonction de mettre seulement le nom du fichier à importer. De plus, dès lors que l'on voudra exporter un tableau ou une figure crée(e) avec RStudio, l'export se fera automatiquement vers le dossier de travail en cours si rien n'est

spécifié dans la fonction d'export quant à l'emplacement du document. Pour fixer le répertoire de travail, il existe plusieurs manières de faire. Une manière relativement simple de procéder est la suivante. Suivre le chemin d'accès suivant à partir de RStudio : **Session > Set Working Directory > Choose Directory...**. (Ou appuyer sur **Ctr+Shift+H**.). Puis, sélectionner le dossier où se trouve la base de données. Enfin, cliquer sur **Open**. Une commmande apparaît alors dans la Console. Le répertoire vient donc d'être fixé. Pour ne pas devoir faire la même manipulation à chaque nouvelle ouverture de RStudio, il est possible de copier cette commande qui est apparue dans la Console (ne pas copier le symbole `>`) et de la coller au début du script. On pourra noter que cette commande contient la fonction `setwd()`, et qu'à l'intérieur se trouve le chemin d'accès au répertoire de travail qui est écrit entre des guillemets.

2.3 Importer la base de données

Il existe plusieurs fonctions pour importer une base de données dans RStudio. La fonction `read_csv2()` du package `readr` permet d'importer par exemple des fichiers .csv qui, structurellement, séparent les données avec des points-virgules. C'est généralement le type de structure de fichier .csv que l'on obtient après avoir réalisé un export à partir d'un fichier Excel. Pour illustrer ici l'importation d'une base de données, il est possible d'en créer une dans le répertoire de travail actif en y exportant un tableau de données qui existe déjà avec le logiciel R. Le logiciel R dispose en effet d'un grand nombre de jeux de données différents que tout utilisateur peut consulter et manipuler. L'ensemble des jeux de données disponibles suite à l'installation par défaut de R est visible en lançant dans la Console la commande `data()`. Au fur et à mesure de la découverte des analyses montrées dans ce document, différents jeux de données seront utilisés en fonction des besoins. Pour le moment, il est possible d'utiliser le jeu de données qui s'appelle `iris`. Même si on ne le voit pas dans la fenêtre Environnement de RStudio, il est bel et bien là, disponible, prêt à être utilisé. Afin d'exporter ce jeu de données dans le répertoire de travail fixé au préalable, il est possible d'utiliser la fonction `write_csv2()` du package `readr`. Pour cela, il suffit d'utiliser le nom du jeu de données, puis d'indiquer entre guillemets le nom que l'on veut que le fichier exporté ait, tout en n'oubliant pas de mettre l'extension .csv à la fin du nom pour indiquer le format d'export, comme ci-dessous.

```
write_csv2(x = iris, path = "iris.csv")
```

Si l'on jette un oeil dans le répertoire de travail, il est alors possible d'y voir un nouveau fichier .csv du nom de `iris`. Maintenant qu'il existe une base de données dans le répertoire de travail actif, il est possible de concrétiser la procédure de son importation dans RStudio. Comme évoqué plus tôt dans ce document, il est intéressant, et en réalité nécessaire, d'assigner cette base de données à un nom pour pouvoir plus facilement manipuler le jeu de données

par la suite. Ici, nous allons tout naturellement associer ce nouvel objet au nom `iris`.

```
iris <- read_csv2(file = "iris.csv")
```

Suite à l'activation de la commande, RStudio nous montre un message d'information sur la manière dont la fonction `read_csv2()` a configuré le jeu de données importé. Ce message apparaît car la fonction importe le jeu de données non pas sous la forme d'un *data frame* comme nous avons pu en créer auparavant, mais sous la forme d'un *tibble*, qui désigne un format de tableau que l'on ne peut obtenir qu'en passant par le biais de fonctions associées à l'ensemble de packages `tidyverse`. Pour comprendre l'intérêt d'un *tibble*, revenons au format classique d'un *data frame* à l'aide de la fonction `as.data.frame()`.

```
iris <- as.data.frame(x = iris)
```

À présent, regardons ce qu'il se passe si on lance le nom `iris` dans la Console...

```
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     setosa
## 12           4.8         3.4          1.6         0.2     setosa
## 13           4.8         3.0          1.4         0.1     setosa
## 14           4.3         3.0          1.1         0.1     setosa
## 15           5.8         4.0          1.2         0.2     setosa
## 16           5.7         4.4          1.5         0.4     setosa
## 17           5.4         3.9          1.3         0.4     setosa
## 18           5.1         3.5          1.4         0.3     setosa
## 19           5.7         3.8          1.7         0.3     setosa
## 20           5.1         3.8          1.5         0.3     setosa
## 21           5.4         3.4          1.7         0.2     setosa
## 22           5.1         3.7          1.5         0.4     setosa
## 23           4.6         3.6          1.0         0.2     setosa
```

```
## 24          5.1         3.3          1.7         0.5     setosa
## 25          4.8         3.4          1.9         0.2     setosa
## 26          5.0         3.0          1.6         0.2     setosa
## 27          5.0         3.4          1.6         0.4     setosa
## 28          5.2         3.5          1.5         0.2     setosa
## 29          5.2         3.4          1.4         0.2     setosa
## 30          4.7         3.2          1.6         0.2     setosa
## 31          4.8         3.1          1.6         0.2     setosa
## 32          5.4         3.4          1.5         0.4     setosa
## 33          5.2         4.1          1.5         0.1     setosa
## 34          5.5         4.2          1.4         0.2     setosa
## 35          4.9         3.1          1.5         0.2     setosa
## 36          5.0         3.2          1.2         0.2     setosa
## 37          5.5         3.5          1.3         0.2     setosa
## 38          4.9         3.6          1.4         0.1     setosa
## 39          4.4         3.0          1.3         0.2     setosa
## 40          5.1         3.4          1.5         0.2     setosa
## 41          5.0         3.5          1.3         0.3     setosa
## 42          4.5         2.3          1.3         0.3     setosa
## 43          4.4         3.2          1.3         0.2     setosa
## 44          5.0         3.5          1.6         0.6     setosa
## 45          5.1         3.8          1.9         0.4     setosa
## 46          4.8         3.0          1.4         0.3     setosa
## 47          5.1         3.8          1.6         0.2     setosa
## 48          4.6         3.2          1.4         0.2     setosa
## 49          5.3         3.7          1.5         0.2     setosa
## 50          5.0         3.3          1.4         0.2     setosa
## 51          7.0         3.2          4.7         1.4 versicolor
## 52          6.4         3.2          4.5         1.5 versicolor
## 53          6.9         3.1          4.9         1.5 versicolor
## 54          5.5         2.3          4.0         1.3 versicolor
## 55          6.5         2.8          4.6         1.5 versicolor
## 56          5.7         2.8          4.5         1.3 versicolor
## 57          6.3         3.3          4.7         1.6 versicolor
## 58          4.9         2.4          3.3         1.0 versicolor
## 59          6.6         2.9          4.6         1.3 versicolor
## 60          5.2         2.7          3.9         1.4 versicolor
## 61          5.0         2.0          3.5         1.0 versicolor
## 62          5.9         3.0          4.2         1.5 versicolor
## 63          6.0         2.2          4.0         1.0 versicolor
## 64          6.1         2.9          4.7         1.4 versicolor
## 65          5.6         2.9          3.6         1.3 versicolor
## 66          6.7         3.1          4.4         1.4 versicolor
## 67          5.6         3.0          4.5         1.5 versicolor
## 68          5.8         2.7          4.1         1.0 versicolor
## 69          6.2         2.2          4.5         1.5 versicolor
```

```
## 70          5.6         2.5          3.9         1.1 versicolor
## 71          5.9         3.2          4.8         1.8 versicolor
## 72          6.1         2.8          4.0         1.3 versicolor
## 73          6.3         2.5          4.9         1.5 versicolor
## 74          6.1         2.8          4.7         1.2 versicolor
## 75          6.4         2.9          4.3         1.3 versicolor
## 76          6.6         3.0          4.4         1.4 versicolor
## 77          6.8         2.8          4.8         1.4 versicolor
## 78          6.7         3.0          5.0         1.7 versicolor
## 79          6.0         2.9          4.5         1.5 versicolor
## 80          5.7         2.6          3.5         1.0 versicolor
## 81          5.5         2.4          3.8         1.1 versicolor
## 82          5.5         2.4          3.7         1.0 versicolor
## 83          5.8         2.7          3.9         1.2 versicolor
## 84          6.0         2.7          5.1         1.6 versicolor
## 85          5.4         3.0          4.5         1.5 versicolor
## 86          6.0         3.4          4.5         1.6 versicolor
## 87          6.7         3.1          4.7         1.5 versicolor
## 88          6.3         2.3          4.4         1.3 versicolor
## 89          5.6         3.0          4.1         1.3 versicolor
## 90          5.5         2.5          4.0         1.3 versicolor
## 91          5.5         2.6          4.4         1.2 versicolor
## 92          6.1         3.0          4.6         1.4 versicolor
## 93          5.8         2.6          4.0         1.2 versicolor
## 94          5.0         2.3          3.3         1.0 versicolor
## 95          5.6         2.7          4.2         1.3 versicolor
## 96          5.7         3.0          4.2         1.2 versicolor
## 97          5.7         2.9          4.2         1.3 versicolor
## 98          6.2         2.9          4.3         1.3 versicolor
## 99          5.1         2.5          3.0         1.1 versicolor
## 100         5.7         2.8          4.1         1.3 versicolor
## 101         6.3         3.3          6.0         2.5  virginica
## 102         5.8         2.7          5.1         1.9  virginica
## 103         7.1         3.0          5.9         2.1  virginica
## 104         6.3         2.9          5.6         1.8  virginica
## 105         6.5         3.0          5.8         2.2  virginica
## 106         7.6         3.0          6.6         2.1  virginica
## 107         4.9         2.5          4.5         1.7  virginica
## 108         7.3         2.9          6.3         1.8  virginica
## 109         6.7         2.5          5.8         1.8  virginica
## 110         7.2         3.6          6.1         2.5  virginica
## 111         6.5         3.2          5.1         2.0  virginica
## 112         6.4         2.7          5.3         1.9  virginica
## 113         6.8         3.0          5.5         2.1  virginica
## 114         5.7         2.5          5.0         2.0  virginica
## 115         5.8         2.8          5.1         2.4  virginica
```

```
## 116          6.4         3.2          5.3         2.3  virginica
## 117          6.5         3.0          5.5         1.8  virginica
## 118          7.7         3.8          6.7         2.2  virginica
## 119          7.7         2.6          6.9         2.3  virginica
## 120          6.0         2.2          5.0         1.5  virginica
## 121          6.9         3.2          5.7         2.3  virginica
## 122          5.6         2.8          4.9         2.0  virginica
## 123          7.7         2.8          6.7         2.0  virginica
## 124          6.3         2.7          4.9         1.8  virginica
## 125          6.7         3.3          5.7         2.1  virginica
## 126          7.2         3.2          6.0         1.8  virginica
## 127          6.2         2.8          4.8         1.8  virginica
## 128          6.1         3.0          4.9         1.8  virginica
## 129          6.4         2.8          5.6         2.1  virginica
## 130          7.2         3.0          5.8         1.6  virginica
## 131          7.4         2.8          6.1         1.9  virginica
## 132          7.9         3.8          6.4         2.0  virginica
## 133          6.4         2.8          5.6         2.2  virginica
## 134          6.3         2.8          5.1         1.5  virginica
## 135          6.1         2.6          5.6         1.4  virginica
## 136          7.7         3.0          6.1         2.3  virginica
## 137          6.3         3.4          5.6         2.4  virginica
## 138          6.4         3.1          5.5         1.8  virginica
## 139          6.0         3.0          4.8         1.8  virginica
## 140          6.9         3.1          5.4         2.1  virginica
## 141          6.7         3.1          5.6         2.4  virginica
## 142          6.9         3.1          5.1         2.3  virginica
## 143          5.8         2.7          5.1         1.9  virginica
## 144          6.8         3.2          5.9         2.3  virginica
## 145          6.7         3.3          5.7         2.5  virginica
## 146          6.7         3.0          5.2         2.3  virginica
## 147          6.3         2.5          5.0         1.9  virginica
## 148          6.5         3.0          5.2         2.0  virginica
## 149          6.2         3.4          5.4         2.3  virginica
## 150          5.9         3.0          5.1         1.8  virginica
```

RStudio nous montre tout le jeu de données dans la Console, ce qui n'est pas très utile, d'autant plus que l'on peut perdre de vue la première ligne de titre lorsque le jeu de données contient beaucoup de lignes. Retournons donc au format *tibble* grâce à la fonction `as_tibble()` du package `tibble`, et voyons ce qu'il se passe lorsqu'on lance à nouveau le nom `iris` dans la Console.

```
iris <- as_tibble(x = iris)
iris
```

```
## # A tibble: 150 x 5
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <chr>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            1.4         0.2 setosa
##  3          4.7         3.2          1.3         0.2 setosa
##  4          4.6         3.1          1.5         0.2 setosa
##  5          5           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
## # ... with 140 more rows
```

Cette fois, RStudio n'affiche que les premières lignes du jeu de données, et il fournit en plus de cela des informations quant aux types de variables présentes dans le jeu de données, en-dessous de la ligne de titres. Maintenant que la base de données a été importée, il ne reste plus qu'à voir différentes fonctions pour pouvoir configurer la base de données telle que nous la voulons pour réaliser confortablement les analyses.

2.4 Manipuler la base de données

2.4.1 Vérifier le succès de l'importation de la base

Avant de débuter les analyses de la base de données, une bonne pratique est de vérifier si la base de données a été correctement importée avec RStudio. Une manière rapide de faire cela est de regarder le nombre d'observations (i.e., de lignes) et de variables (i.e., de colonnes) associés à l'objet crée lors de l'importation et visible dans la fenêtre Environnement de RStudio, puis de cliquer sur le nom associé à l'objet. Lors de l'étape précédente, nous avons importé le jeu de données `iris` en l'appelant ainsi. Lorsque l'on cherche le nom `iris` dans la fenêtre Environnement, on peut voir que l'objet associé contient 150 observations et 5 variables, signes que la structure du jeu de données a été bien interprétée par R si l'on sait que ce sont effectivement les dimensions du jeu de données en question. Puis, lorsque l'on clique sur le nom `iris` dans la liste des noms montrés dans la fenêtre Environnement, RStudio ouvre un onglet qui contient les données. Il est alors possible de voir d'un simple coup d'oeil si les données sont bien présentes et organisées en lignes et en colonnes comme attendu.

2.4.2 Vérifier et reconfigurer les types des variables de la base

Il convient de vérifier que les types des variables que RStudio a associés aux variables du jeu de données importé soient bien en accord avec ce qui était attendu. Pour vérifier les types des variables, il est possible d'utiliser la fonction `str()` avec le nom auquel on a associé la base de données.

```
str(iris)
```

```
## Classes 'tbl_df', 'tbl' and '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     : chr  "setosa" "setosa" "setosa" "setosa" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Sepal.Length = col_double(),
##   ..   Sepal.Width = col_double(),
##   ..   Petal.Length = col_double(),
##   ..   Petal.Width = col_double(),
##   ..   Species = col_character()
##   .. )
```

Après avoir activé la commande contenant la fonction `str()`, la Console affiche plusieurs lignes d'information, avec à chaque fois le nom de la variable, son type, et les premières valeurs de la variable. Plusieurs termes peuvent être rencontrés selon la manière dont R a interprété les variables du jeu de données, notamment (mais pas seulement) :

- num : désigne une variable quantitative continue.
- int : dégine une variable quantitative discrète (avec des nombres entiers).
- Factor : désigne une variable qualitative nominale.
- chr : désigne une variable texte.
- Date : désigne une variable date.

Le logiciel R s'appuie donc sur une classification des types de variables plus complexe que celle que nous avons présentée précédemment. On peut noter que les abbréviations montrées pour indiquer le type de variable en utilisant la fonction `str()` sont différentes de celles montrées lorsque l'on observe un jeu de données au format *tibble* dans la Console, mais ces différences reflètent en réalité principalement une divergence dans les stratégies d'écriture de l'information par les concepteurs des packages et des fonctions. En outre, si l'on veut déterminer

le type d'une seule variable, ou plus globalement le type de l'objet qui nous intéresse, on peut utiliser la fonction `class()`. Utiliser un nom de variable avec cette fonction renverra le type de la variable, comme montré ci-dessous.

```
class(x = iris$Sepal.Length)
```

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

Lorsque le type de variable ne correspond pas à celui attendu, il peut être utile de se questionner sur les raisons de la mauvaise interprétation que R en a faite. L'une des erreurs qui peuvent régulièrement survenir est le fait d'obtenir une variable de type *Factor* ou de type *chr* au lieu d'une variable de type *num* lorsque l'importation du jeu de données a été réalisée avec une fonction d'importation mal configurée (e.g., il est possible que la fonction d'importation du jeu de données considérait les valeurs comme des nombres décimaux lorsqu'ils avaient des points (.) alors qu'en réalité les nombres décimaux étaient écrits avec des virgules (,) dans le jeu de données. Une autre possibilité est que l'on n'ait pas indiqué dans la fonction d'importation sous quelle forme se présentaient les valeurs manquantes de la base de données (e.g., s'il y a des valeurs manquantes notées "NA" dans la base de données, mais que cela n'est pas indiqué dans la fonction d'importation, R peut interpréter les variables concernées comme des variables *chr*). En utilisant la fonction `read_csv2()` du package `readr`, ces écueils sont plus facilement évités car les paramètres par défaut de la fonction nous facilitent le travail. En revanche, d'autres fonctions, plus anciennes, comme `read.csv2()` qui est une fonction de base de R, nécessitent plus de vigilance.

Lorsque la modification du type de la variable est nécessaire, une stratégie possible est de créer une variable portant exactement le même nom à partir de la variable initiale et à laquelle on applique une fonction capable d'imposer un certain type de variable. Il existe une fonction pour chaque type de variable à définir, notamment (mais pas seulement) :

- La fonction `as.numeric()` pour obtenir un type de variable quantitative.
- La fonction `as.factor()` pour un obtenir un type de variable qualitative.
- La fonction `as.character()` pour un obtenir un type de variable texte.
- La fonction `as.Date()` pour obtenir un type de variable date.

Par exemple, nous aurions pu vouloir faire en sorte que toutes les variables du jeu de données `iris` soient de type texte :

```
iris$Sepal.Length <- as.character(x = iris$Sepal.Length)
iris$Sepal.Width <- as.character(x = iris$Sepal.Width)
iris$Petal.Length <- as.character(x = iris$Petal.Length)
iris$Petal.Width <- as.character(x = iris$Petal.Width)
iris$Species <- as.character(x = iris$Species)
```

Remarquons qu'à chaque fois, la variable désignée à gauche de la flèche d'assignation est exactement la même que celle qui est indiquée à droite de la flèche d'assignation dans la fonction `as.character()`, ce qui implique que la création de la nouvelle variable entraîne la suppression et le remplacement de la précédente qui portait le même nom. Il est possible de vérifier la conséquence de ces commandes avec la fonction `str()`.

```
str(iris)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: chr  "5.1" "4.9" "4.7" "4.6" ...
##  $ Sepal.Width : chr  "3.5" "3" "3.2" "3.1" ...
##  $ Petal.Length: chr  "1.4" "1.4" "1.3" "1.5" ...
##  $ Petal.Width : chr  "0.2" "0.2" "0.2" "0.2" ...
##  $ Species     : chr  "setosa" "setosa" "setosa" "setosa" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Sepal.Length = col_double(),
##   ..   Sepal.Width = col_double(),
##   ..   Petal.Length = col_double(),
##   ..   Petal.Width = col_double(),
##   ..   Species = col_character()
##   .. )
```

Cette stratégie de modification du type de la variable peut convenir lorsqu'il y a peu de variables à modifier. Cependant, lorsque la liste s'allonge, il peut être plus lisible, en matière de code, de fonctionner autrement : en utilisant le symbole `%>%` (qu'on appelle *pipe*), et la fonction `mutate()` du package `dplyr`.

```
iris <-
  iris %>%
  mutate(Sepal.Length = as.numeric(x = Sepal.Length),
         Sepal.Width = as.numeric(x = Sepal.Width),
         Petal.Length = as.numeric(x = Petal.Length),
         Petal.Width = as.numeric(x = Petal.Width),
         Species = as.factor(x = Species))
```

Ici, le symbole `%>%` permet d'indiquer à R que toutes les fonctions qui sont écrites après ce symbole s'appliquent à ce qui a été défini avant ce symbole. La fonction `mutate()`, dont nous reparlerons peu après, permet de créer de nouvelles variables dans le cadre de cette stratégie, soit en écrasant les anciennes variables si les anciens noms sont conservés, soit en créant de nouvelles variables si de nouveaux noms sont utilisés. Remarquons également qu'avec ce code, nous venons de créer un nouvel objet (en l'assignant à nouveau au nom`iris`) à partir de l'ancien objet, mais dont on a transformé les types des variables, perdant dans le même temps l'ancien objet.

2.4.3 Sélectionner des variables avec select()

Certains jeux de données peuvent être très larges, c'est-à-dire qu'ils peuvent contenir beaucoup de colonnes, parfois inutiles, et qui peuvent être gênantes lorsque l'on veut avoir une vue claire du contenu du jeu de données. La fonction `select()` du package `dplyr` permet de sélectionner des colonnes facilement.

```
iris %>%
  select(Petal.Length, Petal.Width, Species)
```

```
## # A tibble: 150 x 3
##    Petal.Length Petal.Width Species
##           <dbl>       <dbl> <fct>
##  1          1.4         0.2 setosa
##  2          1.4         0.2 setosa
##  3          1.3         0.2 setosa
##  4          1.5         0.2 setosa
##  5          1.4         0.2 setosa
##  6          1.7         0.4 setosa
##  7          1.4         0.3 setosa
##  8          1.5         0.2 setosa
##  9          1.4         0.2 setosa
## 10          1.5         0.1 setosa
## # ... with 140 more rows
```

2.4.4 Renommer des variables avec rename()

Il est possible que certains titres de variables ne soient pas clairs ou trop longs, ce qui peut être gênant pour écrire un code le plus lisible possible. La fonction `rename()` du package `dplyr` permet de gérer cela. Dans l'exemple ci-dessous, on observe que le nouveau nom doit être écrit à gauche du signal `=`, alors que l'ancien nom doit être écrit à droite du signe `=`.

```
iris %>%
  rename(Sepal_long = Sepal.Length,
         Sepal_lar = Sepal.Width,
         Petal_long = Petal.Length,
         Petal_lar = Petal.Width,
         Especes = Species)
```

```
## # A tibble: 150 x 5
##    Sepal_long Sepal_lar Petal_long Petal_lar Especes
##         <dbl>     <dbl>      <dbl>     <dbl> <fct>
##  1        5.1       3.5        1.4       0.2 setosa
```

```
##  2        4.9       3         1.4       0.2 setosa
##  3        4.7       3.2       1.3       0.2 setosa
##  4        4.6       3.1       1.5       0.2 setosa
##  5        5         3.6       1.4       0.2 setosa
##  6        5.4       3.9       1.7       0.4 setosa
##  7        4.6       3.4       1.4       0.3 setosa
##  8        5         3.4       1.5       0.2 setosa
##  9        4.4       2.9       1.4       0.2 setosa
## 10        4.9       3.1       1.5       0.1 setosa
## # ... with 140 more rows
```

2.4.5 Créer des variables avec mutate()

Certaines analyses peuvent nécessiter d'ajouter des variables à partir de calculs réalisés sur des variables qui existent déjà dans le jeu de données. La fonction `mutate()`, du package `dplyr`, et que nous avons déjà rencontrée précédemment, permet cela. Dans l'exemple ci-dessous, on observe que le nom de la nouvelle variable à créer est à gauche du signe `=` et que le calcul créant les nouvelles valeurs est décrit à droite du signe `=`.

```
iris %>%
  mutate(ratio_sepal = Sepal.Length / Sepal.Width,
         ratio_petal = Petal.Length / Petal.Width)
```

```
## # A tibble: 150 x 7
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species ratio_sepal
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>         <dbl>
##  1          5.1         3.5          1.4         0.2 setosa         1.46
##  2          4.9         3            1.4         0.2 setosa         1.63
##  3          4.7         3.2          1.3         0.2 setosa         1.47
##  4          4.6         3.1          1.5         0.2 setosa         1.48
##  5          5           3.6          1.4         0.2 setosa         1.39
##  6          5.4         3.9          1.7         0.4 setosa         1.38
##  7          4.6         3.4          1.4         0.3 setosa         1.35
##  8          5           3.4          1.5         0.2 setosa         1.47
##  9          4.4         2.9          1.4         0.2 setosa         1.52
## 10          4.9         3.1          1.5         0.1 setosa         1.58
## # ... with 140 more rows, and 1 more variable: ratio_petal <dbl>
```

2.4.6 Sélectionner des lignes avec filter()

En fonction des besoins de l'analyse, on peut vouloir ne retenir que certaines lignes du fichier de données. La fonction `filter()` du package `dplyr` est faite

pour réaliser ce filtrage. Plusieurs opérateurs sont disponibles pour ne retenir que les lignes que l'on veut :

Table 2.3: Les opérateurs utilisables avec la fonction 'filter()'

Opération	Opérateur
Égal	==
Inférieur ou égal	<=
Supérieur ou égal	>=
Différent de	!=

De plus, dans la configuration du code, ces opérateurs peuvent être couplés à l'opérateur `|` (OU) et à l'opérateur `&` (ET). Dans l'exemple ci-dessous, le code permet, à partir du jeu de données `iris`, de ne garder que les lignes du jeu de données qui contiennent les noms d'espèce *setosa* OU *virginica*, ET en même temps qui affichent une longueur de sépale inférieure ou égale à 5.

```
iris %>%
  filter((Species == "setosa" | Species == "virginica") &
           Sepal.Length <= 5)
```

```
## # A tibble: 29 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1          4.9         3            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           3.6          1.4         0.2 setosa
##  5          4.6         3.4          1.4         0.3 setosa
##  6          5           3.4          1.5         0.2 setosa
##  7          4.4         2.9          1.4         0.2 setosa
##  8          4.9         3.1          1.5         0.1 setosa
##  9          4.8         3.4          1.6         0.2 setosa
## 10          4.8         3            1.4         0.1 setosa
## # ... with 19 more rows
```

2.4.7 Arranger les lignes avec arrange()

On peut vouloir que le jeu de données soit arrangé, ou trié, selon un certain ordre, en fonction des valeurs d'une variable donnée. La fonction `arrange()` du package `dplyr` est très utile pour gérer ce genre de réalisation. L'exemple ci-dessous conduit à trier les données selon un ordre croissant en fonction des valeurs de la variable `Sepal.Length`. Le fait de mettre le symbole `-` devant le nom de la variable aurait conduit à un tri décroissant.

```
iris %>%
  arrange(Sepal.Length)
```

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1          4.3         3            1.1         0.1 setosa
##  2          4.4         2.9          1.4         0.2 setosa
##  3          4.4         3            1.3         0.2 setosa
##  4          4.4         3.2          1.3         0.2 setosa
##  5          4.5         2.3          1.3         0.3 setosa
##  6          4.6         3.1          1.5         0.2 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          4.6         3.6          1           0.2 setosa
##  9          4.6         3.2          1.4         0.2 setosa
## 10          4.7         3.2          1.3         0.2 setosa
## # ... with 140 more rows
```

2.4.8 Résumer des variables avec group_by() et summarize()

Bien qu'une base de données puisse contenir énormément de lignes, on peut n'en vouloir que la version résumée. Les fonctions `group_by()` et `summarize()` du package `dplyr` permettent de faire cela aisément. Dans l'exemple ci-dessous, la fonction `group_by()` permet d'indiquer que les calculs réalisés par la suite avec la fonction `summarize()` doivent être exécutés pour les modalités de la variable `Species` prises séparément. La fonction `summarize()`, quant à elle, permet d'exéctuer différents calculs. Dans l'exemple ci-dessous, il s'agit de moyennes, obtenues à l'aide de la fonction `mean()`. De plus, la fonction `summarize()` permet, comme montré ci-dessous, d'indiquer à gauche du `=` le nom du titre du calcul alors effectué .

```
iris %>%
  group_by(Species) %>%
  summarize(mean_sep_len = mean(Sepal.Length),
            mean_sep_wid = mean(Sepal.Width))
```

```
## # A tibble: 3 x 3
##   Species    mean_sep_len mean_sep_wid
##   <fct>             <dbl>        <dbl>
## 1 setosa             5.01         3.43
## 2 versicolor         5.94         2.77
## 3 virginica          6.59         2.97
```

Au cours des illustrations montrant l'usage des fonctions `select()` jusqu'à `summarize()`, il aura été possible de noter que les commandes n'écrasaient pas le jeu de données initial, ni ne créaient de nouveaux jeux de données, car aucune assignation à un nom n'était faite. Lorsqu'une assignation est réalisée, il est conseillé d'utiliser un nouveau nom, différent de celui utilisé pour le jeu de données initial, pour pouvoir revenir au jeu de données originel lorsque cela est souhaité. Ci-dessous un exemple de création d'un nouvel objet de type tableau (assigné au nom `iris2`) à partir de l'utilisation de la plupart des fonctions que nous venons de voir et qui peuvent être utilisées dans un même bloc de code grâce au *pipe* (`%>%`).

```
iris2 <-
  iris %>%
  select(Petal.Length, Petal.Width, Species) %>%
  rename(Petal_len = Petal.Length, Petal_wid = Petal.Width) %>%
  mutate(Petal_ratio = Petal_len / Petal_wid) %>%
  filter((Species == "setosa" | Species == "virginica") & Petal_ratio > 8) %>%
  arrange(-Petal_ratio)
iris2
```

```
## # A tibble: 7 x 4
##   Petal_len Petal_wid Species Petal_ratio
##       <dbl>     <dbl> <fct>         <dbl>
## 1       1.5       0.1 setosa        15
## 2       1.5       0.1 setosa        15
## 3       1.4       0.1 setosa        14.0
## 4       1.4       0.1 setosa        14.0
## 5       1.1       0.1 setosa        11
## 6       1.9       0.2 setosa         9.50
## 7       1.7       0.2 setosa         8.5
```

2.4.9 Passer d'une disposition en lignes à une disposition en colonnes et inversement avec pivot_wider() et pivot_longer()

Il convient de respecter certaines règles de base lors de la conception d'une base de données (e.g., mettre les observations en lignes et les variables en colonnes). Toutefois, dans certains cas, même après avoir bien respecter les règles, la manière selon laquelle la base de données a été organisée peut ne pas être encore adéquate pour pouvoir utiliser certaines fonctions. Prenons par exemple le cas où toutes les valeurs numériques d'une variable quantitative auraient été mises dans une même colonne en regard d'une variable qualitative pour que chaque valeur numérique corresponde à une modalité de cette variable qualitative (c'est le cas, par exemple, avec le jeu de données `iris`), et que la fonction à utiliser

nécessiterait que l'on ait une colonne pour chacune des modalités de la variable qualitative, avec des colonnes mises côte à côte. Une fonction qui permet alors de passer d'un format "long" (i.e., toutes les valeurs numériques sont dans la même colonne) à un format "large" (i.e., les valeurs numériques sont réparties dans différentes colonnes selon la modalité à laquelle elles sont associées), est la fonction `pivot_wider()` du package `tidyr`. Pour pouvoir utiliser cette fonction, il faut qu'il y ait une variable permettant d'identifier à quel individu ou groupe appartienne les données dont on va changer l'organisation. Dans une base de données classique, il y a toujours une variable présente pour cela. Toutefois, dans le jeu de données `iris`, il n'y a pas une telle variable. Pour pouvoir illustrer l'utilisation de la fonction `pivot_wider()`, nous avons donc ajouté arbitrairement une variable `id` grâce à la fonction `mutate()` pour simuler le fait que les données de `iris` auraient été acquises en référence à des individus bien identifiés, avec à chaque fois une valeur pour les trois modalités de la variable `Species`.

```
iris2 <-
  iris %>%
  mutate(id = rep(1:50, times = 3)) %>%
  select(id, Species,  everything()) %>%
  arrange(id, Species) %>%
  as_tibble()
iris2
```

```
## # A tibble: 150 x 6
##       id Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
##    <int> <fct>             <dbl>       <dbl>        <dbl>       <dbl>
##  1     1 setosa              5.1         3.5          1.4         0.2
##  2     1 versicolor          7           3.2          4.7         1.4
##  3     1 virginica           6.3         3.3          6           2.5
##  4     2 setosa              4.9         3            1.4         0.2
##  5     2 versicolor          6.4         3.2          4.5         1.5
##  6     2 virginica           5.8         2.7          5.1         1.9
##  7     3 setosa              4.7         3.2          1.3         0.2
##  8     3 versicolor          6.9         3.1          4.9         1.5
##  9     3 virginica           7.1         3            5.9         2.1
## 10     4 setosa              4.6         3.1          1.5         0.2
## # ... with 140 more rows
```

La fonction `pivot_wider()` permet alors de mettre en colonnes les valeurs des variables sélectionnées pour chacune des trois modalités de la variable `Species` (ci-dessous, seules les trois colonnes relatives à la longueur des sépales des fleurs (`Sepal.Length`) sont montrées, mais le reste a bien été crée).

```
iris3 <-
  iris2 %>%
  pivot_wider(names_from = Species,
              values_from = Sepal.Length : Petal.Width)
iris3
```

```
## # A tibble: 50 x 13
##       id Sepal.Length_se~ Sepal.Length_ve~ Sepal.Length_vi~
##    <int>            <dbl>            <dbl>            <dbl>
##  1     1              5.1              7                6.3
##  2     2              4.9              6.4              5.8
##  3     3              4.7              6.9              7.1
##  4     4              4.6              5.5              6.3
##  5     5              5                6.5              6.5
##  6     6              5.4              5.7              7.6
##  7     7              4.6              6.3              4.9
##  8     8              5                4.9              7.3
##  9     9              4.4              6.6              6.7
## 10    10              4.9              5.2              7.2
## # ... with 40 more rows, and 9 more variables: Sepal.Width_setosa <dbl>,
## #   Sepal.Width_versicolor <dbl>, Sepal.Width_virginica <dbl>,
## #   Petal.Length_setosa <dbl>, Petal.Length_versicolor <dbl>,
## #   Petal.Length_virginica <dbl>, Petal.Width_setosa <dbl>,
## #   Petal.Width_versicolor <dbl>, Petal.Width_virginica <dbl>
```

L'argument `names_from` a permis d'indiquer la variable à partir de laquelle on a dispatché les valeurs en colonnes, et l'argument `values_from` a permis de préciser les variables pour lesquelles on voulait que les valeurs numériques soient dispatchées. L'utilisation des deux-points (`:`) nous a permis de sélectionner toutes les variables allant de `Sepal.Length` à `Petal.Width` dans le jeu de données.

Dans une situation inverse à celle que nous venons de voir, nous pourrions avoir les données représentées en colonnes (comme c'est le cas avec le jeu de données crée `iris3`), alors que nous les voudrions en lignes (comme c'était le cas avec le jeu de données `iris` à l'origine). La fonction `pivot_longer` permet de faire cette conversion.

```
iris4 <-
  iris3 %>%
  pivot_longer(cols = c(Sepal.Length_setosa, Sepal.Length_versicolor, Sepal.Length_virginica),
               names_to = "Species",
               values_to = "Sepal_len") %>%
  select(id, Species, Sepal_len)
iris4
```

```
## # A tibble: 150 x 3
##       id Species                 Sepal_len
##    <int> <chr>                       <dbl>
##  1     1 Sepal.Length_setosa           5.1
##  2     1 Sepal.Length_versicolor       7
##  3     1 Sepal.Length_virginica        6.3
##  4     2 Sepal.Length_setosa           4.9
##  5     2 Sepal.Length_versicolor       6.4
##  6     2 Sepal.Length_virginica        5.8
##  7     3 Sepal.Length_setosa           4.7
##  8     3 Sepal.Length_versicolor       6.9
##  9     3 Sepal.Length_virginica        7.1
## 10     4 Sepal.Length_setosa           4.6
## # ... with 140 more rows
```

Dans la fonction `pivot_longer()` ci-dessus, nous avons indiqué à l'aide de l'argument `cols` et de la fonction `c()` l'ensemble des variables dont nous souhaitions voir les valeurs regroupées dans une même colonne. L'argument `names_to` nous a permis de donner un nom à la variable qualitative qui comporte les modalités à associer aux valeurs numériques, et l'argument `values_to` nous a permis de donner un nom à la colonne où se trouvent maintenant les valeurs numériques. Les deux nouvelles colonnes ainsi crées se trouvent tout à gauche du tableau `iris4`, mais nous avons sélectionné pour cet exemple seulement les colonnes qui permettaient d'illustrer le résultat de la fonction. On peut remarquer cependant qu'en créant ces colonnes, R a dû dupliquer les valeurs de la variable id et des autres colonnes pour pouvoir garder la correspondance entre les données, chose que l'on peut voir en tapant `iris4` dans la Console.

Nous venons de voir plusieurs fonctions qui peuvent être très utiles pour pouvoir facilement préparer sa base de données en vue des futures analyses. Il ne s'agit que d'une vue très superficielle de tout le potentiel de manipulation des données qu'ont ces fonctions. Pour une vue approndie des possibilités qu'offrent ces fonctions, la lecture de l'ouvrage *R for Data Science* d'Hadley Wickham et de Garrett Grolemund (2017) sera particulièrement enrichissante. Cet ouvrage est en libre accès ici : https://r4ds.had.co.nz.

2.5 Résumé

- La base de données est un tableau comportant l'ensemble des données avec les observations organisées en lignes et les variables organisées en colonnes.
- Les grands types de variables que l'on peut retrouver dans une base de données sont les variables quantitatives continues, les variables quantitatives discrètes, les variables qualitatives nominales, et les variables qualitatives ordinales.

- Avant d'initier un travail d'analyse, il peut être utile de fixer le répertoire avec la fonction `setwd()`.
- Pour importer un jeu de données au format .csv, il est possible d'utiliser la fonction `readr::read_csv2()`.
- Pour exporter un jeu de données au format .csv, il est possible d'utiliser la fonction `readr::write_csv2()`.
- Pour mettre un tableau de donnnées au format *data frame*, utiliser la fonction `as.data.frame()`.
- Pour mettre un tableau de données au format *tibble*, utiliser la fonction `tibble::as_tibble()`.
- Pour lister les variables présentes dans un tableau de données, utiliser la fonction `str()`.
- Pour modifier les types des variables, utiliser des fonctions comme `as.numeric()`, `as.factor()`, `as.character()`, `as.Date()`, etc.
- Pour sélectionner les variables d'un tableau de données, utiliser la fonction `dplyr::select()`.
- Pour renommer les variables d'un tableau de données, utiliser la fonction `dplyr::rename()`.
- Pour créer de nouvelles variables dans un tableau de données, utliser la fonction `dplyr::mutate()`.
- Pour sélectionner des lignes dans un tableau de données, utiliser la fonction `dplyr::filter()`.
- Pour trier les lignes d'un tableau de données, utiliser la fonction `dplyr::arrange()`.
- Pour résumer les variables d'un tableau de données, utiliser les fonctions `dplyr::group_by()` et `dplyr::summarize()`.
- Pour passer d'un tableau de données au format *long* à un tableau de données au format *wide*, utiliser la fonction `tidyr::pivot_wider()`.
- Pour passer d'un tableau de données au uformat *wide* à un tableau de données au format *long*, utiliser la fonction `tidyr::pivot_longer`.
- Pour enchaîner l'application de fonctions, utiliser le symbole `%>%` (*pipe*, package `magrittr`).

Chapter 3

Analyses descriptives univariées

Réaliser une analyse descriptive univariée signifie que l'on s'intéresse à une seule variable en particulier. L'enjeu est ici de prendre connaissance de la distribution de la variable, c'est-à-dire de la manière selon laquelle se répartissent les observations en fonction des valeurs que prend la variable. Autrement dit, il s'agit d'avoir une idée du nombre d'observations (qui peut être le nombre d'individus dans certains cas, tels que dans des études portant sur l'être humain) en lien avec les différentes valeurs qui ont été obtenues. De manière complémentaire, l'analyse descriptive univariée vise à prendre connaissance des indices statistiques qui caractérisent la variable, ainsi qu'à déterminer les indices statistiques qui seraient les plus pertinents pour la résumer.

Dans cette partie, les notions de **population** et d'**échantillon** vont revenir à plusieurs reprises. La notion de population désigne tous les individus existant qui satisfont à un ou plusieurs critères particuliers (e.g., les adultes de moins de 30 ans). En général, lorsque l'on souhaite étudier un phénomène dans une population cible, il est impossible de prendre en compte tous les individus de la population en question. L'alternative est alors de conduire l'étude sur un échantillon, c'est-à-dire une fraction de la population composée d'individus qui représentent la population étudiée. La distinction entre population et échantillon est importante à faire à plusieurs égards. Si l'étude n'a pu être conduite que sur un échantillon, cela implique de mettre en oeuvre des procédures statistiques pour estimer avec plus ou moins d'incertitude le résultat réel concernant la population étudiée, cela à partir du résultat trouvé dans l'échantillon observé. La seule analyse descriptive de l'échantillon ne suffit donc pas en soit à décrire une population. En revanche, lorsque l'étude a pu être conduite sur l'ensemble de la population à étudier (e.g., l'équipe de France dans un sport donné), il n'y a par définition pas lieu de chercher à conduire des procédures statistiques particulières pour estimer le résultat réel pour la population en question. Dans

ce chapitre, les procédures d'analyse proposées servent en général à seulement décrire la variable telle qu'elle est donnée à voir à partir des données que l'on a en sa possession. L'objectif n'est donc pas ici de discuter particulièrement des statistiques les plus pertinentes à utiliser lorsqu'il s'agit de chercher à résumer la distribution d'une variable à l'échelle d'une population à partir d'un échantillon initial. Pour le moment, il s'agit d'être en mesure de décrire l'échantillon (ou la population si les données obtenues concernent toute la population) que l'on a sous les yeux.

Dans le cadre de cette partie, nous allons commencer à voir comment produire des graphiques dans RStudio, et par là-même, découvrir progressivement le package ggplot2. Le package ggplot2 n'est pas le plus simple à utiliser lorsque l'on découvre le logiciel R. D'ailleurs, de nombreux manuels portant sur R privilégient les packages et fonctions de base de R lorsqu'il s'agit de montrer comment obtenir des graphiques relativement simples pour analyser ses données. Cependant, les packages et fonctions de base de R sont rapidement limités lorsqu'il s'agit de réaliser des graphiques relativement complexes. Le parti pris ici est donc d'initier dès à présent à l'utilisation du package ggplot2 pour réaliser des graphiques, même simples, afin de pouvoir être plus rapidement à l'aise dès lors qu'il s'agira par la suite de produire des graphiques relativement élaborés à l'aide de ce package. Cependant, l'ambition n'est pas ici de permettre la maîtrise complète du package ggplot2. Pour cela, il vaut mieux se référer à des documentations spécialisées telles que la seconde édition de l'ouvrage *ggplot2* d'Hadley Wickham (2016), en sachant qu'une troisième édition est en cours de développement et est accessible en ligne ici : https://ggplot2-book.org.

3.1 Variables quantitatives

3.1.1 Visualiser la distribution de la variable

Dans le cadre de l'analyse de variables quantitatives, il est toujours utile de d'abord visualiser graphiquement la distribution des données à l'aide d'un **histogramme**. Un histogramme, c'est un graphique avec des barres dont la largeur représente un intervalle donné de valeurs numériques, et dont la hauteur représente le nombre d'observations associées à une valeur qui est située dans l'intervalle en question. Plus une barre est longue, plus il y a d'observations concernées par l'intervalle de valeurs. Un exemple d'histogramme est montré ci-dessous.

```
ggplot(data = iris, aes(x = Sepal.Length)) +
  geom_histogram(fill = "white", color = "black")
```

Pour générer cet histogramme, nous avons utilisé les fonctions `ggplot()` et `geom_histogram()` du package `ggplot2`. La fonction `ggplot()` est nécessaire

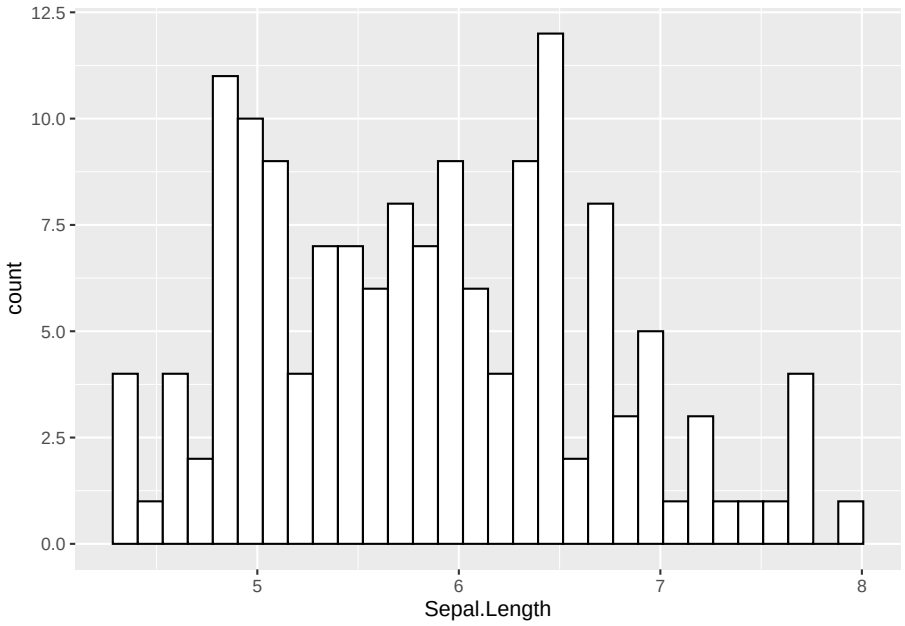


Figure 3.1: Exemple d'histogramme

pour initier le graphique. Si on lance la commande `ggplot()` dans la Console, on peut voir qu'un écran grisé apparaît à droite de l'écran du PC dans la fenêtre **Plots** de RStudio. Cet écran grisé est tel un tableau vierge qui ne demande qu'à être complété grâce à des commandes supplémentaires que l'on doit préciser dans le code. Dans le code montré ci-dessus, on remarque que la fonction `ggplot()` a été configurée à l'aide de deux arguments : `data`, et `aes()`. L'argument `data` désigne le jeu de données à partir duquel n'importe quelle autre fonctionnalité du package `ggplot2` sera utilisée si rien d'autre n'est précisé dans le reste du code. Comme on peut le voir, le jeu de données utilisé ici est `iris`, que nous avons déjà rencontré dans la partie précédente. L'argument `aes()`, lui, permet de désigner les données à partir desquelles les éléments graphiques indiqués par la suite devront être réalisés. Dans le cadre d'une analyse univariée, nous n'avons besoin que d'une seule variable. Celle-ci peut être renseignée à droite de `x =`, et on aura reconnu ici le nom d'une variable effectivement présente dans le jeu de données `iris`. Une fois que ces informations sont renseignées, nous ne sommes pas encore en mesure de voir un quelconque graphique. Pour cela, il faut que la fonction `ggplot()` soit accompagnée d'une fonction qui permette d'indiquer quel type de graphique on veut. C'est à cela que sert ici la fonction `geom_histogram()`. On peut noter que l'ajout de cette fonction a été réalisé grâce au signe `+`, en écrivant la fonction *après* ce signe. La fonction `geom_histogram()` aurait pu être écrite directement après le symbole `+`, mais pour des raisons de lisibilité, nous sommes allés à la ligne. (Attention : Aller à la ligne *avant* le signe `+` n'est en revanche

pas possible.) De manière intéressante (et importante) pour la suite, on pourra noter que dans ce cas de figure, nous aurions pu aussi utiliser le symbole *pipe* (`%>%`) pour enchaîner la création d'un graphique à la suite de l'écriture du jeu de données qu'on aurait indiqué initialement dans le code, comme cela :

```
iris %>%
  ggplot(aes(x = Sepal.Length)) +                    # On remarque ici que l'argument da
  geom_histogram(fill = "white", color = "black")
```

Comme l'indique le message qui accompagne le graphique, l'histogramme a été réalisé sur la base de 30 *bins*. Cela signifie que pour faire ce graphique, R a découpé en 30 intervalles égaux l'intervalle allant de la valeur la plus faible de la variable (i.e., le minimum) à la valeur la plus élevée de la variable (i.e., le maximum). Il s'agit de la méthode par défaut utilisée par la fonction `geom_histogram()`. Toutefois, cette méthode par défaut n'est pas vraiment adaptée, comme cela l'est indiqué d'ailleurs dans la documentation d'aide associée à cette fonction. Et puis, lorsqu'il s'agit d'appréhender au mieux la distribution d'une variable avec un histogramme, une bonne pratique est d'observer ce qu'il se passe avec différentes largeurs de *bins*. La largeur d'une *bin* peut être modifiée à l'aide de l'argument `binwidth`. L'unité de la valeur associée à cet argument correspond à l'unité de la variable étudiée.

```
# Graphique avec binwidth = 0.3
ggplot(data = iris, aes(x = Sepal.Length)) +
  geom_histogram(binwidth  = 0.3, fill = "white", color = "black") +
  ggtitle("binwidth = 0.3")

# Graphique avec binwidth = 0.7
ggplot(data = iris, aes(x = Sepal.Length)) +
  geom_histogram(binwidth  = 0.7, fill = "white", color = "black") +
  ggtitle("binwidth = 0.7")
```

En plus de l'histogramme, une autre manière de prendre connaissance graphiquement de la distribution des données d'une variable quantitative est d'utiliser une **boîte à moustaches**. Pour ce faire, il convient d'utiliser la fonction `geom_boxplot()`, comme illustré ci-dessous. (On pourra noter l'ajout d'une ligne de code avec une fonction `theme()` dont ne présentera pas les détails ici ; cette fonction nous sert juste ici à ne pas montrer des chiffres qui auraient été ajoutés par défaut à gauche du graphique et qui n'auraient eu aucun intérêt.)

```
ggplot(data = iris, aes(x = Sepal.Length)) +
  geom_boxplot() +
    theme(axis.text.y = element_blank(),
          axis.ticks.y = element_blank())
```

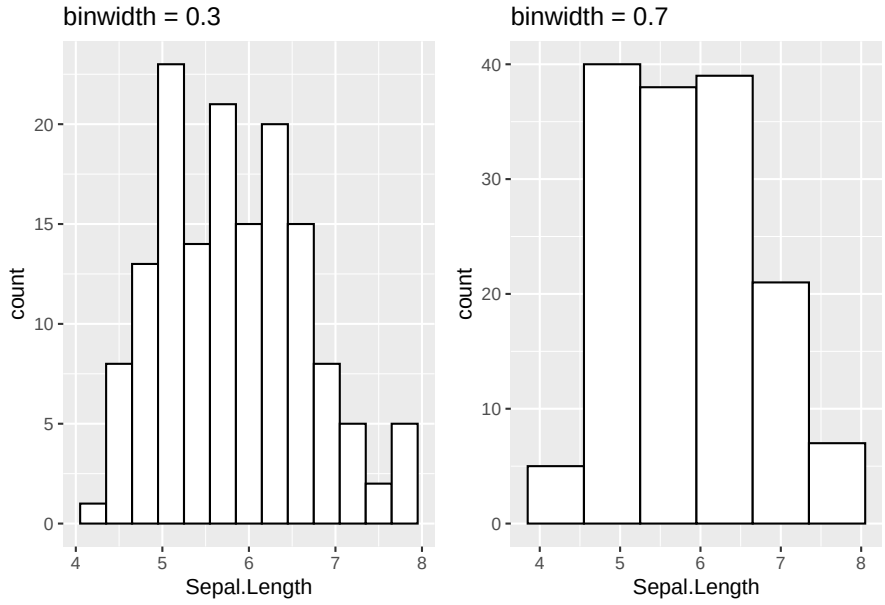


Figure 3.2: Différentes largeurs d'intervalles pour un histogramme

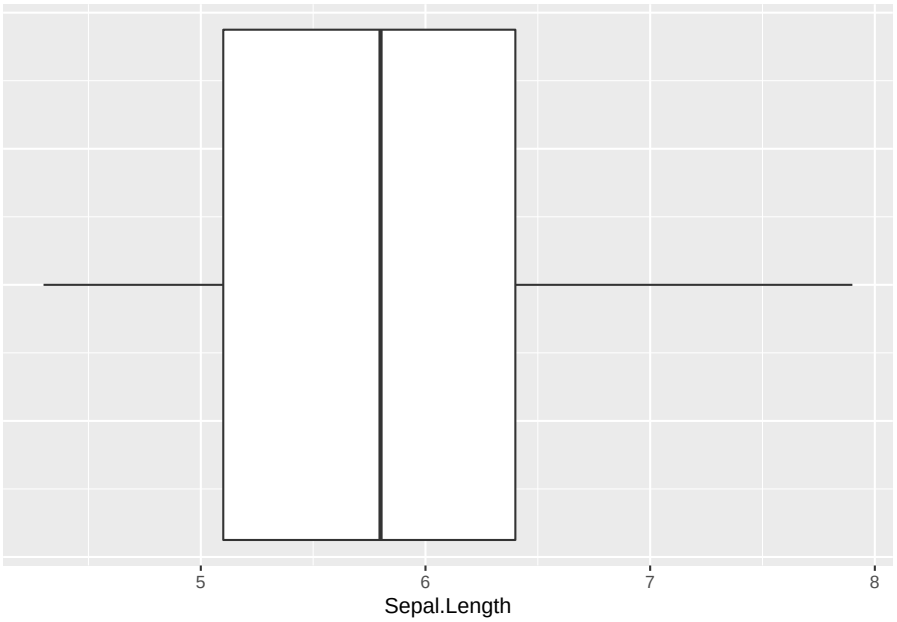


Figure 3.3: Exemple de boîte à moustaches

La boîte à moustaches nous livre plusieurs informations. Tout d'abord, ses extrémités nous indiquent ce qu'on appelle le premier quartile (ici représenté par le bord gauche de la boîte) et le troisième quartile (ici représenté par le bord droit de la boîte). Le premier quartile (Q1) désigne la valeur en-dessous de laquelle on retrouve 25 % des observations de la variable (i.e., 25 % des observations sont associées à une valeur plus faible que Q1), alors que le troisième quartile (Q3) représente la valeur en-dessous de laquelle on retrouve 75 % des observations (i.e., 75 % des observations sont associées à une valeur plus faible que Q3). Cela indique alors que sur le graphique montré ci-dessus, l'intervalle qui sépare le bord gauche du bord droit de la boîte contient 50 % des observations. La ligne noire à l'intérieur de la boîte blanche désigne la médiane, qui est la valeur pour laquelle on a 50 % des observations qui ont une valeur inférieure à cette valeur repère, et pour laquelle on a 50 % des observations qui ont une valeur supérieure à cette valeur repère. Les lignes noires en-dehors de la boîte sont les moustaches. Dans le cas présent, la moustache de gauche s'étend jusqu'à la valeur minimale de la variable, et la moustache de droite s'étend jusqu'à la valeur maximale de la variable. Dans le cas où le minimum (ou le maximum) aurait été éloigné de la médiane de plus de 1.5 fois la différence entre Q3 et Q1 (qu'on appelle **intervalle interquartile**), l'extrémité de la moustache se serait arrêtée à la dernière valeur avant cette limite, et toute valeur ayant dépassé cette limite aurait été représentée par un point. Pour illustrer ce dernier cas de figure, on peut modifier manuellement une valeur de la variable Sepal.Length de telle sorte à ce qu'il y ait une nouvelle valeur qui soit particulièrement éloignée de la boîte. Une telle valeur s'appelle un *outlier*.

```
iris$Sepal.Length[3] <- 12                    # On modifie ici, pour l'exemple, la va
ggplot(data = iris, aes(x = Sepal.Length)) +  # en lui assignant la valeur 12.
  geom_boxplot() +
    theme(axis.text.y = element_blank(),
          axis.ticks.y = element_blank())
```

Une boîte à moustaches a donc notamment l'intérêt de mettre en évidence des valeurs qui apparaissent comme "étranges" par rapport au reste des données. Lorsqu'il semble évident que l'*outlier* est une valeur erronnée (ou quand on veut tout simplement vérifier qu'il s'agit d'une erreur ou non), il est intéressant de savoir à quelle observation (i.e., à quel individu dans certains contextes) cette valeur étrange appartient, pour ensuite éventuellement la corriger. Malheureusement, la fonction `geom_boxplot()` ne dispose pas d'argument pour permettre d'identifier facilement à quelle observation appartient cette valeur. Cependant, on peut s'appuyer sur des fonctions créées manuellement dans R pour parvenir à cela. Dans ce cas de figure, le site https://stackoverflow.com est souvent intéressant car riche de solutions. C'est d'ailleurs à partir de ce site que provient la fonction montrée ci-dessous (https://stackoverflow.com/questions/33524669/labeling-outliers-of-boxplots-in-r) qui va nous permettre ensuite de savoir, à partir du graphique, à quelle observation correspond cette donnée étrange.

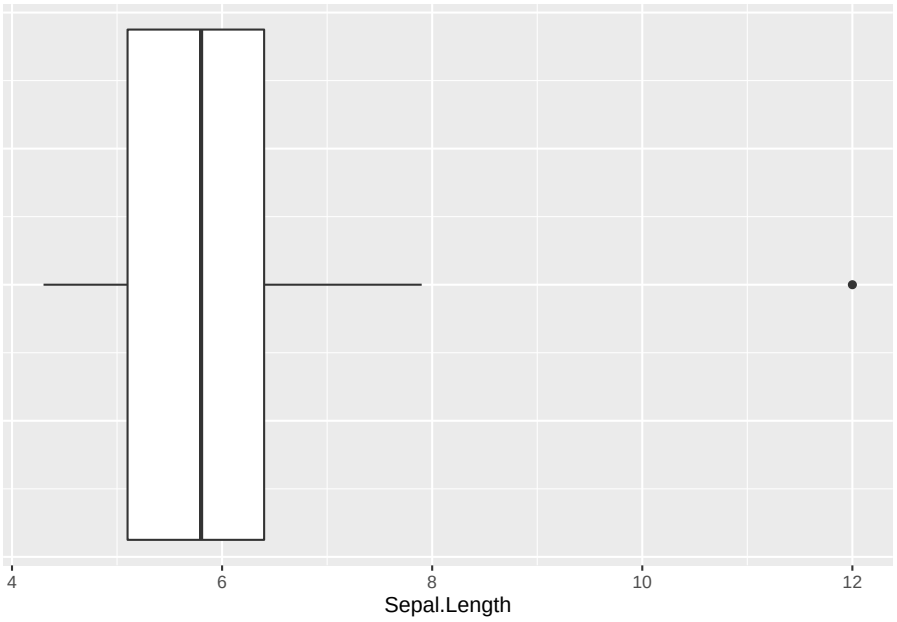


Figure 3.4: Visualisation d'un outlier

```
is_outlier <- function(x) {
  return(x < quantile(x, 0.25) - 1.5 * IQR(x) | x > quantile(x, 0.75) + 1.5 * IQR(x))
}
```

Voici donc, ci-dessus, à quoi ressemble une fonction à son état brut, avec : le nom de la fonction à gauche de la flèche d'assignation (`<-`), la commande `function()` qui permet d'amorcer la création de la fonction, et les lignes de code entre les accolades { } qui indiquent les actions que la fonction réalise. La seule chose qu'il faut comprendre à ce stade, c'est que cette fonction, qui va donc s'appeler par la suite `is_outlier()`, a besoin pour fonctionner qu'on lui indique un nom de variable (représenté par la lettre `x` dans le code ci-dessus), et que le résultat de cette fonction sera une nouvelle variable qui contiendra seulement des `TRUE` ou des `FALSE`, en sachant que `TRUE` correspondra au fait que la valeur de la variable étudiée était un *outlier*, et que `FALSE` correspondra au fait que la valeur de la variable étudiée n'était pas un *outlier*. (Notons ici que la définition d'un *outlier* est la même que celle décrite plus haut, à savoir une valeur qui serait éloignée de la médiane de plus de 1.5 fois l'intervalle interquartile.) Mais regardons concrètement ce que donne cette fonction lorsqu'elle est appliquée à la variable Sepal.Length du jeu de données `iris` (NB : La fonction ne marchera que si elle a été activée/créée auparavant) :

```
is_outlier(x = iris$Sepal.Length)
```

```
##   [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [144] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Quand on regarde bien, on voit que la troisème observation de cette nouvelle variable que l'on vient de créer (seulement de manière temporaire ici car on ne l'a pas assignée à un nom) contient la valeur `TRUE`, ce qui est en accord avec la valeur que nous avons introduite auparavant dans la variable Sepal.Length. Le fait d'observer ces valeurs `TRUE` et `FALSE` n'est évidemment pas une stratégie très pratique pour déterminer à quelle observation correspondrait l'*outlier*, et c'est pourquoi l'étape suivante consiste à montrer comment on peut se servir de cette fonction `is_outlier()` pour faire apparaître sur un graphique de boîte à moustaches les observations à qui appartiendraient les valeurs étranges.

```
iris %>%
  mutate(id = as.factor(rep(1:50, times = 3)),                          # Ajout d'un n
         id_outlier = ifelse(is_outlier(x = Sepal.Length), id, "")) %>% # Création d'u
  ggplot(aes(x = Sepal.Length, y = "")) +
  geom_boxplot() +
  geom_text(aes(label = id_outlier),  hjust = -1) +                     # Ajout des nu
    theme(axis.text.y = element_blank(),
          axis.ticks.y = element_blank(),
          axis.title.y = element_blank())
```

Il y a plusieurs choses à expliquer par rapport au graphique qui comporte à présent, comme on peut le voir ci-dessus, le numéro id associé à l'observation pour laquelle nous avions modifié la valeur. Tout d'abord, il faut noter qu'avant de créer le graphique, nous avons ajouté temporairement au jeu de données, avec la fonction `mutate()`, la variable `id_outlier`. Cette variable a été créée à l'aide de deux fonctions en réalité : la fonction `ifelse()`, et la fonction `is_outlier()` qu'on a présentée succintement plus haut. Ici, la fonction `ifelse()` a fonctionné

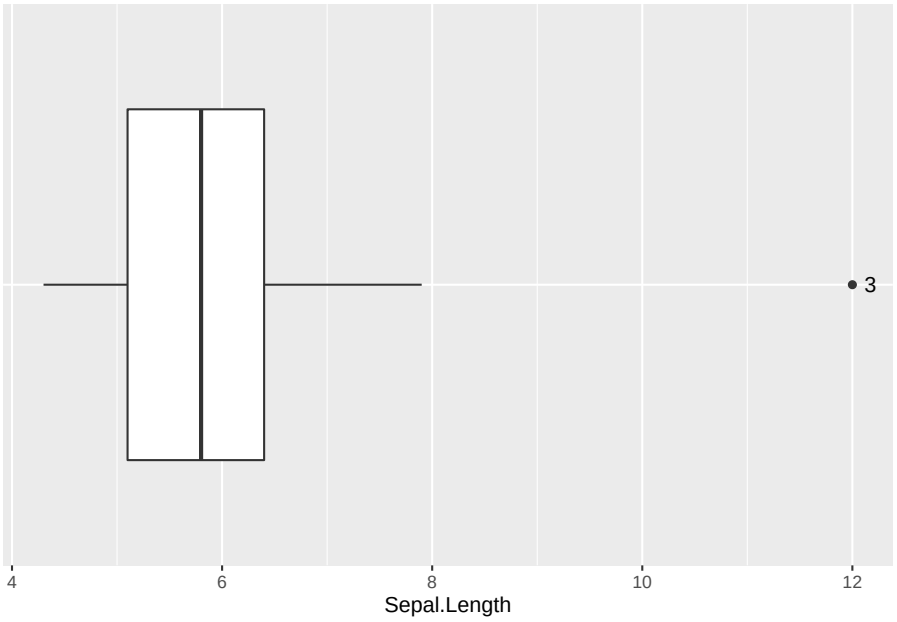


Figure 3.5: Identification d'un outlier

comme cela : si la fonction `id_outlier()` renvoyait la valeur `TRUE`, alors on conservait le numéro id de la variable Sepal.Length, sinon, on mettait un espace vide. Cela veut dire que la variable id_outlier ne contenait que les numéros id pour lesquels la fonction `is_outlier()` avait renvoyé la valeur `TRUE`. Pour visualiser ce qu'il s'est passé, on peut revoir la conséquence du début du code qui a permis de faire le graphique (cf. à droite du tableau ci-dessous) :

```
iris %>%
  mutate(id = as.factor(rep(1:50, times = 3)),
         id_outlier = ifelse(is_outlier(x = Sepal.Length), id, "")) %>%
  select(id, Sepal.Length, id_outlier)
```

```
## # A tibble: 150 x 3
##    id    Sepal.Length id_outlier
##    <fct>        <dbl> <chr>
##  1 1              5.1 ""
##  2 2              4.9 ""
##  3 3             12   3
##  4 4              4.6 ""
##  5 5              5   ""
##  6 6              5.4 ""
##  7 7              4.6 ""
```

```
##  8 8             5   ""
##  9 9             4.4 ""
## 10 10            4.9 ""
## # ... with 140 more rows
```

Une fois cette procédure réalisée, le reste du code, et notamment la fonction `geom_text()`, a permis d'ajouter des éléments textuels au graphique, en l'occurence en s'appuyant sur la variable id_outlier, tel que configuré avec l'argument `aes()` de la fonction `geom_text()`.

L'ensemble de la procédure présentée ci-dessus a le mérite de conduire exactement au résultat que l'on veut et seulement à ce résultat. Toutefois, elle peut être un peu fastidieuse. Un moyen sans doute plus rapide, mais qui a également ses limites, est l'utilisation de la fonction `ggbetweenstats` du package `ggstatsplot`. Un exemple d'utilisation de cette fonction est montré ci-dessous.

```
library(ggstatsplot)
iris %>%
  mutate(id = as.factor(rep(1:50, times = 3))) %>%
  ggbetweenstats(x = Species,
                 y = Sepal.Length,
                 outlier.tagging = TRUE,
                 outlier.label = id)
```

Dans cet exemple, nous avons indiqué, pour l'argument `x` de la fonction `ggbetweenstats`, une variable qualitative du jeu de données `iris`, car la fonction contraint à devoir renseigner une variable pour cet argument afin de pouvoir afficher le graphique. Ainsi, il n'était pas possible de visualiser toutes les données de manière groupée, c'est-à-dire toutes espèces confondues si l'on considère le jeu de données `iris`. Plusieurs éléments sont montrés sur le graphique. La seule chose qui nous importe ici, c'est le numéro en haut à gauche, qui est l'identifiant de l'observation présentant une valeur anormale.

Lorsque la valeur anormale identifiée est effectivement une erreur de saisie dans la base de données, il convient de corriger la valeur avec la fonction d'assignation comme nous l'avons fait précédemment :

```
iris$Sepal.Length[3] <- 4.7  # Le nombre entre crochets désigne la position de l'obser
```

On remarque ainsi qu'au-delà de prendre connaissance de la forme de la distribution, passer par ces étapes graphiques permet aussi de s'assurer qu'il n'y a pas eu d'erreur lors de la saisie des données dans la base (du moins, pas d'erreur visible et qui risquerait d'impacter grandement les calculs futurs). Passer par l'analyse graphique est donc recommandé avant de pouvoir se fier aux résultats numériques que l'on pourrait calculer par la suite, tels que les indices statistiques qui permettent de résumer numériquement une variable. Afin de faciliter

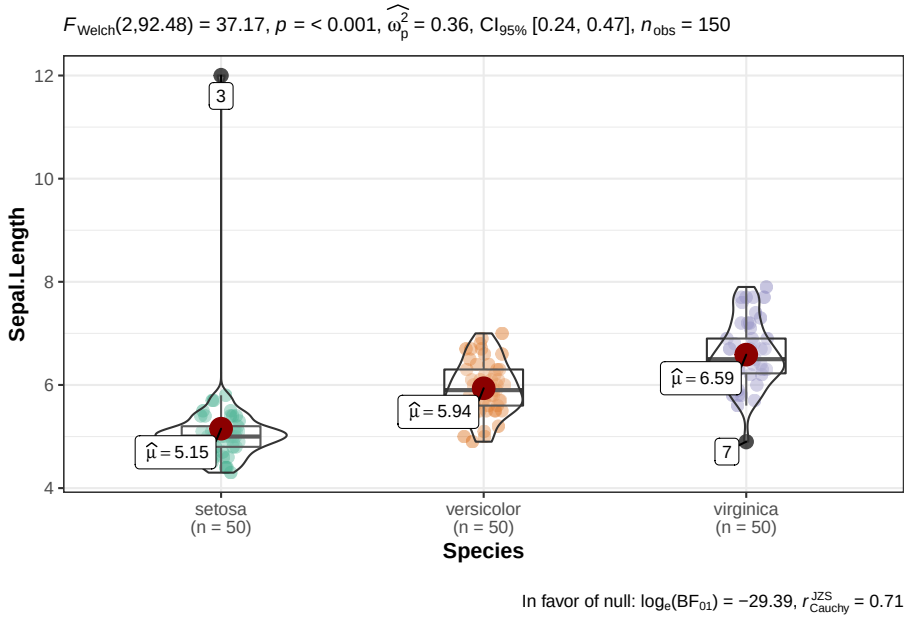


Figure 3.6: Visualisation d'un outlier avec 'ggstatplot::ggbetweenstats'

l'analyse graphique de la distribution de la variable étudiée, nous avons crée une fonction, appelée `g_distri()` (cf. ci-dessous), visant à produire à la fois un histogramme et une boîte à moustaches représentant la variable étudiée, ainsi qu'un graphique de type nuage de points montrant les valeurs individuelles (la fonction a été configurée de telle sorte à ce que les points apparaissent davantage de couleur blanche là où il y a plus d'observations ; les observations isolées seront montrées avec des points relativement transparents).

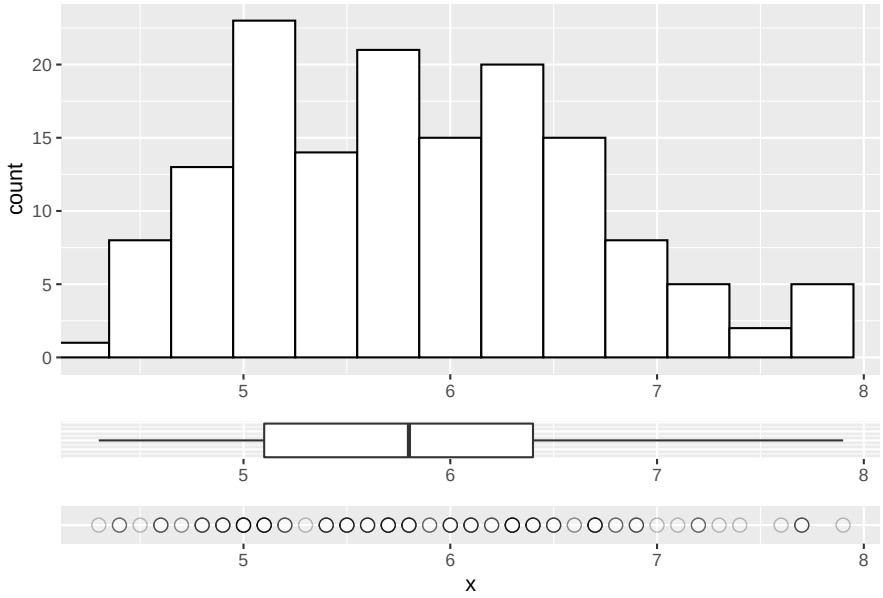
```
g_distri <- function(x = NULL, binwidth = NULL) {
  require(ggplot2)
  require(magrittr)
  require(patchwork)

  data <- as.data.frame(x)
  names(data) <- "x"

  g1 <-
    data %>%
    ggplot(aes(x = x)) +
    geom_histogram(binwidth = binwidth, fill = "white", color = "black") +
    coord_cartesian(xlim = c(min(x, na.rm = TRUE), max(x, na.rm = TRUE))) +
    theme(axis.title.x = element_blank())
```

```
  g2 <-
    data %>%
    ggplot(aes(x = x)) +
    geom_boxplot() +
    coord_cartesian(xlim = c(min(x, na.rm = TRUE), max(x, na.rm = TRUE))) +
    theme(axis.text.y = element_blank(),
          axis.title.x = element_blank(),
          axis.ticks.y = element_blank())

    g3 <-
      data %>%
      ggplot(aes(x = x)) +
      geom_point(aes(x = x, y = ""), shape = 21, fill = "white", alpha = 0.25, size = 3
      ylab("") +
      coord_cartesian(xlim = c(min(x, na.rm = TRUE), max(x, na.rm = TRUE))) +
      theme(axis.text.y = element_blank(),
          axis.ticks.y = element_blank())

  g1 + g2 + g3 + plot_layout(nrow = 3, heights = c(2, 0.2, 0.2))
}
```

L'utilisation de la fonction `g_distri()` est illustrée ci-dessous (pour rappel, il est important d'activer/créer la fonction avant de pouvoir l'utiliser) :

```
g_distri(x = iris$Sepal.Length, binwidth = 0.3)
```

\begin{figure}



{

}

\caption{Vsualisation de la distribution avec la fonction `g_distri()`}
\end{figure}

Lors de l'analyse de données, différentes formes typiques de distribution peuvent être rencontrées, notamment des formes **gaussiennes**, **asymétriques**, **leptocurtiques**, et **platycurtiques** (Dart & Chatellier, 2003). Ces formes sont illustrées sur la figure ci-dessous. Les formes gaussiennes sont observées en présence de variables suivant ce qu'on appelle une **loi normale**. Très souvent, on associe une distribution gaussienne, et donc une loi normale, à une distribution en forme de cloche, bien que l'analogie à la cloche pourrait se discuter. Les formes asymétriques traduisent le fait que la majorité des observations sont concentrées sur une extrémité de l'intervalle des valeurs possibles, et qu'il existe des observations, non majoritaires, avec des valeurs pouvant être très éloignées de la majorité des données, mais seulement d'un seul côté de la distribution. Enfin, les formes leptocurtiques et platycurtiques sont appelées ainsi par comparaison à la forme gaussienne. En présence d'une forme leptocurtique, la distribution s'avère plus pointue, avec des queues (qui sont les extrémités de la distribution) plus longues qu'avec une forme gaussienne. Dans le cadre d'une distribution platycurtique, la distribution est plus aplatie, avec des queues plus courtes qu'avec une forme gaussienne (Dart & Chatellier, 2003). La distribution uniforme est un cas particulier de distribution platycurtique, et c'est cette distribution qui est en réalité montrée sur la figure ci-dessous.

Comme il le sera vu plus tard, de nombreuses analyses statistiques reposent

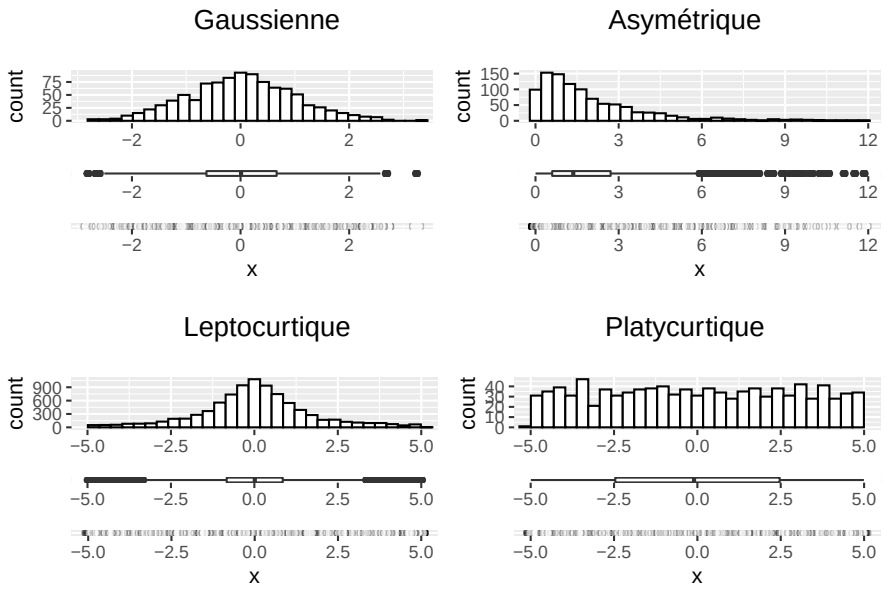


Figure 3.7: Différents types de distributions

sur l'hypothèse que la variable étudiée est gaussienne. Il est donc important de chercher, graphiquement dans un premier temps, à savoir si la distribution de la variable est effectivement gaussienne. Le fait d'être capable d'identifier les autres formes de distribution peut être aussi important afin de mener des analyses appropriées. Dans les exemples montrés ci-dessus, les distributions ont été créées à partir de 1000 valeurs générées de manière aléatoire de telle sorte à suivre des lois prédéfinies et ainsi illustrer différentes distributions possibles. C'est pour cette raison que les formes de distribution montrées sur la figure ci-dessus sont si nettes. Lorsque l'on travaille dans certains domaines ou contextes, tel qu'avec l'être humain, il peut être compliqué d'obtenir autant de données, et les formes de distribution seront alors plus dures à identifier.

Une fois qu'une première analyse graphique des données a été réalisée, il peut être utile de chercher à résumer de manière numérique la variable. Plusieurs types de statistiques peuvent être utilisés à cet effet : les **indices de position**, les **indices de dispersion**, les **indices d'asymétrie**, et les **indices d'aplatissement**.

3.1.2 Les indices de position

Les indices de position servent à donner un ordre de grandeur de la variable. Autrement dit, ces indices permettent de positionner la variable sur une

échelle de valeurs numériques. De plus, ces statistiques peuvent être utilisées pour donner une idée de ce qu'on appelle la **tendance centrale**, c'est-à dire-la valeur typique d'une distribution qui donne une bonne indication de la localisation de la majorité des observations (Rousselet & Wilcox, 2020). Différentes statistiques peuvent être étudiées à cette fin : la **moyenne**, la **médiane**, la **moyenne rognée**, et le **mode**.

La moyenne

Si l'on pose que N est le nombre de valeurs dans une variable (on parle également de **taille** de la variable), que i est la i-ème observation (i-ème position) dans la variable, et que Xi est la valeur associée à la i-ème position, alors le calcul de la moyenne, notée $\overline{X}$, peut être écrit tel que ci-dessous :

$$\overline{X} = \frac{1}{N} \sum_{i=1}^{N} Xi$$

Cette expression mathématique signifie que la moyenne s'obtient en additionnant ($\sum$) les valeurs allant de la position 1 à la position N de la variable, et en divisant le tout par le nombre total de valeurs N contenues dans la variable. Pour mieux comprendre, prenons par exemple une variable qui ne contiendrait que les cinq premières valeurs de la variable Sepal.Length du jeu de données `iris` et qu'on appelle `sample_iris`.

```
sample_iris <- iris$Sepal.Length[1:5]
sample_iris
```

```
## [1] 5.1 4.9 4.7 4.6 5.0
```

La moyenne de la variable `sample_iris` peut alors s'obtenir en divisant la somme des valeurs de la variable par le nombre de valeurs contenues dans la variable, qui est ici de 5 :

```
(5.1 + 4.9 + 4.7 + 4.6 + 5.0) / 5
```

```
## [1] 4.86
```

Évidémment, ce n'est pas très pratique de fonctionner comme cela. Aussi, R permet de calculer directement la moyenne avec la fonction `mean()` :

```
mean(x = sample_iris)
```

```
## [1] 4.86
```

Dans certains cas, il se peut qu'il y ait des valeurs manquantes dans la variable à étudier. Ces valeurs manquantes sont en principe notées **NA**. Introduisons une valeur manquante dans notre variable `sample_iris`, et essayons de calculer la moyenne à nouveau :

```
sample_iris[2] <- NA
sample_iris
```

```
## [1] 5.1  NA 4.7 4.6 5.0
```

```
mean(x = sample_iris)
```

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

Comme nous pouvons le voir ci-dessus, quand il y a une valeur manquante dans la variable, l'utilisation de la fonction `mean()` configurée par défaut renvoie la valeur NA, ce qui signifie que R n'a pas pu calculer de valeur moyenne, ce qui est normal car nous lui avons demandé de le faire en utilisant une valeur inconnue. Dans ce cas là, pour pouvoir faire le calcul de la moyenne seulement à partir des valeurs connues, il faut configurer la fonction pour que les valeurs manquantes ne soient pas considérées pour le calcul. L'argument à configurer dans ce cas là est `na.rm` en lui associant la valeur `TRUE`.

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

```
## [1] 4.85
```

La gestion des valeurs manquantes telle que nous venons de la voir s'effectue de la même manière avec beaucoup de fonctions dans R. Ainsi, il s'agira de fonctionner de la même manière avec la plupart des fonctions de base que nous pourrons rencontrer par la suite et qui seront concernées par ce genre de problème. Par ailleurs, si les exemples ci-dessus ont été réalisés à l'aide d'une variable isolée (i.e., ne faisant pas partie d'un tableau de données), c'est évidemment possible d'utiliser la fonction `mean()` directement à partir d'un tableau de données :

```
mean(x = iris$Sepal.Length)
```

```
## [1] 5.843333
```

La moyenne, c'est en quelque sorte le "centre de gravité" de la variable (Navarro, 2018). L'un de ses intérêts est que son calcul prend en compte

toutes les informations contenues dans la variable, ce qui est utile quand on a relativement peu de données (Navarro, 2018). En revanche, un inconvénient est qu'elle est très sensible aux valeurs extrêmes, et en particulier aux valeurs qui seraient particulièrement basses ou particulièrement élevées par rapport à la majorité des valeurs de la variable (il s'agirait ici d'*outliers*), et cela est d'autant plus vrai lorsque la taille de l'échantillon étudié est faible. Dans ce dernier cas, il y a donc un risque assez important que la moyenne ne représente pas bien la tendance centrale, c'est-à-dire la valeur ou la zone de valeurs où sont situées la majorité des observations. Ce risque existe aussi même avec des tailles d'échantillon relativement importantes lorsque la distribution est asymétrique, comme illustré sur la figure ci-dessous. Sur cette figure, on peut voir qu'avec une distribution gaussienne, la moyenne correspond parfaitement à la tendance centrale. En revanche, avec une distribution asymétrique à droite, on voit que la moyenne est "tirée vers la droite" par rapport à la tendance centrale sous l'effet des valeurs, certes moins nombreuses, mais d'une grandeur plus importante.

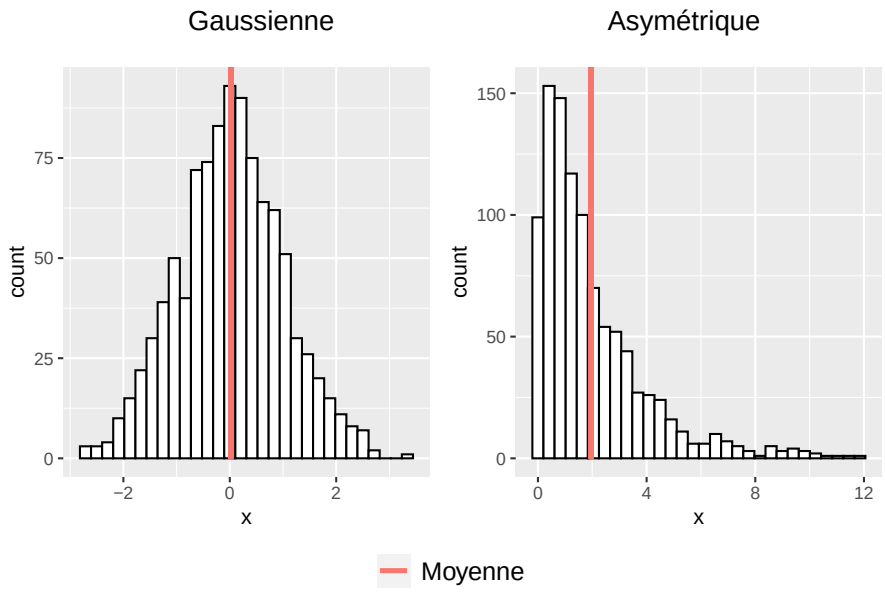


Figure 3.8: Moyenne d'une variable présentant une distribution asymétrique

La médiane

La médiane est le deuxième indice de position que l'on considère régulièrement lorsqu'il s'agit de résumer numériquement une variable quantitative. Pour l'obtenir, il faut d'abord classer les valeurs de la variable selon un ordre croissant. La médiane pour une variable de taille N est alors la valeur correspondant au rang $(N + 1) / 2$. Ainsi, la médiane désigne la valeur qui

sépare les valeurs de la variable en deux groupes d'observations de même taille (Chatellier & Durieux, 2003). Dans l'exemple ci-dessous, il y a cinq observations, et donc cinq valeurs, qui ont été triées par ordre croissant. La médiane est alors la valeur correspondant au rang (5 + 1) / 2 = 3, soit 4.9.

```
##   1   2   3   4   5
## 4.6 4.7 4.9 5.0 5.1
```

Dans le cas où le nombre d'observations contenues dans la variable étudiée serait un nombre pair, la médiane s'obtiendrait différemment. En effet, avec une variable qui contiendrait par exemple six valeurs, la médiane serait associée, selon la méthode expliquée ci-dessus, au rang (6 + 1) / 2 = 3.5, or ce rang, évidemment, n'existe pas. Dans ce cas, la médiane s'obtient en faisant la moyenne des deux nombres du milieu. Par exemple, ci-dessous, la médiane correspond à la moyenne des valeurs de la 3ème et de la 4ème observation, ce qui donne 4.95.

```
##   1   2   3   4   5   6
## 4.6 4.7 4.9 5.0 5.1 5.4
```

Dans R, la médiane d'une variable s'obtient facilement à l'aide de la fonction `median()`.

```
vec <- c(4.6, 4.7, 4.9, 5.0, 5.1, 5.4)
median(x = vec)
```

```
## [1] 4.95
```

Contrairement à la moyenne, la médiane prend donc en compte moins d'informations relatives aux données. Toutefois, en tant que valeur "du milieu", la médiane présente l'intérêt de ne pas être influencée par les valeurs extrêmes. En raison de cela, la médiane est susceptible de mieux refléter la tendance centrale que la moyenne en présence de faibles échantillons avec des *outliers*, ou en présence d'une forme de distribution asymétrique. Ce dernier cas est illustré sur la figure ci-dessous.

La moyenne rognée

Parfois, il est possible de rencontrer ce qu'on appelle la moyenne rognée. Le principe est ici de calculer la moyenne non pas en prenant en compte toutes les valeurs de la variable, mais en écartant un certain pourcentage des valeurs situées à l'extrémité basse et à l'extrémité haute du classement des valeurs de la variable. Cette procédure consiste à pouvoir calculer une moyenne qui ne serait pas influencée par des *outliers*. Pour pouvoir caculer une moyenne rognée, il faut à nouveau utiliser la fonction `mean()`, en précisant cette fois l'argument `trim` avec la valeur du pourcentage de données que l'on veut rogner aux extrémités de la variable :

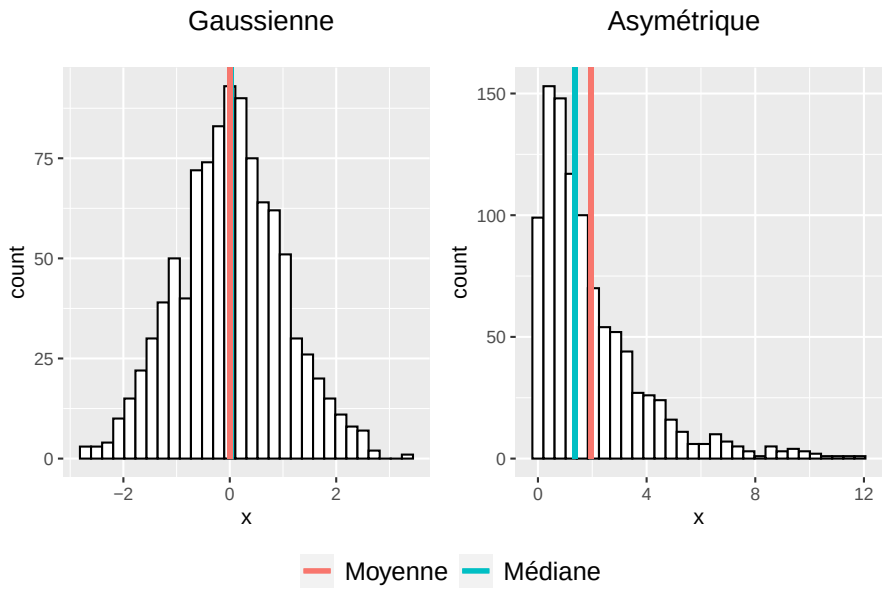


Figure 3.9: Médiane d'une variable présentant une distribution asymétrique

```
mean(x = iris$Sepal.Length, trim = 0.05)
```

Dans l'exemple ci-dessus, le code a été configuré pour rogner 5 % des observations situées aux extrémités de la variable. Notons que lorsque l'argument `trim` est mis à 0 (ce qui est son paramétrage par défaut), cela consiste à calculer la moyenne normale, et que lorsque l'argument `trim` est mis à 0.50, cela revient à calculer la médiane puisque la fonction supprime alors 50 % des observations de part et d'autre du milieu de la variable.

Le mode

Le mode désigne la valeur qui est la plus fréquemment retrouvée dans une variable. Il n'existe pas de fonction de base dans R pour pouvoir déterminer directement le mode et pour connaître le nombre de fois que le mode apparaît dans la variable. Toutefois, nous pouvons utiliser le package `lsr` crée par Danielle Navarro (2018) pour retrouver ces informations en présence d'une variable quantitative. Une fois le package `lsr` installé puis chargé, nous pouvons utiliser la fonction `modeOf()` pour déterminer le mode, et la fonction `maxFreq()` pour savoir à quelle fréquence revient le mode dans la variable.

```
library(lsr)
modeOf(x = iris$Sepal.Length)                    # Détermination du mode.
```

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

```
maxFreq(x = iris$Sepal.Length)                    # Détermination de la fréquence du m
```

```
## [1] 10
```

Bien que cela ne soit pas le cas dans l'exemple ci-dessus, il faut comprendre qu'il est à tout à fait possible d'avoir plusieurs modes si plusieurs valeurs reviennent à des fréquences maximales similaires dans la variable. Dans ce cas, la fonction `modeOf()` affichera les différentes valeurs de mode, et la fonction `maxFreq()` continuera de n'afficher qu'une seule valeur de fréquence puisque par définition, le mode désigne la valeur associée à la fréquence d'apparition maximale dans la variable, or il ne peut n'y avoir qu'une seule fréquence maximale... Cela signifie quelque part qu'une variable peut contenenir autant de modes que de valeurs si chaque valeur n'est représentée qu'une seule fois dans la variable. Cet inconvénient est probablement l'une des raisons pour lesquelles le mode n'est que très peu utilisé, si ce n'est jamais utilisé, pour décrire la tendance centrale d'une variable quantitative.

3.1.3 Les indices de dispersion

Les indices de dispersion permettent de rendre compte de la manière selon laquelle les observations sont étalées, ou réparties, autour des indices de position. Plusieurs statistiques sont disponibles pour caractériser la dispersion, à savoir : l'**étendue**, l'**écart-type**, et l'**intervalle interquartile**.

L'étendue

L'étendue est la mesure la plus simple de la dispersion des données contenues dans une variable. Elle est exprimée avec la plus petite valeur (minimum) et la plus grande valeur (maximum) observée, ou alors parfois avec la différence de ces deux valeurs. Par exemple, dans la variable ci-dessous dont les données ont été classées en ordre croissant, le minimum est 4.5, le maximum est 20.2, et l'étendue peut être donnée par l'intervalle [4.5 - 20.2]. Pour obtenir ces différents résultats dans R, il est possible d'utiliser les fonctions `min()`, `max()`, et `range()`. L'amplitude de l'intervalle serait ici de : 20.2-4.5 = 15.7.

```
vec <- c(4.5, 7.8, 10.8, 13.9, 20.2)
min(vec)
```

```
## [1] 4.5
```

```
max(vec)
```

```
## [1] 20.2
```

```
range(vec)
```

```
## [1]  4.5 20.2
```

L'écart-type

L'écart-type est une statistique qui donne une idée de la mesure selon laquelle les valeurs de la variable sont éloignées de la moyenne. Pour calculer l'écart-type, il faut en réalité d'abord calculer la variance σ^2, dont le calcul est le suivant :

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(Xi - \overline{X})^2$$

Cette formule signifie que pour obtenir la variance, il faut d'abord faire la somme des carrés des différences entre chaque valeur et la moyenne de la variable. Cela fait, la variance s'obtient en divisant cette somme de carrés par le nombre N de valeurs de la variable. L'écart-type σ, c'est alors la racine carrée de la variance :

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Xi - \overline{X})^2}$$

Ces calculs sont valides lorsque l'on a en sa possession les données de toute la population que l'on souhaite étudier. Toutefois, lorsque l'on a en sa possession des données issues seulement d'un échantillon de la population, ces calculs biaisent les estimations de la variance et de l'écart-type correspondant à la population étudiée. Cette notion de biais traduit le fait que lorsqu'on répète un grand nombre de fois le calcul de la variance ou de l'écart-type à partir, à chaque fois, d'échantillons de population différents, on a en moyenne un décalage entre la valeur de l'estimation et la "vraie" valeur de la variance et de l'écart-type. Ce décalage systématique est tel qu'il convient dans ce cas là de diviser la somme des carrés des différences $(Xi - \overline{X})$ par *N-1* plutôt que par N (Grenier, 2007). La formule de l'écart-type non biaisé, noté s, est alors la suivante :

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(Xi - \overline{X})^2}$$

L'écart-type est la mesure de dispersion classiquement associée à la moyenne. Si on note une moyenne $\overline{X}$ et un écart-type s, alors le résumé d'une variable à l'aide de ces statistiques s'écrit comme suit : $\overline{X} \pm s$. Lorsque l'écart-type est divisé par la moyenne arithmétique de la variable, on obtient une valeur appelée **coefficient de variation**. Avec le logiciel R, les fonctions pour calculer la variance et l'écart-type non biaisés sont respectivement `var()` et `sd()`.

```
vec <- c(4.5, 7.8, 10.8, 13.9, 20.2)
var(x = vec)
```

```
## [1] 36.153
```

```
sd(x = vec)
```

```
## [1] 6.012736
```

L'intervalle interquartile

L'intervalle interquartile désigne l'étendue entre le premier (Q1) et le troisième (Q3) quartile d'une variable. Comme expliqué auparavant dans le cadre de la boîte à moustaches, Q1 et Q3 désignent respectivement les valeurs en-dessous desquelles 25 % et 75 % des observations de la variable se trouvent (Chatellier & Durieux, 2003). Pour un échantillon de taille N, la procédure classique pour calculer les quartiles est différente selon que le rapport N / 4 est un nombre entier ou non. Lorsque ce rapport n'est pas un nombre entier, Q1 est la valeur correspondant au rang immédiatement supérieur à N / 4. Par exemple, pour la variable ci-dessous, qui a une taille N de 5 valeurs, le rapport N / 4 est égal à 1.25. Q1 est donc la valeur correspond au rang directement supérieur, c'est-à-dire au rang 2, qui est ici la valeur 7.8.

```
##    1    2    3    4    5
##  4.5  7.8 10.8 13.9 20.2
```

Lorsque le rapport N / 4 est un nombre entier, Q1 correspond à la moyenne des valeurs associées respectivement aux rangs N / 4 et (N / 4) + 1. Par exemple, pour la variable ci-dessous, qui a une taille N de 8 valeurs, le rapport N / 4 est à égal 2. Q1 est donc la moyenne des valeurs correspondant au rang 2 et au rang 3 (i.e., les valeurs 7.8 et 10.8), qui équivaut ici à 9.3.

```
##    1    2    3    4    5    6    7    8
##  4.5  7.8 10.8 13.9 20.2 25.6 37.5 43.9
```

Table 3.1: Comparaison des quantiles obtenus selon différentes configurations de la fonction 'quantile()'

Quantile	Type_2	Type_7
0.25	9.30	10.050
0.75	31.55	28.575

La démarche demeure la même pour déterminer Q3, à ceci près qu'on utilise le nombre $3N$ et non plus le nombre N pour les calculs (Labreuche, 2010). Cette méthode de calcul, classique, est en principe à privilégier en présence d'une variable discrète. Si l'on souhaite obtenir les quartiles selon cette méthode avec le logiciel R, il faut utiliser la fonction `quantile()` de la manière suivante :

```
quantile(x = vec, probs = c(0.25, 0.75), type = 2)
```

```
##   25%   75%
##  9.30 31.55
```

On remarque ici que la fonction `quantile()` a plusieurs arguments. L'argument `probs` désigne les quantiles que l'on souhaite obtenir. Ici, le quantile 0.25 correspond à Q1, et le quantile 0.75 correspond à Q3. L'argument `type` permet de configurer le type de calcul à effectuer pour obtenir les valeurs des quantiles recherchés. L'indication du chiffre 2 pour l'argument `type` permet d'obtenir les quantiles selon la méthode de calcul présentée ci-dessus, qui, comme nous l'avons précisé, est dédiée à l'étude d'une variable quantitative discrète. Par défaut, en revanche, la fonction `quantile()` utilise le chiffre 7 pour l'argument `type`, ce qui renvoie à une méthode de calcul des quantiles qui serait davantage pertinente pour étudier des variables quantitatives continues. Comparons les résultats obtenus avec les deux méthodes de calcul :

On remarque que les résultats de la fonction `quantile()` sont différents selon la configuration de l'argument `type`. Le choix de la configuration est donc important. Pour comprendre comment R a calculé les valeurs associées aux quantiles 0.25 et 0.75 dans le cadre de la seconde méthode (i.e., avec `type` = 7), regardons le tableau ci-dessous.

Le tableau montre les données sur lesquelles R s'est appuyé pour déterminer les valeurs des quantiles recherchés (i.e., les quantiles 0.25 et 0.75 pour Q1 et Q3, respectivement). Les données du tableau sont bien celles relatives à notre variable `vec`, dont on peut reconnaître les valeurs dans la colonne de droite du tableau. La colonne "Quantile" montre les fractions (ou portions) de la variable `vec` associées aux valeurs de la variable compte tenu de leurs rangs respectifs. Par exemple, la valeur 25.6, dont le rang est 6, correspond au quantile 0.71 (approximativement). Cela veut dire que 71 % des observations

Table 3.2: Quantiles d'une variable quantitative continue

Rang	Quantile	Valeur
1	0.0000000	4.5
2	0.1428571	7.8
3	0.2857143	10.8
4	0.4285714	13.9
5	0.5714286	20.2
6	0.7142857	25.6
7	0.8571429	37.5
8	1.0000000	43.9

ont une valeur inférieure ou égale à 25.6. Il faut savoir qu'il existe en réalité plusieurs manières de déterminer la valeur du quantile que représente chaque valeur. Dans le cas présent, le quantile représenté, que l'on va noter q, a été déterminé selon la formule suivante :

$$q = (k - 1)/(N - 1)$$

Dans le calcul ci-dessus, k désigne le rang de la valeur considérée, et N désigne la taille de la variable étudiée (i.e., le nombre total de valeurs). Comme on peut le voir dans le tableau ci-dessus, cette méthode de calcul conduit nécessairement à attribuer le quantile 0 à la valeur de rang 1, et la quantile 1 à la valeur de rang N. Lorsque le nombre de valeurs fait que les quantiles 0.25 et 0.75 n'existent pas, R réalise une interpolation de la valeur correspondant au quantile recherché, cela à partir des quantiles qui existent et qui encadrent le quantile recherché, ainsi qu'à partir des valeurs correspondant à ces quantiles. Dans le cas présent, il s'agit plus précisément d'une interpolation linéaire. Voyons ci-dessous en quoi cela consiste.

La figure représente les valeurs de la variable en fonction des quantiles qui leur correspondent. Les segments de couleur noire montrés sur la figure représentent les droites d'équation utilisées pour le calcul des valeurs correspondant aux quantiles 0.25 (Q1) et 0.75 (Q3), qui ne sont pas représentés initialement dans la variable étudiée. Ces droites d'équation relient les points dont les abscisses sont celles qui encadrent directement les quantiles recherchés. Ainsi, pour trouver la valeur correspondant au quantile 0.25, il a suffi de résoudre l'équation y = 21x + 4.8, en remplaçant x par 0.25. De manière analogue, pour trouver la valeur correspondant au quantile 0.75, il a suffi de résoudre l'équation y = 83.3x - 33.9 en remplaçant x par 0.75. Les solutions de ces équations sont montrées en rouge sur la partie gauche de la figure. On retrouve bien les valeurs associées aux quantiles recherchés et qui avaient été initialement obtenues avec la configuration par défaut de la fonction `quantile()`.

Les quartiles Q1 et Q3 sont les mesures de dispersion classiquement associées à la médiane. Si on note une médiane m et l'intervalle interquartile (Q1 - Q3),

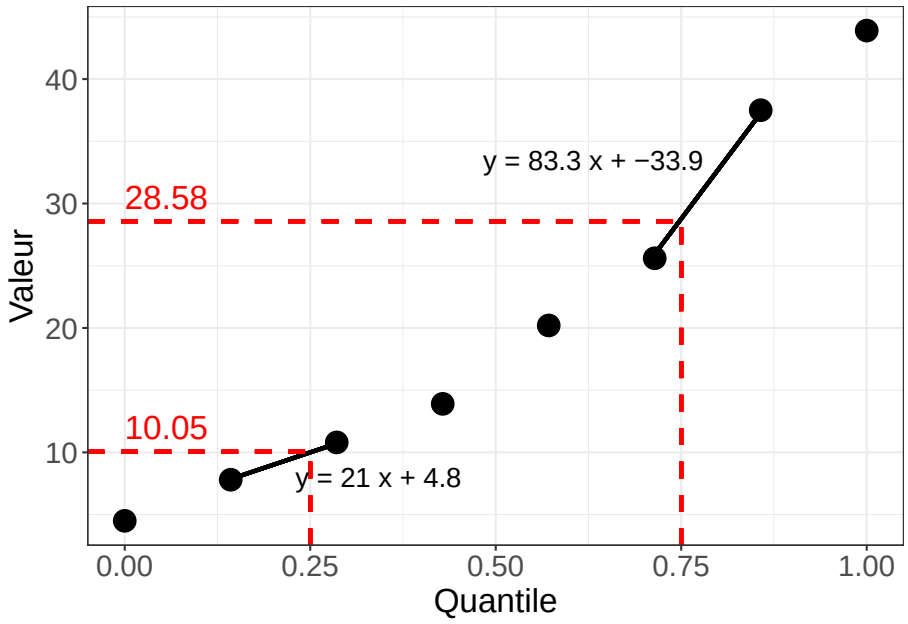


Figure 3.10: Calcul des quartiles avec une variable quantitative continue

alors le résumé d'une variable à l'aide de ces statistiques s'écrit comme suit : m (Q1 - Q3).

3.1.4 Les indices d'asymétrie et d'aplatissement

Le coefficient d'asymétrie (skewness)

Le fait qu'une distribution soit asymétrique désigne le fait que les observations sont réparties de manière inégale de part et d'autre du milieu de la distribution. L'indice statistique qui permet de rendre compte du niveau d'asymétrie est le **coefficient d'asymétrie**, ou *skewness* en anglais. Ce coefficient peut être obtenu à l'aide de la fonction `skewness()` du package `e1071`, qui n'existe pas dans la base de R et qu'il convient d'installer et de charger pour l'utiliser.

```
library(e1071)
skewness(x = iris$Sepal.Width, type = 3)
```

```
## [1] 0.3126147
```

Comme on peut le voir dans l'aide associée à la fonction `skewness()`, il existe en réalité plusieurs méthodes de calcul du coefficient d'asymétrie. La méthode

de `type` 3, qui est celle configurée par défaut pour cette fonction, consiste à faire le calcul suivant :

$$\gamma_1 = \frac{1}{s^3} \frac{\sum_{i=1}^{N} (Xi - \overline{X})^3}{N}$$

Dans ce calcul, s désigne l'écart-type non biaisé de la variable, et N désigne la taille de la variable. Avec cette méthode, on obtient un coefficient négatif lorsque la distribution est asymétrique à gauche (longue queue vers la gauche), un coefficient de 0 lorsque la distribution est parfaitement symétrique, et un coefficient positif lorsque la distribution est asymétrique à droite (longue queue vers la droite). Ceci est illustré sur la figure ci-dessous.

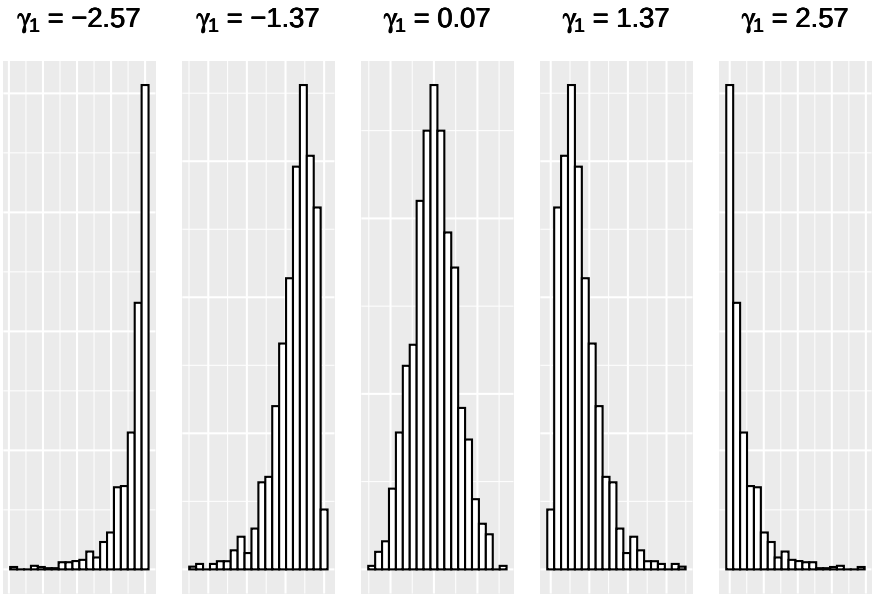


Figure 3.11: Valeur du Skewness selon la forme de la distribution

Pour aller plus loin... Joanes et Gill (1998) ont montré que lorsqu'il s'agit d'estimer le degré d'asymétrie de la distribution relative à la population étudiée, et cela à partir de l'échantillon observé, certaines méthodes de calcul du coefficient d'asymétrie peuvent être plus fiables que d'autres.

Dans le cas où la distribution des valeurs dans la population étudiée suivrait une loi normale, la méthode par défaut présentée ci-dessus serait la plus fiable pour estimer le niveau d'asymétrie lorsque l'échantillon observé est de petite taille ($N < 50$). Cependant, avec des échantillons de grande taille, les méthodes se valeraient.

Dans le cas où la distribution des valeurs de la population étudiée ne suivrait pas une loi normale, et qu'elle s'avèrerait très asymétrique, la méthode de `type` 2 proposée avec la fonction `skewness()` serait la plus fiable, particulièrement en présence d'échantillons de petite taille.

Le coefficient d'applatissement (kurtosis)

Le fait qu'une distribution soit aplatie désigne le fait que la forme de la distribution présente une courbure relativement plate avec des queues de distribution relativement courtes. On parle alors de distribution platycurtique. À l'inverse, lorsque la distribution est pointue avec des queues plus longues, on parle de distribution leptocurtique. L'indice statistique qui permet de rendre compte du degré d'aplatissement est le **coefficient d'aplatissement**, ou *kurtosis* en anglais. Ce coefficient peut être obtenu à l'aide de la fonction `kurtosis()` du package `e1071`.

```
library(e1071)
kurtosis(x = iris$Sepal.Width, type = 3)
```

```
## [1] 0.1387047
```

Comme on peut le voir dans l'aide associée à la fonction `kurtosis()`, il existe en réalité plusieurs méthodes de calcul du coefficient d'aplatissement. La méthode de `type` 3, qui est celle configurée par défaut pour cette fonction, consiste à faire le calcul suivant :

$$\gamma_2 = \frac{1}{s^4}\frac{\sum_{i=1}^{N}(Xi - \overline{X})^4}{N} - 3$$

Dans ce calcul, s désigne l'écart-type non biaisé de la variable, et N désigne la taille de la variable. Avec cette méthode, on obtient un coefficient négatif lorsque la distribution est particulièrement aplatie par rapport à une distribution suivant une loi normale (distribution platycurtique), un coefficient de 0 lorsque la distribution suit une loi normale (distribution mésocurtique), et un coefficient positif lorsque la distribution est particulièrement pointue par rapport à une loi normale (distribution leptocurtique). Ceci est illustré sur la figure ci-dessous.

Pour aller plus loin... Comme avec le coefficient d'asymétrie, Joanes et Gill (1998) ont montré que lorsqu'il s'agit d'estimer le degré d'aplatissement de la distribution relative à la population étudiée, et cela à partir de l'échantillon observé, certaines méthodes de calcul du coefficient d'aplatissement peuvent être plus fiables que d'autres.

Dans le cas où la distribution des valeurs dans la population étudiée suivrait une loi normale, la méthode de `type` 1 proposée avec la fonction `kurtosis()`

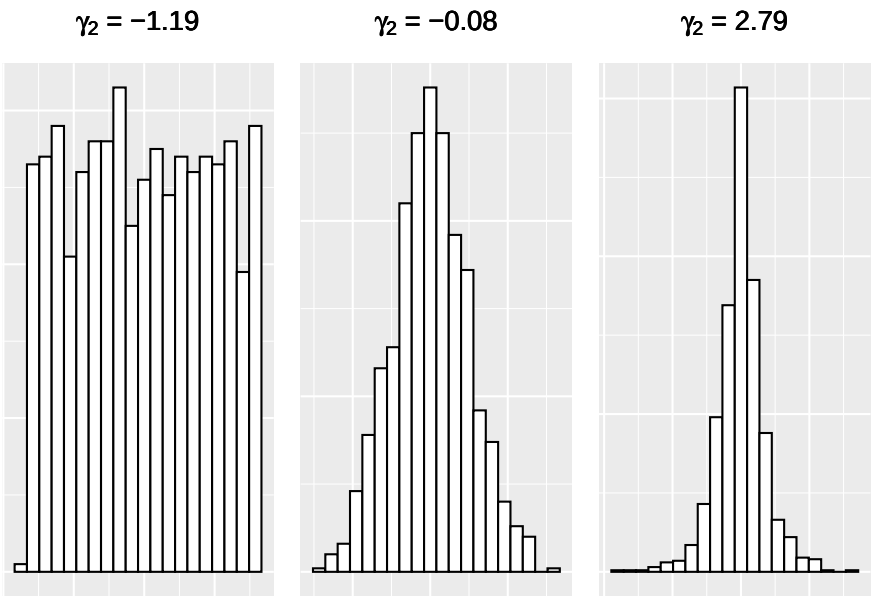


Figure 3.12: Valeur du Kurtosis selon la forme de la distribution

serait la plus fiable pour estimer le niveau d'aplatissement lorsque l'échantillon observé est de petite taille ($N < 50$). Cependant, la méthode par défaut présentée plus haut fournirait des résultats relativement proches de ceux obtenus avec la méthode de `type` 1. De plus, avec des échantillons de grande taille, toutes les méthodes se valeraient.

Dans le cas où la distribution des valeurs de la population étudiée ne suivrait pas une loi normale, et s'avèrerait très asymétrique, la méthode de `type` 2 proposée avec la fonction `kurtosis()` serait la plus fiable, particulièrement en présence d'échantillons de petite taille.

3.1.5 Fonctions pour obtenir un récapitulatif des statistiques descriptives

Il existe plusieurs fonctions pour avoir une vue d'ensemble des statistiques généralement utilisées pour explorer et résumer une variable quantitative. Une fonction particulièrement intéressante est la fonction `describe()` du package `psych`.

```
library(psych)
```

La fonction `describe()` peut être utilisée sur une variable donnée :

```
describe(x = iris$Sepal.Width, quant = c(0.25, 0.75))
```

```
## [150 obs.]
## numeric: 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
## min: 2 - max: 4.4 - NAs: 0 (0%) - 23 unique values
```

La fonction `describe()` peut être aussi utilisée sur un jeu de données entier. (Attention, les résumés numériques fournis pour les variables qualitatives n'auront pas de sens ; les variables qualitatives détectées sont indiquées avec un astérisque dans le tableau de résultats.)

```
describe(x = iris, quant = c(0.25, 0.75))
```

```
## NULL
```

On retrouve la plupart des statistiques que nous avons vues jusqu'à présent, notamment le *skewness* et le *kurtosis* qui ont été ici calculés avec la méthode par défaut du package `e1071`. Pour changer la méthode de calcul de ces deux coefficients, il suffit de modifier l'argument `type` de la fonction, comme dans le cadre de l'utilisation du package `e1071` et des fonctions `skewness()` et `kurtosis()` associées. On notera cependant qu'il ne semble pas possible de modifier la méthode de calcul des quantiles, qui sont ici calculés selon la méthode par défaut configurée telle qu'avec la fonction `quantile()`.

Enfin, il est aussi possible d'obtenir ces récapitulatifs numériques en fonction des modalités d'une variable qualitative du jeu de données, grâce à la fonction `describeBy()` du package `psych`. Dans l'exemple ci-dessous, la variable qualitative est indiquée grâce à l'argument `group`.

```
describeBy(x = iris, quant = c(0.25, 0.75), group = iris$Species)
```

```
##
##  Descriptive statistics by group
## group: setosa
##              vars  n mean   sd median trimmed  mad min max range skew
## Sepal.Length    1 50 5.01 0.35    5.0    5.00 0.30 4.3 5.8   1.5 0.11
## Sepal.Width     2 50 3.43 0.38    3.4    3.42 0.37 2.3 4.4   2.1 0.04
## Petal.Length    3 50 1.46 0.17    1.5    1.46 0.15 1.0 1.9   0.9 0.10
## Petal.Width     4 50 0.25 0.11    0.2    0.24 0.00 0.1 0.6   0.5 1.18
## Species*        5 50 1.00 0.00    1.0    1.00 0.00 1.0 1.0   0.0  NaN
##              kurtosis   se Q0.25 Q0.75
## Sepal.Length    -0.45 0.05   4.8  5.20
## Sepal.Width      0.60 0.05   3.2  3.68
```

```
## Petal.Length     0.65 0.02   1.4  1.58
## Petal.Width      1.26 0.01   0.2  0.30
## Species*          NaN 0.00   1.0  1.00
## ------------------------------------------------------------
## group: versicolor
##              vars  n mean   sd median trimmed  mad min max range  skew
## Sepal.Length    1 50 5.94 0.52   5.90    5.94 0.52 4.9 7.0   2.1  0.10
## Sepal.Width     2 50 2.77 0.31   2.80    2.78 0.30 2.0 3.4   1.4 -0.34
## Petal.Length    3 50 4.26 0.47   4.35    4.29 0.52 3.0 5.1   2.1 -0.57
## Petal.Width     4 50 1.33 0.20   1.30    1.32 0.22 1.0 1.8   0.8 -0.03
## Species*        5 50 2.00 0.00   2.00    2.00 0.00 2.0 2.0   0.0   NaN
##              kurtosis   se Q0.25 Q0.75
## Sepal.Length    -0.69 0.07  5.60   6.3
## Sepal.Width     -0.55 0.04  2.52   3.0
## Petal.Length    -0.19 0.07  4.00   4.6
## Petal.Width     -0.59 0.03  1.20   1.5
## Species*          NaN 0.00  2.00   2.0
## ------------------------------------------------------------
## group: virginica
##              vars  n mean   sd median trimmed  mad min max range  skew
## Sepal.Length    1 50 6.59 0.64   6.50    6.57 0.59 4.9 7.9   3.0  0.11
## Sepal.Width     2 50 2.97 0.32   3.00    2.96 0.30 2.2 3.8   1.6  0.34
## Petal.Length    3 50 5.55 0.55   5.55    5.51 0.67 4.5 6.9   2.4  0.52
## Petal.Width     4 50 2.03 0.27   2.00    2.03 0.30 1.4 2.5   1.1 -0.12
## Species*        5 50 3.00 0.00   3.00    3.00 0.00 3.0 3.0   0.0   NaN
##              kurtosis   se Q0.25 Q0.75
## Sepal.Length    -0.20 0.09  6.23  6.90
## Sepal.Width      0.38 0.05  2.80  3.18
## Petal.Length    -0.37 0.08  5.10  5.88
## Petal.Width     -0.75 0.04  1.80  2.30
## Species*          NaN 0.00  3.00  3.00
```

3.1.6 Quelles statistiques choisir pour résumer une variable quantitative dans un rapport ?

Les statistiques les plus couramment utilisées pour résumer une variable quantitative sont les paramètres de position (moyenne et médiane principalement) en lien avec les paramètres de dispersion correspondants. Aucun paramètre de position ne surpasse les autres dans toutes les situations. Le choix du paramètre de position, en lien avec le paramètre de dispersion associé, dépend de l'objectif de l'analyse.

Lorsqu'il s'agit de simplement décrire une distribution des données obtenues à titre exploratoire, certains auteurs proposent d'utiliser les trois paramètres de position (moyenne, médiane, mode), en sachant que la médiane, d'un point de

vue purement descriptif, peut être le paramètre le plus adapté dans de nombreuses situations (Gonzales & Ottenbacher, 2001). Dans le cas où la distribution s'avèrerait plutôt gaussienne, la moyenne et l'écart-type sont intéressants car dans ce cas, on sait que :

1. Approximativement **68.3 %** des observations sont comprises dans l'intervalle $[\overline{X}$ - 1 s ; $\overline{X}$ + 1 $s]$,
2. Approximativement **95.5 %** des observations sont comprises dans l'intervalle $[\overline{X}$ - 2 s ; $\overline{X}$ + 2 $s]$,
3. Approximativement **99.7 %** des observations sont comprises dans l'intervalle $[\overline{X}$ - 3 s ; $\overline{X}$ + 3 $s]$.

Ceci est illustré sur la figure ci-dessous.

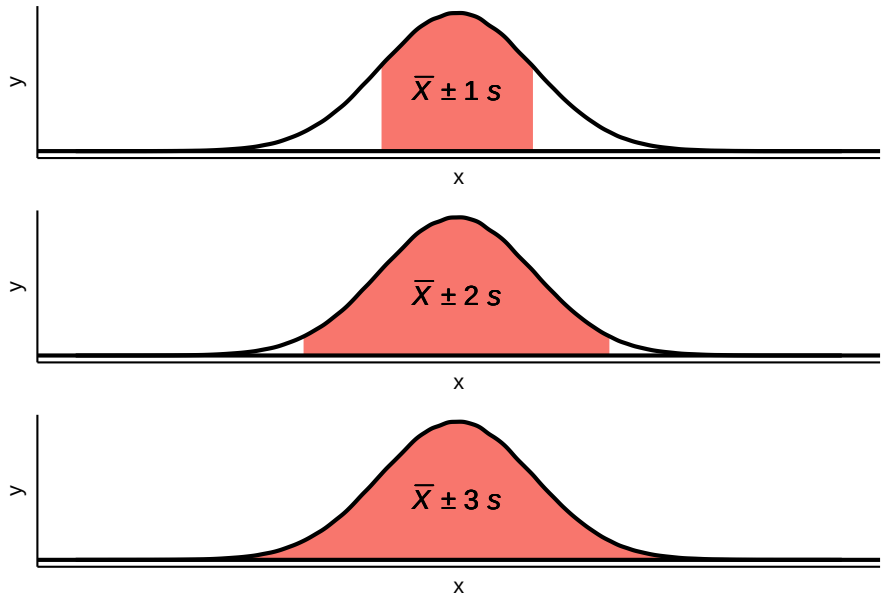


Figure 3.13: Proportions des observations incluses dans différents intervalles liés à des multiples de l'écart-type

Lorsqu'il s'agit plus précisément de vouloir renseigner sur la tendance centrale relative à l'échantillon étudié, le choix dépend de la forme de la distribution observée. Lorsque la distribution est gaussienne, la moyenne, la médiane, et le mode, sont similaires et donc se valent. Toutefois, lorsque la distribution est asymétrique et unimodale (i.e., avec un seul pic), le mode reflètera mieux la tendance centrale. De plus, lorsque la distribution est asymétrique, la médiane aura tendance à mieux représenter la tendance centrale que la moyenne

(Rousselet & Wilcox, 2020). Notons que dans certains cas où la distribution semble asymétrique, la médiane peut se retrouver malgré tout plus éloignée du mode que la moyenne, comme illustré dans l'exemple ci-dessous emprunté à Gonzales et al. (2001).

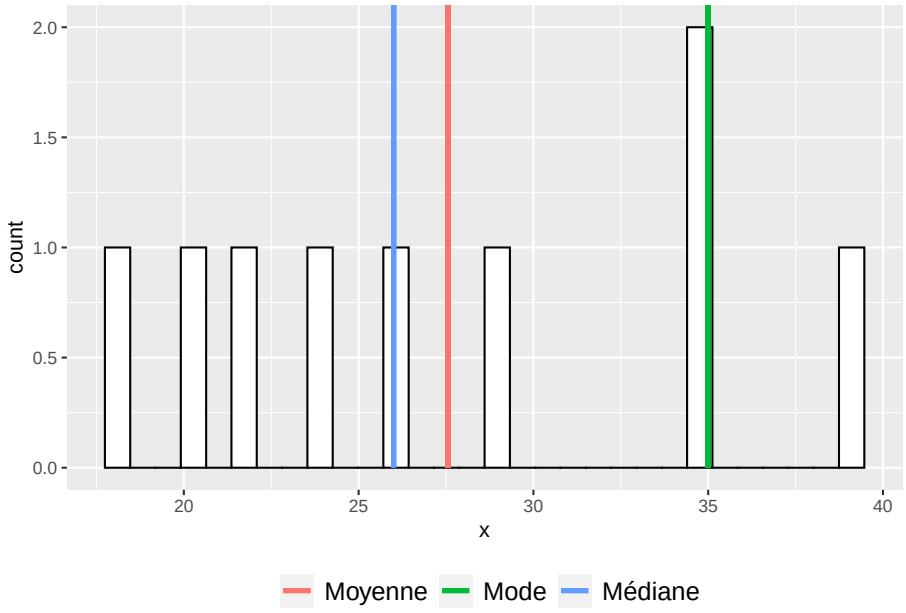


Figure 3.14: Moyenne et médiane dans le cadre d'une distribution asymétrique

3.2 Variables qualitatives

3.2.1 Visualiser la distribution de la variable

Comme dans le cadre de variables quantitatives, une bonne pratique est de visualiser graphiquement la distribution de la variable. Ici, il s'agit plus précisément de prendre connaissance des effectifs correspondant aux différentes modalités de la variable. Une manière rapide de procéder pour cela est d'utiliser la fonction `ggplot()` et la fonction `geom_bar()`. Illustrons cela avec le jeu de données `diamonds`, qui contient notamment la variable qualitative `color`.

```
# Aperçu du jeu de données
diamonds
```

```
## # A tibble: 53,940 x 10
```

```
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7 0.24  Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8 0.26  Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  9 0.22  Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 10 0.23  Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

```
# Visualisation de la distribution de la variable color
diamonds %>%
  ggplot(aes(x = color)) +
    geom_bar()
```

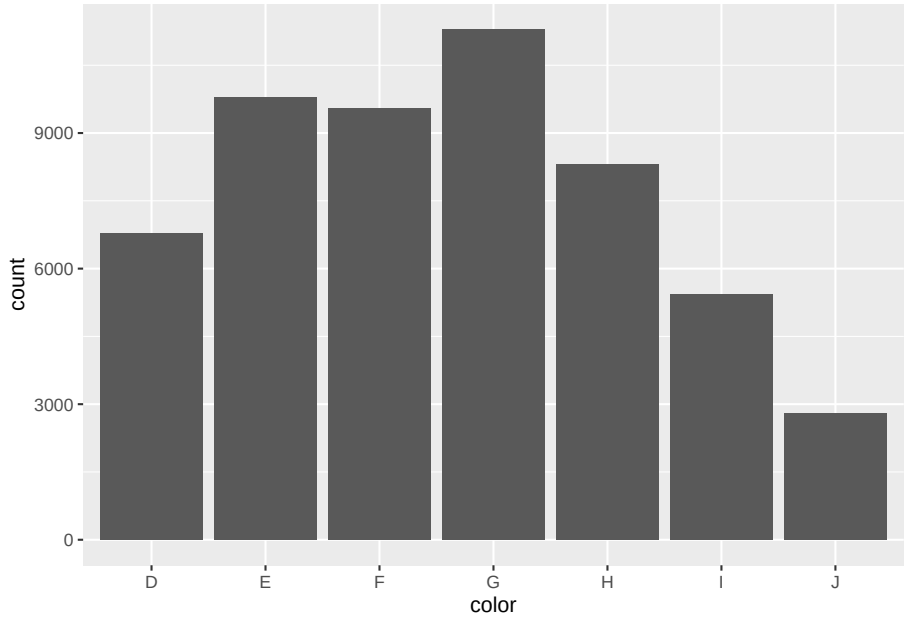


Figure 3.15: Exemple de diagramme en barres

Bien que rapide, cette manière de procéder devient vite limitée lorsque l'on veut améliorer le graphique, tel qu'en ajoutant les valeurs des effectifs au-dessus des barres ou encore en changeant l'ordre d'apparition des barres.

Pour gagner en capacité de modification du graphique, on peut d'abord passer par une étape intermédiaire consistant à créer un mini-jeu de données où la variable `color` serait déjà résumée à l'aide des fonctions `group_by()` et `summarize()` du package `dplyr`, de telle sorte à n'avoir que la valeur de l'effectif en regard de chaque modalité. Ceci est montré dans le code ci-dessous. Si l'on poursuit le code avec les fonctions `ggplot()` et `geom_bar()`, on arrive alors au même résultat que précédemment. (On note qu'il a fallu adapter l'argument `stat` de la fonction `geom_bar()` pour que le graphique montre bien en ordonnées la valeur de la variable nouvellement appelée `count`, qui comprend les effectifs de chaque modalité.)

```
diamonds %>%
  group_by(color) %>%
  summarize(count = n())
```

```
## # A tibble: 7 x 2
##   color count
##   <ord> <int>
## 1 D      6775
## 2 E      9797
## 3 F      9542
## 4 G     11292
## 5 H      8304
## 6 I      5422
## 7 J      2808
```

```
diamonds %>%
  group_by(color) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = color, y = count)) +
     geom_bar(stat = "identity")
```

Certes, pour le moment, cette seconde procédure n'a fait que rajouter des étapes par rapport à la première procédure. Toutefois, en reprenant la logique de la seconde procédure, on peut à présent créer des graphiques en barres plus élaborés relativement facilement.

```
# Diagramme en barres avec les effectifs affichés à proximité des barres
g1 <-
  diamonds %>%
  group_by(color) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = color, y = count)) +
     geom_bar(stat = "identity") +
```

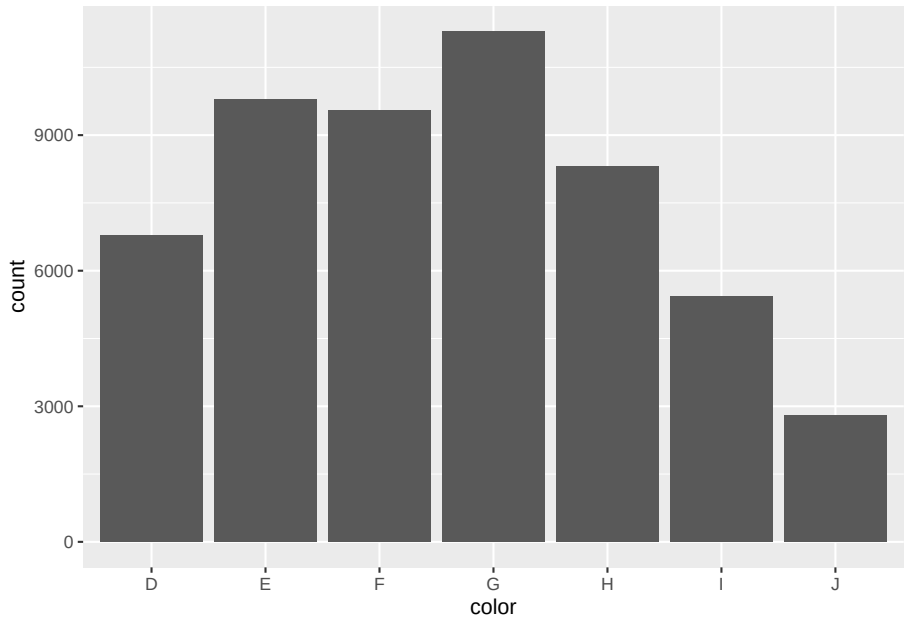


Figure 3.16: Diagramme en barres obtenu avec une seconde méthode d'écriture du code

```
    geom_text(aes(label = count, x = color, y = count), size = 2, nudge_y = 700) +
    ggtitle("g1")

# Diagramme en barres réorganisées manuellement
g2 <-
  diamonds %>%
  group_by(color) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = fct_relevel(color, "J", "I", "H", "G", "F", "E", "D"), y = count)) +
    geom_bar(stat = "identity") +
    xlab("color") +
    ggtitle("g2")

# Diagramme en barres réorganisées en ordre croissant
g3 <-
  diamonds %>%
  group_by(color) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = fct_reorder(color, count), y = count)) +
    geom_bar(stat = "identity") +
    xlab("color") +
```

```
    ggtitle("g3")

# Diagramme en barres réorganisées en ordre décroissant
g4 <-
  diamonds %>%
  group_by(color) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = fct_reorder(color, -count), y = count)) +
     geom_bar(stat = "identity") +
     xlab("color") +
     ggtitle("g4")

# Diagramme en barres pivoté
g5 <-
  diamonds %>%
  group_by(color) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = fct_reorder(color, -count), y = count)) +
  geom_bar(stat = "identity") +
     coord_flip() +
     ggtitle("g5")

# Dotplot pivoté
g6 <-
  diamonds %>%
  group_by(color) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = fct_reorder(color, -count), y = count)) +
     geom_point() +
     coord_flip() +
     ggtitle("g6")
```

Dans le code montré ci-dessus, on remarque que le réagencement manuel des barres (cf. g2) a pu être réalisé grâce à la fonction `fct_relevel()` du package `forcats`. Pour les autres diagrammes, la réorganisation des barres en ordre croissant ou décroissant sur la base de la valeur de l'effectif (`count`) a pu se faire grâce à la fonction `fct_reorder()` du package `forcats`. Les noms des arguments de ces fonctions n'ont pas été indiqués pour alléger le code. Le plus important, c'est d'indiquer en premier argument la variable dont l'ordre d'apparition des modalités doit être réorganisé (il s'agissait de la variable `color` dans cet exemple). En second argument, il convient d'indiquer la logique de réorganisation de l'apparition des modalités (ce qui a été fait sur la base des valeurs de la variable `count` dans les exemples ci-dessus utilisant la fonction `fct_reorder()`).

Toujours dans le code montré ci-dessus, on peut voir aussi, dans la fonction

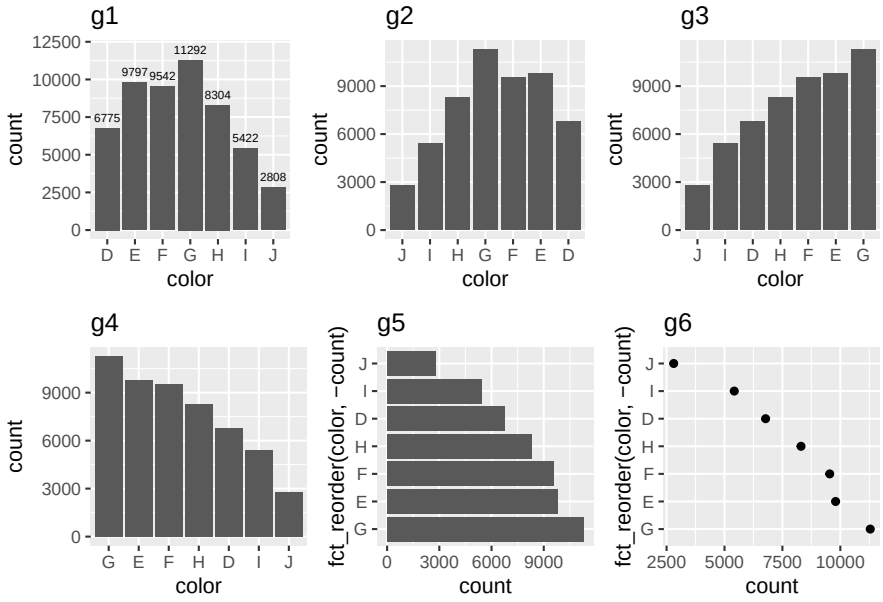


Figure 3.17: Différentes sortes de diagrammes en barres

`geom_text()` du graphique g1, la présence de l'argument `nudge_y`. Cet argument permet de régler le décalage entre le haut de la barre et le texte. L'ordre de grandeur du chiffre indiqué est celui de la variable indiquée en ordonnées.

On observe que la réorganisation des barres selon un ordre croissant ou décroissant clarifie l'information délivrée par le graphique. Toutefois, cette réorganisation est en principe surtout recommandée pour des variables qualitative nominales, c'est-à-dire des variables pour lesquelles il n'existe pas un ordre naturel des modalités. Lorsqu'il existe un ordre naturel des modalités, comme c'est le cas pour des variables qualitatives ordinales, les modalités devraient préférentiellement suivre leur ordre naturel.

En plus des effectifs, il est aussi possible de prendre connaissance de la distribution à l'aide des proportions, c'est-à-dire des ratios entre les effectifs liés aux différentes modalités et l'effectif total. Les proportions peuvent être visualisées avec un diagramme circulaire (i.e., un camembert), avec un diagramme en barres empilées, ou avec des barres disposées côte-à-côte. Les diagrammes circulaires et avec barres empilées mettent en avant le fait que les parties individuelles étudiées font partie d'un même ensemble. Les diagrammes circulaires peuvent être utilisés efficacement à cet effet lorsqu'ils montrent des fractions simples telles qu'un quart, un tiers, ou une moitié. Toutefois, les parties individuelles sont plus facilement comparables lorsqu'on

utilise des barres mises côte-à-côte, comme montré ci-avant. Les diagrammes avec barres empilées sont quant à eux difficiles à comprendre lorsqu'il s'agit de n'étudier qu'une seule variable qualitative (Wilke, 2018).

```
# Diagramme en barres empilées montrant les effectifs
g1 <-
  diamonds %>%
  group_by(color) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = "", y = count, fill = color)) +
  geom_bar(stat = "identity", na.rm = TRUE) +
  geom_text(aes(label = count), size = 2, position = position_stack(vjust = 0.5)) +
  ggtitle("g1")

# Diagramme en barres empilées montrant les proportions
g2 <-
  diamonds %>%
  group_by(color) %>%
  summarize(count = n() / length(diamonds$color) * 100) %>%
  ggplot(aes(x = "", y = count, fill = color)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = paste0(round(count, digits = 2), " %")), size = 2, position = p
  ggtitle("g2")

# Diagramme circulaire montrant les effectifs
g3 <-
  diamonds %>%
  group_by(color) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = "", y = count, fill = color)) +
  geom_bar(stat = "identity", position = "stack") +
  coord_polar(theta = "y", start = 0, direction = -1) +
  geom_text(aes(label = count), size = 2, position = position_stack(vjust = 0.5)) +
  ggtitle("g3")

# Diagramme circulaire montrant les proportions
g4 <-
  diamonds %>%
  group_by(color) %>%
  summarize(count = n() / length(diamonds$color) * 100) %>%
  ggplot(aes(x = "", y = count, fill = color)) +
  geom_bar(stat = "identity", position = "stack") +
  coord_polar(theta = "y", start = 0, direction = -1) +
  geom_text(aes(label = paste0(round(count, digits = 2), " %")), size = 2, position = p
  ggtitle("g4")
```

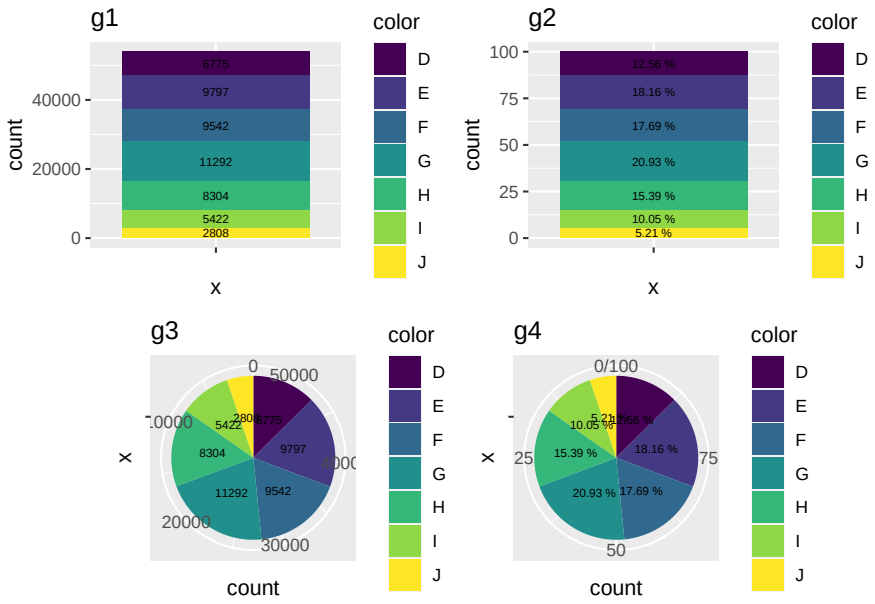


Figure 3.18: Différentes sortes de diagrammes pour représenter des proportions

3.2.2 Déterminer la tendance centrale

Dans le cadre de variables qualitatives, la tendance centrale peut être renseignée à l'aide du mode en présence d'une variable nominale, et à l'aide de la médiane ou du mode en présence d'une variable ordinale (Gonzales & Ottenbacher, 2001). Avec des variables qualitatives, il est possible de prendre connaissance du mode facilement à l'aide d'un tableau de résultats qui récapitulerait, par ordre décroissant, les effectifs et les proportions associées aux différentes modalités de la variable étudiée. Ce type de tableau peut être obtenu à l'aide de la fonction `freq()` du package `questionr`, qu'il convient d'installer et charger au préalable afin de pouvoir l'utiliser. Dans le tableau ci-dessous, le mode de la variable `color` dans le jeu de données `diamonds` est la modalité présente sur la première ligne du tableau.

```
library(questionr)
freq(diamonds$color, valid = TRUE, total = TRUE, sort = "dec")
```

```
##           n     %  val%
## G     11292  20.9  20.9
## E      9797  18.2  18.2
## F      9542  17.7  17.7
## H      8304  15.4  15.4
```

```
## D      6775  12.6  12.6
## I      5422  10.1  10.1
## J      2808   5.2   5.2
## Total 53940 100.0 100.0
```

3.3 Résumé

- En statistiques, la notion de population désigne l'ensemble des individus existant qui présentent un ou plusieurs critères d'intérêt. Un échantillon est alors une fraction de la population étudiée, composée d'individus qui en principe sont représentatifs de la population étudiée.

- Avec R, les graphiques peuvent être réalisés, en autres, à l'aide du package `ggplot2` et des fonctions associées.

- Lors de l'analyse d'une variable quantitative, une première étape doit être de visualiser graphiquement la distribution des observations. Cela peut se faire à l'aide d'un histogramme (`ggplot2::geom_histogram()`), d'une boîte à moustaches (`ggplot2::geom_boxplot()`), ou encore d'un nuage de points (`ggplot2::geom_point()`).

- Ces trois derniers types de graphiques peuvent être obtenus simultanément à l'aide de la fonction `g_distri()` présentée dans ce chapitre.

- La distribution d'une variable quantitative peut être notamment (mais pas seulement) de forme gaussienne, asymétrique, leptocurtique, ou encore platycurtique.

- Les indices de position disponibles pour résumer une variable quantitative sont la moyenne (`mean()`), la médiane (`median()`), le mode (`lsr::modeOf()`), et la moyenne rognée (`mean(trim = ...)`).

- Les indices de dispersion disponibles pour résumer une variable quantitative sont l'étendue (`min()`, `max()`, `range()`), l'écart-type (`sd()`), et les quartiles (`quantile(probs = c(0.25, 0.75))`).

- L'indice statistique permettant de décrire le niveau d'asymétrie d'une variable quantitative est le coefficient d'asymétrie (`e1071::skewness()`).

- L'indice statistique permettant de décrire le niveau d'aplatissement d'une variable quantitative est le coefficient d'aplatissement (`e1071::kurtosis()`).

- Les fonctions `psych::describe()` et `psych::describeBy()` permettent de récapituler les indices statistiques généralement étudiés dans le cadre de variables quantitatives.

- Lorsque la distribution d'une variable quantitative est gaussienne, approximativement 68.3 %, 95.5 %, et 99.7 % des observations sont situées dans $\pm$ 1 s, $\pm$ 2 s, et $\pm$ 3 s autour de la moyenne, respectivement (s étant l'écart-type de la variable).
- Lorsque la distribution d'une variable quantitative est asymétrique, la médiane peut être l'indicateur le plus adapté pour décrire la tendance centrale, en particulier en présence de petits échantillons.
- Lors de l'analyse d'une variable qualitative, une première étape doit être de visualiser graphiquement la distribution des effectifs. Cela peut se faire à l'aide d'un diagramme en barres (`ggplot2::geom_bar()`).
- Les variables qualitatives nominales devraient être visualisées avec une organisation des modalités selon un ordre croissant ou décroissant.
- Les variables qualitatives ordinales devraient être visualisées avec une organisation des modalités selon leur ordre naturel.
- Dans le cadre de variables qualitatives nominales, la tendance centrale peut être étudiée à l'aide du mode.
- Dans le cadre de variables qualitatives ordinales, la tendance centrale peut être étudiée à l'aide de la médiane ou du mode.
- La fonction `questionr::freq()` permet de récapituler les effectifs et les proportions relatifs aux modalités d'une variable qualitative.

Chapter 4

Analyses descriptives bivariées

Réaliser une analyse bivariée désigne le fait d'étudier la relation qui peut exister entre deux variables. Dans ce chapitre, nous allons voir les procédures graphiques et calculatoires qui permettent d'étudier et de quantifier le degré de relation pouvant exister entre deux variables dans les cas suivants : entre deux variables quantitatives, entre deux variables qualitatives, et entre une variable quantitative et une variable qualitative. Comme dans le chapitre précédent, l'objectif est ici d'explorer et de décrire les données et leurs relations à l'échelle d'un échantillon, sans pour autant chercher à déterminer l'incertitude qu'il peut exister dans les statistiques calculées en vue de les utiliser pour réaliser une inférence dans la population représentée.

4.1 Relation entre deux variables quantitatives

4.1.1 Etudier graphiquement la relation

Comme dans le cadre d'analyses univariées, une bonne pratique, lorsqu'on étudie une relation bivariée, est de faire un graphique. Avec des variables quantitatives, il s'agit de montrer les valeurs d'une variable en fonction des valeurs de l'autre variable, chose que permet un simple nuage de points. Plusieurs types de relations peuvent alors être rencontrés, ces relations pouvant potentiellement s'apparenter à autant de fonctions mathématiques que l'on connaît. Parmi les plus connues, on a par exemple les relations linéaires, les relations logarithmiques, ou encore les relations quadratiques, illustrées ci-dessous.

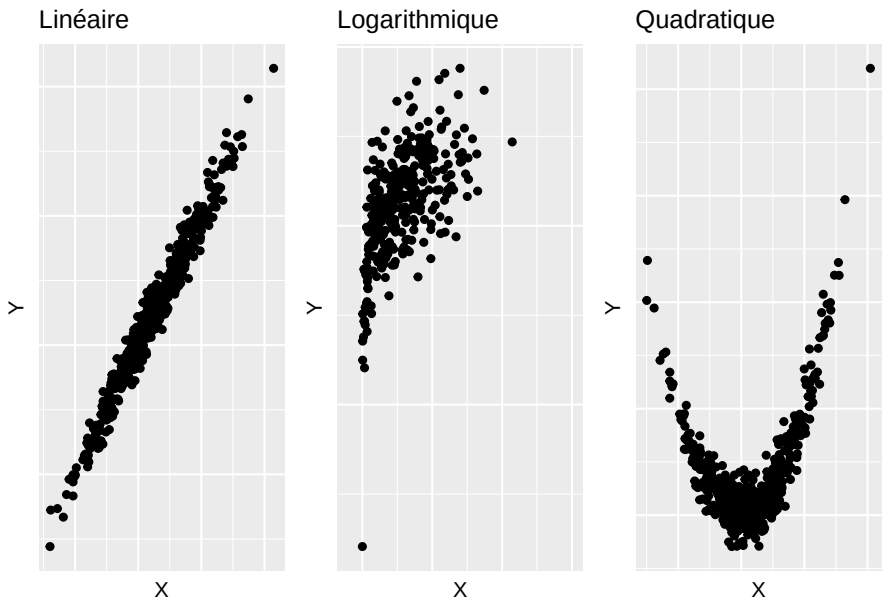


Figure 4.1: Différentes formes de relation

Dans R, pour obtenir un nuage de points à partir d'un jeu de données, il est possible d'utiliser la fonction `ggplot()` en l'associant à la fonction `geom_point()` du package `ggplot2`, comme dans l'exemple ci-dessous qui utilise le jeu de données `mtcars` (qui est intégré à R de base) et les variables `hp` (*gross horsepower*) et `mpg` (*miles/US gallon*). Dans cet exemple, on peut voir que la relation semble gobalement linéaire négative (voire curvilinéaire négative si l'on donne de l'importance au point isolé à droite du graphique).

```
ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point()
```

4.1.2 Etudier numériquement la relation

Le coefficient de corrélation de Pearson

Lorsque la relation étudiée semble linéaire, l'étude numérique classique consiste à calculer le coefficient de corrélation de Pearson, noté r, dont la valeur vise à renseigner dans quelle mesure le nuage de points représentant le lien entre les deux variables étudiées suit une droite. Avant de se lancer dans le calcul du coefficient de corrélation de Pearson pour étudier la relation entre une variable X et une variable Y, il peut donc être utile de compléter le nuage

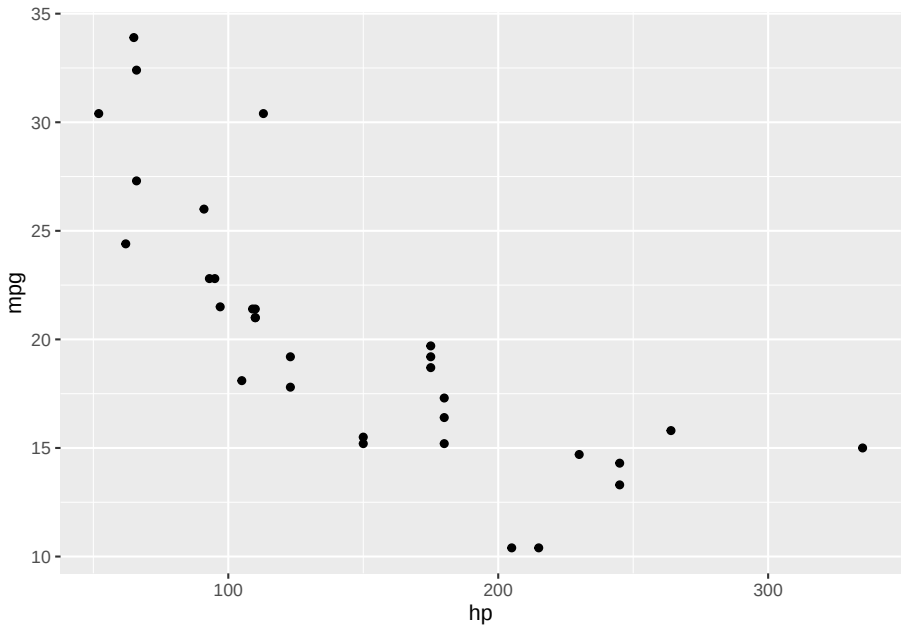


Figure 4.2: Exemple de nuage de points

de points montré ci-dessus avec une droite d'équation de type $Y = aX + b$. Cette équation serait la meilleure modélisation possible de la relation linéaire entre X et Y, de telle sorte que parmi l'infinité d'équations qui pourraient lier X à Y, c'est cette équation qui au total donnerait la plus petite erreur lorsque l'on voudrait prédire Y à partir de X. Si X et Y sont liées de manière linéaire, alors le nuage des points relatifs aux deux variables devrait s'étaler le long de cette droite. Pour obtenir cette droite en plus du nuage de points, il est possible d'utiliser la fonction `geom_smooth()` du package `ggplot2`.

```
ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  geom_smooth(formula = y ~ x, method = "lm", se = FALSE)
```

Dans la fonction `geom_smooth()` qui a été utilisée dans l'exemple ci-dessus, on note que l'argument `formula` pourrait être considéré comme facultatif car il s'agit ici de la configuration par défaut de la fonction. En revanche, l'argument `method` doit être ici configuré avec `"lm"` (pour *linear model*) car ce n'est pas la méthode graphique configurée par défaut dans la fonction. Enfin, l'argument `se` permet de montrer ou non un intervalle de confiance autour de la droite de régression, ce qui n'a pas été activé ici (par défaut, l'argument `se` est configuré pour montrer cet intervalle de confiance). Dans l'exemple montré ci-dessus, la représentation graphique encourage fortement à penser que l'un des types de

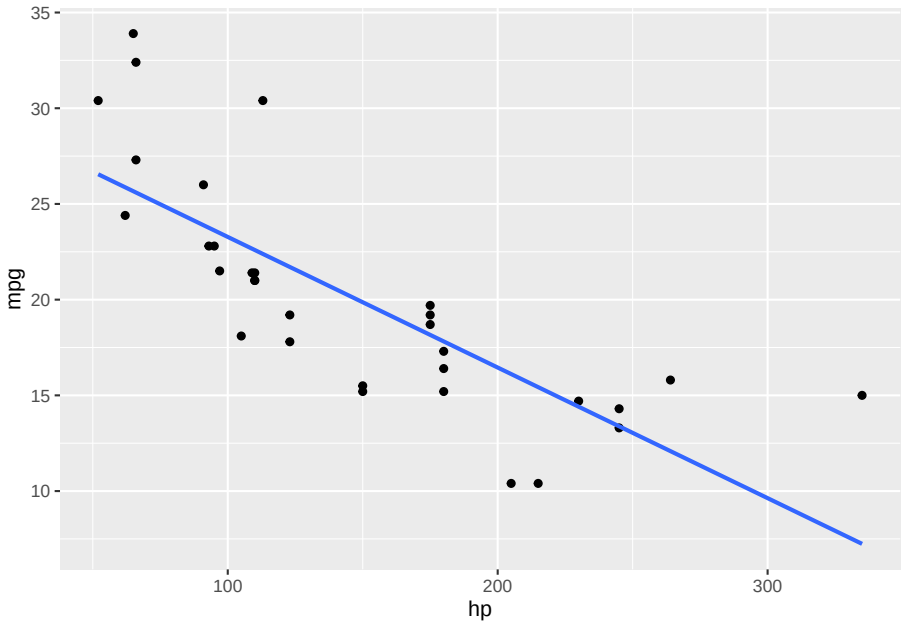


Figure 4.3: Nuage de points avec droite de régression

relations à envisager prioritairement dans l'étude des deux variables est la relation linéaire. Cette information rend pertinente l'utilisation du coefficient de corrélation de Pearson pour une étude numérique de la relation en question.

La valeur du coefficient de corrélation de Pearson peut aller de 1 (suggérant une relation linéaire positive parfaite) à -1 (suggérant une relation linéaire négative parfaite). Des valeurs proches de 0 suggèreraient une abscence de relation linéaire. La formule du coefficient de corrélation de Pearson (r) pour un échantillon est notée ci-dessous :

$$r_{X,Y} = \frac{COV_{X,Y}}{s_X s_Y} = \frac{\sum_{i=1}^{N}(Xi - \overline{X})(Yi - \overline{Y})}{N-1}\frac{1}{s_X s_Y},$$

COV désignant la covariance entre les variables X et Y, Xi et Yi les valeurs de X et Y pour une observation i, $\overline{X}$ et $\overline{Y}$ les moyennes des variables X et Y, N le nombre d'observations, et s_X et s_Y les écarts-types respectifs des variables X et Y. Cette formule indique que le coefficient de corrélation de Pearson s'obtient en divisant la covariance des deux variables étudiées par le produit de leurs écarts-types respectifs.

Le tableau ci-dessous montre les premières étapes du calcul de la covariance pour des couples de variables fictifs $(X1, Y1)$, $(X1, Y2)$, et $(X1, Y3)$. En

Table 4.1: Calcul de la covariance

X1	Y1	Y2	Y3	X1Y1	X1Y2	X1Y3
0	0	0	0	36	40.285714	-36
2	2	1	-2	16	22.857143	-16
4	4	15	-4	4	-16.571429	-4
6	6	5	-6	0	0.000000	0
8	8	11	-8	4	8.571429	-4
10	10	3	-10	16	-14.857143	-16
12	12	12	-12	36	31.714286	-36

particulier, la partie droite du tableau (de X1Y1 à X1Y3) montre le calcul du produit $(Xi - \overline{X})(Yi - \overline{Y})$ pour les différents couples de variables et cela pour chaque ligne du jeu de données.

Ce que ce tableau peut permettre de rendre compte, c'est que plus les deux variables étudiées évolueront de manière consistante dans des sens identiques comme avec $X1$ et $Y1$, ou de manière consistante dans des sens opposés comme avec $X1$ et $Y3$, plus les produits $(Xi - \overline{X})(Yi - \overline{Y})$ donneront respectivement des grands scores positifs ou des grands scores négatifs, et moins les scores $(Xi - \overline{X})(Yi - \overline{Y})$ à additionner pour le calcul de la covariance s'annuleront. En effet, avec une relation relativement linéaire et positive les scores seront plus systématiquement positifs, et avec une relation relativement linéaire et négative les scores seront plus systématiquement négatifs. Toutefois, lorsqu'on aura des variables qui n'évolueront pas de manière consistante dans le même sens ou dans un sens opposé comme avec $X1$ et $Y2$, les scores positifs et négatifs liés aux calculs $(Xi - \overline{X})(Yi - \overline{Y})$ auront tendance à s'annuler et donneront lieu à une somme des scores $(Xi - \overline{X})(Yi - \overline{Y})$ diminuée, et donc à une covariance et au final à un coefficient de corrélation de Pearson tirés vers 0. Ces différents cas de figure et les calculs $(Xi - \overline{X})(Yi - \overline{Y})$ correspondants sont illustrés sur la figure ci-dessous. Sur cette figure, chaque carré correspond au calcul $(Xi - \overline{X})(Yi - \overline{Y})$, le carré étant bleu lorsque le résultat du calcul est positif, et rouge lorsque le résultat est négatif. L'aire d'un carré illustre la grandeur du score issu du calcul. Sur les figures de gauche et de droite, on distingue une relation linéaire parfaite, ce qui maximise les scores à additionner pour le calcul de la covariance, dans le positif pour la figure de gauche et dans le négatif pour la figure de droite. Au milieu, on remarque que le manque de relation linéaire donne lieu à des carrés à la fois bleus et rouges, indiquant que les scores associés aux calculs $(Xi - \overline{X})(Yi - \overline{Y})$ de la covariance s'annulent et diminuent ainsi la valeur finale de la covariance.

Dans R, le coefficient de corrélation de Pearson peut être obtenu avec la fonction `cor()`. Dans l'exemple ci-dessous qui reprend les variables du jeu de données `mtcars` utilisées plus haut, on observe un coefficient négatif, relativement proche de -1, suggérant une relation relativement linéaire et négative entre les variables étudiées.

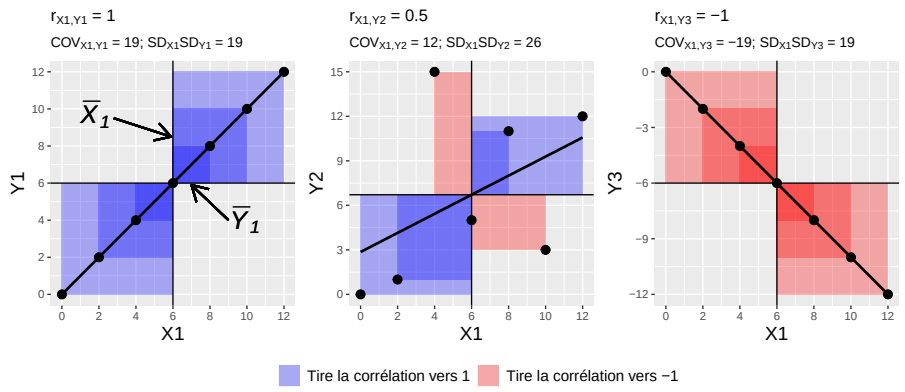


Figure 4.4: Détermination de la covariance

```
cor(x = mtcars$hp, y = mtcars$mpg, method = "pearson")
```

```
## [1] -0.7761684
```

Toutefois, la fonction `cor.test()` sera plus intéressante pour la suite car elle permet de calculer des indices statistiques de probabilité qui seront nécessaires dès lors qu'il s'agira de chercher à inférer la valeur d'une corrélation dans une population d'où l'échantillon étudié provient. La valeur de la corrélation est donnée à la fin de la liste des informations qui apparaissent suite à l'activation de la fonction.

```
cor.test(x = mtcars$hp, y = mtcars$mpg, method = "pearson")
```

```
##
##  Pearson's product-moment correlation
##
## data:  mtcars$hp and mtcars$mpg
## t = -6.7424, df = 30, p-value = 1.788e-07
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.8852686 -0.5860994
## sample estimates:
##        cor
## -0.7761684
```

Sur la base de travaux antérieurs, Hopkins et al. (2009) ont fait une proposition de classification pour qualifier la valeur du coefficient de

Table 4.2: Taille d'effet pour une corrélation

Petite	Moyenne	Grande	Très grande	Extrêmement grande
0.1	0.3	0.5	0.7	0.9

corrélation qui serait obtenue dans le cadre d'une relation linéaire. Cette proposition est montrée ci-dessous :

Pour visualiser le lien que l'on peut faire entre la forme du nuage de points et la valeur du coefficient de corrélation de Pearson que l'on peut obtenir, la page web proposée par Kristoffer Magnusson (https://rpsychologist.com/correlation) peut être particulièrement intéressante. Cette page web donne la possibilité de faire varier manuellement la valeur du coefficient de corrélation de Pearson pour ensuite voir un nuage de points type correspondant à cette valeur. Faites un essai !

A noter que la valeur du coefficient de corrélation de Pearson est très dépendante de la manière dont sont distribuées les variables, avec une influence particulière de la variabilité des données (Halperin, 1986). L'influence de la variabilité est illustrée sur la figure ci-dessous. A gauche, on observe un nuage de points représentant une population complète, avec en conséquence une variablité le long des axes X et X relativement importante. La valeur du coefficient de corrélation de Pearson est ici particulièrement élevée. A droite, on observe exactement les mêmes valeurs, mais sur un intervalle dont l'étendue a été manuellement restreinte, diminuant ainsi la variabilité. On observe alors une diminution de la valeur du coefficient de corrélation de Pearson, alors qu'il s'agit à l'origine du même jeu de données que celui de gauche. Cet exemple doit faire prendre conscience qu'il faut faire attention lorsqu'on cherche à comparer des coefficient de corrélation de Pearson obtenus avec différents échantillons présentant des caractéristiques différentes, car si ces échantillons n'ont pas les mêmes niveaux de variabilité, les valeurs des coefficients de corrélation ne seront pas vraiment comparables, en sachant que c'est l'échantillon qui présente la plus grande variabilité qui aura mathématiquement plus de chances de présenter une valeur de coefficient de corrélation plus élevée.

Un exemple extrême de l'influence de la variabilité des données sur la valeur du coefficient de corrélation de Pearson est montré sur la figure ci-dessous. Les deux graphiques montrent exactement les mêmes données, à ceci près que sur le graphique de droite, on a remplacé en ordonnées une valeur du graphique de gauche pour lui donner la valeur de 10. L'influence de cette action sur la valeur du coefficient de corrélation est particulièrement nette, alors qu'une seule valeur a été modifée. Ceci montre qu'il faut faire attention aux valeurs extrêmes qui pourraient grandement influencer la valeur de corrélation obtenue, notamment en présence d'échantillons de taille relativement faible. Dans le cas où la valeur du coefficient de corrélation de Pearson serait très influencée par une valeur, il pourrait être une bonne pratique de calculer la

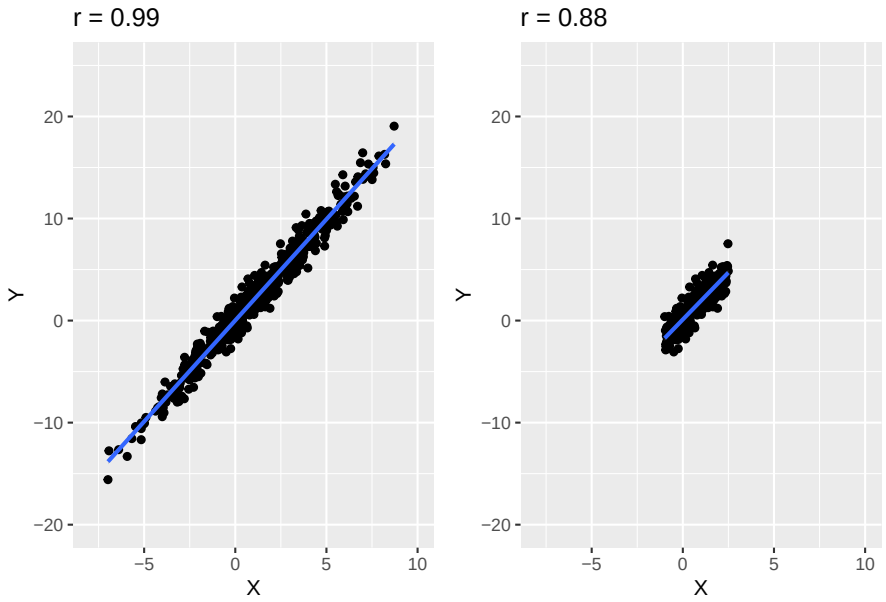


Figure 4.5: Coefficient de corrélation de Pearson et variabilité des données

valeur du coefficient de corrélation avec et sans cette valeur afin de pouvoir quantifier son influence sur la relation étudiée (Halperin, 1986). Une alternative pourrait être aussi d'étudier la relation à l'aide d'autres types de coefficients que celui de Pearson, tels que celui de Spearman, présenté plus bas. Toutes ces informations doivent en tous les cas faire prendre conscience qu'il n'est pas toujours pertinent de calculer le coefficient de corrélation de Pearson. En ce sens, lorsqu'on cherche à inférer la valeur du coefficient de corrélation de Pearson dans la population étudiée, et cela avec un degré d'incertitude bien défini, il convient de vérifier certains prérequis, lesquels seront abordés plus tard dans ce livre.

Lorsque la relation étudiée ne semble pas linéaire mais s'apparente assez clairement à d'autres fonctions mathématiques, telles que des relations logarithmiques ou polynomiales, il est possible de transformer une des variables, voire les deux, pour rendre la relation linéaire et à nouveau étudiable à l'aide du coefficient de corrélation de Pearson (Halperin, 1986). Toutefois, il est aussi possible de créer des modèles de régression non linéaires afin de regarder si ces modèles correspondent bien aux données. La détermination et la validation d'un modèle non linéaire qui correspondrait bien aux données confirmerait alors que la relation étudiée a une forme particulière et potentiellement prédictible. Les procédures pour explorer différents modèles de régression (linéaires et non linéaires) sont abordées au chapitre suivant. Enfin, une dernière alternative possible, pour étudier la relation entre deux

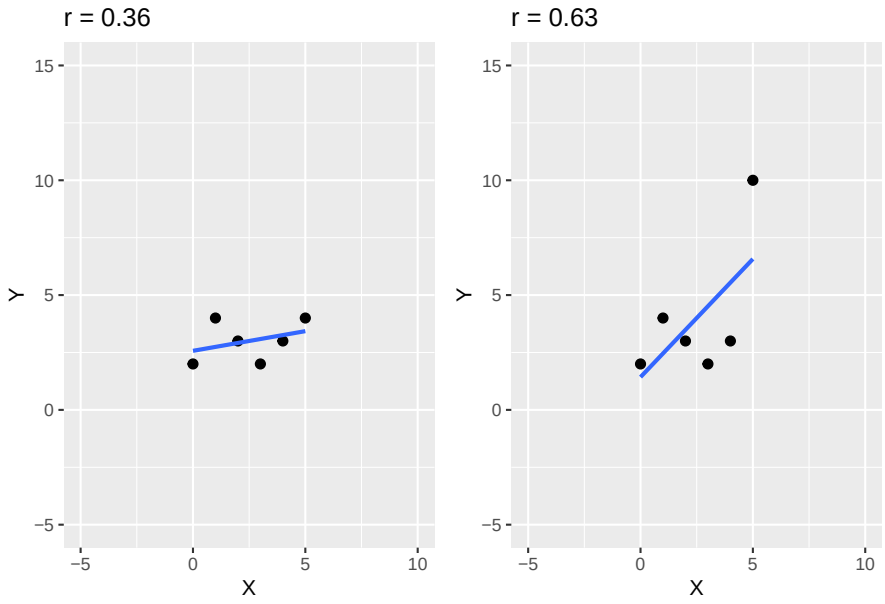


Figure 4.6: Coefficient de corrélation de Pearson et outliers

variables quantitatives dont les distributions ne permettraient pas d'utiliser correctement le coefficient de corrélation de Pearson, serait l'utilisation de coefficients de corrélation basés sur les rangs, tels que le coefficient de corrélation de Spearman.

Le coefficient de corrélation de Spearman

Lorsque le coefficient de corrélation de Pearson ne permet pas de caractériser fiablement le degré de relation linéaire **entre les valeurs** de deux variables (par exemple en présence de valeurs aberrantes au sein d'un échantillon de petite taille), une alternative peut être d'étudier le degré de relation linéaire **entre les rangs** de ces deux variables. Le rang, c'est le classement (ou la position) d'une observation donnée en fonction de sa valeur. Dans une variable, les observations avec les valeurs les plus faibles seront associées aux rangs les plus bas alors que les observations avec les valeurs les plus élevées seront associées aux rangs les plus élevés. Une illustration de la notion de rang est proposée ci-dessous pour la variable `hp` du jeu de données `mtcars`. Dans le tableau ci-dessous, les lignes ont été ordonnées sur la base des rangs de la variable `hp`. On pourra remarquer que dans le tableau, nous avons ce qu'on appelle des ex-aequos, c'est-à-dire que plusieurs observations peuvent présenter les mêmes valeurs, et donc avoir le même rang.

Le fait d'étudier l'existence d'une relation linéaire entre les rangs et non plus entre les valeurs de deux variables permet de s'affranchir de l'influence possible

de valeurs très extrêmes, dans l'une et/ou l'autre variable, sur le calcul final de la corrélation. Pour déterminer alors la valeur de la corrélation, une manière de procéder est d'appliquer la méthode de calcul du coefficient de corrélation de Pearson en utilisant non plus les valeurs des variables, mais les rangs correspondants. Cette methode, c'est celle du calcul du coefficient de corrélation de Spearman (*rho*). Si l'on suit *stricto sensu* cette définition, nous pourrions alors utiliser le code suivant pour avoir le coefficient de corrélation de Spearman :

```
cor(x = rank(mtcars$hp), y = rank(mtcars$mpg), method = "pearson")
```

```
## [1] -0.8946646
```

Toutefois, il existe une manière plus directe d'écrire les choses avec la fonction `cor`, qui contient un argument spécifiquement dédié au calcul du *rho* de Spearman :

```
cor(x = mtcars$hp, y = mtcars$mpg, method = "spearman")
```

```
## [1] -0.8946646
```

La fonction `cor.test` permet aussi de calculer le coefficient de corrélation de Spearman en fournissant d'autres informations potentiellement intéressantes pour donner une idée de la significativité statistique de l'estimation de *rho* pour la population étudiée.

```
cor.test(x = mtcars$hp, y = mtcars$mpg, method = "spearman")
```

```
##
##  Spearman's rank correlation rho
##
## data:  mtcars$hp and mtcars$mpg
## S = 10337, p-value = 5.086e-12
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## -0.8946646
```

Si l'on veut produire une représentation graphique qui illustre la valeur de *rho* obtenue, il pourrait être davantage pertinent de non plus montrer un nuage de points à partir des valeurs des variables mises en lien, mais un nuage de points à partir de leurs rangs respectifs.

```r
mtcars %>%
  mutate(hp_rank = rank(hp), mpg_rank = rank(mpg)) %>%
  ggplot(aes(x = hp_rank, y = mpg_rank)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```

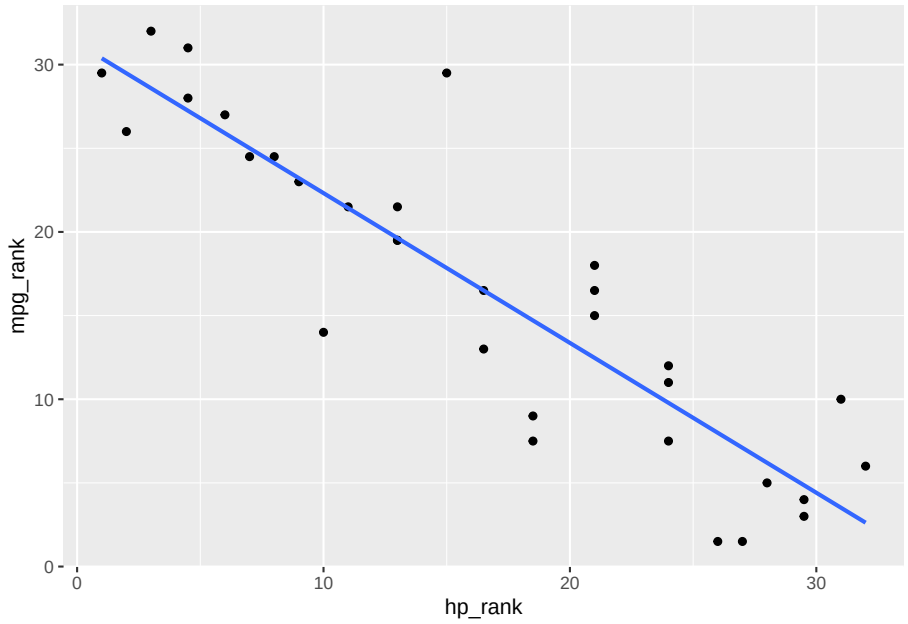


Figure 4.7: Graphique de corrélation pour le coefficient de Spearman

En matière d'interprétation, des valeurs de *rho* positives indiqueront que les deux variables mises en lien tendent à augmenter simultanément, on parlera alors de **relation monotone positive**. Dans le cas inverse, des valeurs négatives indiqueront que les deux variables mises en lien tendent à diminuer simultanément, on parlera alors de relation **monotone négative**. A noter cependant que de par son calcul, la valeur de *rho* ne permet pas de renseigner sur la forme de relation qu'il pourrait y avoir entre les valeurs des deux variables (e.g., linéaire ou curvilinéaire par exemple). Ceci est illustré sur la figure ci-dessous. La figure de gauche montre la relation entre les valeurs des variables X et Y, qui est caractérisée par un coefficient de corrélation de Spearman (*rho*) de 1, indiquant donc que la relation est parfaitement monotone positive, sans préjuger de la forme particulière que pourrait présenter la relation. Pour mieux comprendre pourquoi cette valeur de *rho* est de 1, la figure de droite montre la relation entre les rangs de ces deux variables X et Y. On voit en effet que la relation entre les rangs est effectivement parfaitement linéaire.

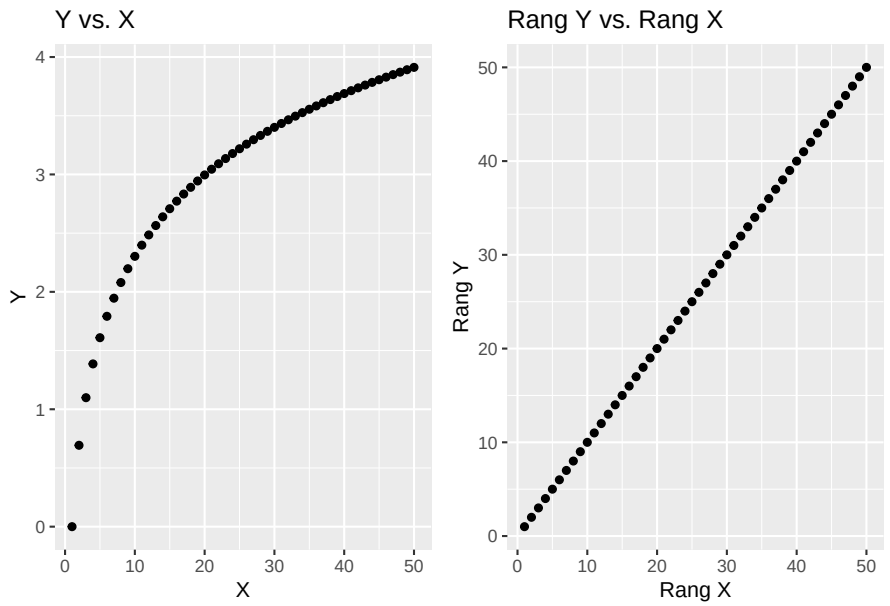


Figure 4.8: Explication du caractère monotone d'une variable évalué par le coefficient de Spearman

4.2 Relation entre deux variables qualitatives

4.2.1 Etudier graphiquement la relation

Plusieurs types de graphiques peuvent être envisagés lorsqu'il s'agit de montrer les données relatives au croisement de deux variables qualitatives. Une première approche consiste à utiliser des graphiques avec barres mises côte-à-côte et groupées sur l'axe horizontal en fonction des modalités d'une première variable, et colorées en fonction de la seconde variable. Cela est illustré sur la figure ci-dessous, qui a été réalisée à partir du jeu de données `JointSports`, lequel est utilisable après installation et chargement du package `vcd`. `JointSports` contient des données résumées avec des effectifs mis en lien avec les modalités de différentes variables qualitatives, comme on a pu l'obtenir avec les fonctions `group_by` et `summarize` dans les derniers exemples du chapitre précédent. (La différence qu'il y a ici avec ces précédents exemples est qu'ici, l'effectif est désigné par la variable `Freq`, alors qu'il s'agissait de la variable `count` auparavant.) Pour information, `JointSports` contient les données d'une enquête s'étant intéressée, en 1983 et 1985, aux opinions d'étudiants danois de 16 à 19 ans quant à la pratique sportive mixte. On note également que le code montré ci-dessous, qui a servi à générer les graphiques à suivre, utilise la fonction `theme_hc()` du package `ggthemes` afin de pouvoir

utiliser des couleurs de remplissage automatique particulières. Le package `ggthemes` doit être installé puis chargé pour pouvoir être utilisé.

```
library(vcd)
library(ggthemes)
```

```
# Reconfiguration de l'ordre des modalités de la variable opinion, et calcul des effectifs totaux
JointSports_new <-
  JointSports %>%
  mutate(opinion = fct_relevel(opinion, "very bad", "bad", "indifferent", "good", "very good"),
         gender = fct_relevel(gender, "Girl", "Boy")) %>%
  group_by(gender, opinion) %>%
  summarize(Freq = sum(Freq))

# Création des graphiques
g1 <-
  ggplot(data = JointSports_new, aes(x = gender, y = Freq, fill = opinion)) +
  geom_bar(stat = "identity",  position = "dodge") +
  scale_fill_brewer(palette = "Greens") +
  theme_hc() +
  theme(legend.position = "right") +
  ggtitle("g1 : Mise en avant de la comparaison des opinions")

g2 <-
  ggplot(data = JointSports_new, aes(x = opinion, y = Freq, fill = gender)) +
  geom_bar(stat = "identity",  position = "dodge") +
  theme_hc() +
  theme(legend.position = "right") +
  ggtitle("g2 : Mise en avant de la comparaison des genres")
```

Pour pouvoir réaliser le graphique g1 montré ci-dessus, il a fallu indiquer dans la fonction `ggplot()`, grâce à `x =`, la variable dont on voulait voir les modalités en abscisses, et il a fallu renseigner pour les ordonnées, à l'aide de `y =`, la variable contenant les effectifs correspondants, le tout toujours à l'intérieur de l'argument `aes()`. Etant donné que les données à montrer le long de l'axe des ordonnées sont explicitement indiquées avec `Freq`, il convient d'indiquer à l'intérieur de la fonction `geom_bar()` l'argument `stat = "identity"`, ce qui contraint à déterminer la hauteur des barres en fonction des valeurs de la variable `Freq`. A l'intérieur de la fonction `ggplot()` et de l'argument `aes()`, c'est l'argument `fill = opinion` qui a permis d'indiquer qu'on voulait des couleurs de remplissage différentes selon les modalités de la variable `opinion`. Enfin, c'est grâce à l'argument `position = "dodge"`, à l'intérieur de la fonction `geom_bar()`, que l'on a pu obtenir des barres mises côte-à-côte, et non pas de manière empilée. Une logique similaire a été utilisée pour le graphique g2 en modifiant le code de telle sorte que la distinction de

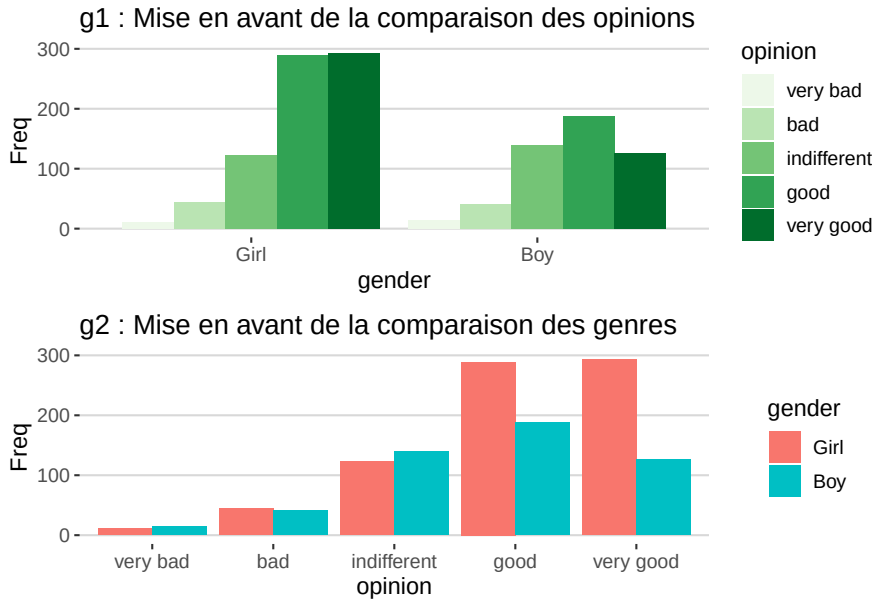


Figure 4.9: Exemples de diagramme en barres mises côte-à-côte

l'information avec des couleurs différentes se fasse avec la variable `gender`, et non plus `opinion`.

Les graphiques g1 et g2 montrent l'importance de la configuration du graphique en fonction des comparaisons que l'on veut principalement faire, et donc du message que l'on veut prioritairement délivrer. Un principe qui peut guider la conception du graphique est le fait qu'il est plus facile de comparer des barres qui sont mises juste côte-à-côte. Sur la base de ce principe, le graphique g1 ci-dessus permet de comparer plus facilement les diverses opinions relevées pour les garçons d'un côté et pour les filles de l'autre, alors que le graphique g2 permet de comparer plus facilement les réponses provenant des deux genres et cela pour chaque type d'opinion. Comme indiqué par Wilke (2018), les types graphiques illustrés avec les graphiques g1 et g2 ci-dessus peuvent parfois se voir attribuer le reproche que s'il est relativement facile de lire les informations encodées par des positions (cf. ligne de base sur les graphiques), il peut être être difficile de lire les informations encodées par une couleur dont la signification est indiquée en légende, car cela demande un effort mental supplémentaire de garder en tête la signification de la légende lorsqu'on lit le graphique. Pour palier ce problème, qui, selon Wilke (2018), est au final une affaire de goût, on pourrait utiliser la fonction `facet_wrap()` pour créer une figure telle qu'illustré ci-dessous. (La figure ci-dessous reprend la logique du graphique g1 montré plus haut, avec un besoin de légende pour la variable `opinion` qui n'existe plus car la fonction `facet_wrap()` a permis de

montrer les diagrammes en barres pour les deux genres de manière séparée, dans des encarts différents, et avec chacun leur propre axe des abscisses relatif aux modalités de la variable `opinion`.)

```
ggplot(data = JointSports_new, aes(x = opinion, y = Freq)) +
  geom_bar(stat = "identity") +
  facet_wrap(. ~ gender)
```

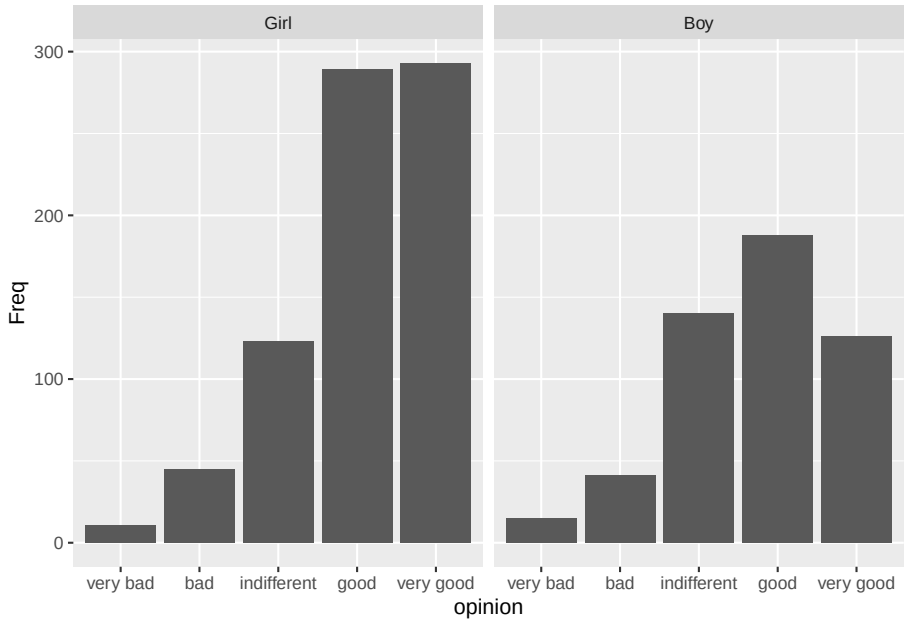


Figure 4.10: Diagrammes en barres côte-à-côte séparés selon une variable catégorielle

Dans certains cas, on peut vouloir comparer les effectifs relatifs aux modalités d'une première variable qualitative avec des barres mises côte-à-côte, et n'utiliser la seconde variable qualitative que pour avoir un peu d'éléments de contexte "à l'intérieur" des effectifs affichés pour la première variable qualitative. La figure ci-dessous illustre ce cas de figure où la hauteur des barres sert prioritairement à comparer les effectifs relatifs à diverses opinions, et la coloration des barres sert à fournir une idée de la répartition hommes / femmes dans les réponses, sans pourtant avoir l'ambition de comparer cette répartition hommes / femmes facilement d'un type d'opinion à un autre.

```
JointSports_new %>%
  ggplot(aes(x = opinion, y = Freq, fill = gender)) +
  geom_bar(stat = "identity") +
```

```
  theme_hc() +
  geom_text(aes(label = Freq), size = 3, position = position_stack(vjust = 0.5))
```

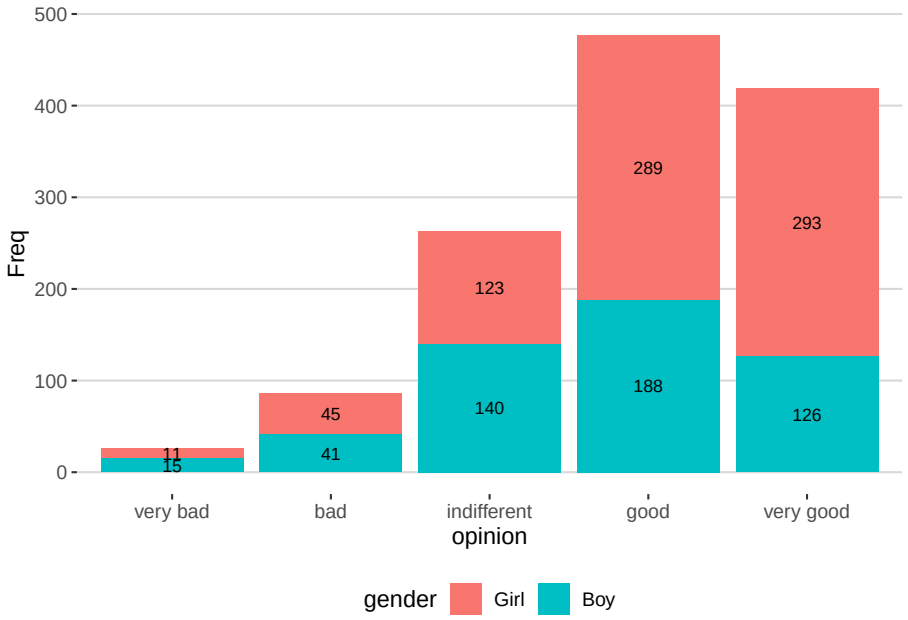


Figure 4.11: Exemple de diagramme en barres empilées

Les graphiques présentés dans cette sous-partie montrent des valeurs d'effectifs, mais selon l'objectif, il pourrait être aussi envisagé d'utiliser ces graphiques pour montrer des proportions. Cela dit, il existe d'autres visualisations possibles des proportions pour visualiser le lien entre deux variables qualitatives. Ces visualisations peuvent être consultées dans l'ouvrage en ligne de Wilke (2018).

4.2.2 Etudier numériquement la relation

Effectifs et proportions

Lorsqu'il s'agit de mener une étude numérique de la relation entre deux variables qualitatives, une première démarche à mettre en oeuvre est de récapituler numériquement les effectifs qui correspondent au croisement des deux variables. Pour cela, la fonction `table()` intégrée à R de base s'avère très pratique. Cependant, cette fonction requiert d'utiliser le jeu de données initial complet (i.e., avec toutes les observations), ce qui n'est pas le cas du jeu de données `JointSports` que nous avons utilisé juste précédemment, car ce

dernier contient des effectifs déjà récapitulés par modalité de variables. Pour pouvoir illustrer le fonctionnement de la fonction `table()` avec les informations du jeu de données `JointSports`, j'ai donc crée un jeu de données complet qui, une fois résumé comme c'est le cas plus haut avec `JointSports`, donnerait les mêmes résultats. Ce nouveau jeu de données se nomme `JointSports_full`.

```
id <- rep(1 : sum(JointSports$Freq))
year <- c(rep("1983", 656), rep("1985", 615))
grade <- c(rep("1st", 350), rep("3rd", 306), rep("1st", 354), rep("3rd", 261))
gender <- c(rep("Boy", 134), rep("Girl", 216), rep("Boy", 115), rep("Girl", 191), rep("Boy", 157)
opinion <- c(
  rep("very good", 31), rep("good", 51), rep("indifferent", 38), rep("bad", 10), rep("very bad",
  rep("very good", 103), rep("good", 67), rep("indifferent", 29), rep("bad", 15), rep("very bad",
  rep("very good", 23), rep("good", 39), rep("indifferent", 36), rep("bad", 15), rep("very bad",
  rep("very good", 61), rep("good", 72), rep("indifferent", 39), rep("bad", 16), rep("very bad",
  rep("very good", 41), rep("good", 67), rep("indifferent", 35), rep("bad", 12), rep("very bad",
  rep("very good", 77), rep("good", 80), rep("indifferent", 27), rep("bad", 10), rep("very bad",
  rep("very good", 31), rep("good", 31), rep("indifferent", 31), rep("bad", 4), rep("very bad", 7
  rep("very good", 52), rep("good", 70), rep("indifferent", 28), rep("bad", 4), rep("very bad", 3
  )

JointSports_full <-
  data.frame(id = id, year = year, grade = grade, gender = gender, opinion = opinion) %>%
  mutate(opinion = fct_relevel(opinion, "very bad", "bad", "indifferent", "good", "very good"))
```

Une fois que l'on a un jeu de données complet sous la main, il est possible de créer ce qu'on appelle un **tableau de contingence**, c'est-à-dire ici un tableau qui récapitule numériquement les effectifs à la croisée des deux variables qui nous intéressent. Pour faire cela, on peut utiliser la fonction `table()` en suivant différentes méthodes montrées ci-dessous. (Le code montré ci-dessous aboutit aux mêmes informations que celles montrées sur le denier graphique ci-dessus.)

```
# 1ère méthode
tab <-
  with(JointSports_full,
       table(opinion, gender))

# 2ème méthode
tab <- table(JointSports_full$opinion, JointSports_full$gender)

# Visualisation du tableau de contingence
tab
```

```
##
```

```
##                Boy Girl
##   very bad      15   11
##   bad           41   45
##   indifferent  140  123
##   good         188  289
##   very good    126  293
```

Un tableau de contingence permet donc de comparer des effectifs en fonction de plusieurs modalités et variables à la fois. Le problème, lorsqu'on utilise des effectifs, est que certaines comparaisons peuvent être difficiles à faire lorsque les effectifs totaux liés aux différentes modalités ne sont pas comparables. Par exemple, dans le tableau montré ci-dessus, l'effectif total des filles est de 761 alors que celui des garçons est de 510, ce qui rend difficile la comparaison des garçons et des filles pour les différents types d'opinion recensés dans l'enquête danoise présentée plus haut. C'est pour cela qu'il convient, dans certains cas, de calculer les proportions correspondant à ces différents effectifs. Pour ce faire, on peut :

- Utiliser la fonction `prop.table()`, qui va convertir en proportions les effectifs montrés plus haut en considérant l'effectif total de tout le tableau :

```
round(prop.table(tab) * 100, digits = 2)
```

```
##
##                 Boy  Girl
##   very bad     1.18  0.87
##   bad          3.23  3.54
##   indifferent 11.01  9.68
##   good        14.79 22.74
##   very good    9.91 23.05
```

- Utiliser la fonction `lprop()` du package `questionr`, qui va convertir en proportions les effectifs montrés plus haut en considérant l'effectif total de chaque ligne du tableau :

```
##
##                Boy   Girl  Total
##   very bad     57.7  42.3 100.0
##   bad          47.7  52.3 100.0
##   indifferent  53.2  46.8 100.0
##   good         39.4  60.6 100.0
##   very good    30.1  69.9 100.0
##   Ensemble     40.1  59.9 100.0
```

- Utiliser la fonction `cprop()` du package `questionr`, qui va convertir en proportions les effectifs montrés plus haut en considérant l'effectif total de chaque colonne du tableau :

```
cprop(tab)
```

```
##
##                Boy   Girl  Ensemble
##   very bad      2.9   1.4   2.0
##   bad           8.0   5.9   6.8
##   indifferent  27.5  16.2  20.7
##   good         36.9  38.0  37.5
##   very good    24.7  38.5  33.0
##   Total       100.0 100.0 100.0
```

Il convient bien de noter que les proportions données par ces différentes fonctions doivent être utilisées selon les comparaisons que l'on veut faire. L'analyse descriptive consiste alors à voir si, tant d'un point de vue graphique que numérique, on observe des différences de scores particulières entre les modalités d'une variable qualitative en fonction des modalités de l'autre variable qualitative. Si l'on considère le dernier tableau, on peut par exemple observer une très légère tendance à ce que les garçons soient davantage polarisés, par rapport aux filles, sur des opinions négatives vis-à-vis des pratiques sportives mixtes, alors que les filles seraient légèrement plus polarisées que les garçons sur des opinions positives, ce qui n'empêche pas que, pour les deux genres, il y a une polarisation principale sur des opinions neutres à positives.

Si l'on a évoqué l'idée que les proportions permettent de mieux comparer les choses, il convient de faire attention malgré tout à la manière dont l'effectif total associé à une modalité d'une variable se répartit selon les modalités de la seconde variable, car une répartition non homogène de l'effectif lié à une modalité d'une variable dans les différentes modalités de la seconde variable peut donner lieu à des conclusions tout à fait différentes selon que l'on considère une analyse globale ou une analyse par modalité. Un exemple connu pour illustrer le type de situations dans lesquelles il faut être vigilant est le cas du taux de réussite des femmes à l'université de Berkeley en 1973 qui était bien inférieur à celui des hommes lorsqu'on considérait les taux de réussite à l'échelle de l'ensemble de l'unversité (Bickel et al., 1975). Toutefois, une analyse par matière permettait de voir que les taux de réussite des femmes étaient très similaires à ceux des hommes dans la plupart des matières, voire étaient largement supérieurs à celui des hommes dans certaines matières. Cette situation, qui paraît de prime abord paradoxale, illustre pleinement ce qu'on appelle un paradoxe de Simpson (i.e., le fait que le résultat observé lors d'une analyse globale d'un groupe puisse se retrouver annulée voire inversée

lors d'une analyse à l'échelle de sous-groupes). Dans le cas présent, le paradoxe s'explique par le fait que la majorité des femmes avaient candidaté dans des matières qui étaient très sélectives, c'est-à-dire où le taux de réussite était faible (il l'était aussi pour les hommes). Très peu de femmes avaient candidaté là où les taux de réussite étaient très élevés (pour les femmes comme pour les hommes). Donc au total, les hommes se retrouvaient avec un pourcentage de réussite global bien meilleur que celui des femmes, seulement parce que en proportions, plus d'hommes s'étaient engagés dans les matières où les taux de réussite étaient bien meilleurs.

4.3 Relation entre une variable quantitative et une variable qualitative

Lorsqu'on analyse une variable quantitative en fonction d'une variable qualitative, on peut notamment distinguer le fait que les données quantitatives soient pairées (*within-subject design* en anglais) ou non (*between-subject design* en anglais). Avoir des données non pairées signifie que pour chaque modalité de la variable qualitative étudiée, les données quantitatives correspondantes ne sont pas liées. Un exemple simple peut être l'analyse de la taille des individus en fonction du sexe. Dans ce cas, les données quantitatives de taille pour le sexe masculin et pour le sexe féminin proviendront forcément d'individus différents et ne formeront donc pas des paires. En revanche, on peut aussi avoir des études dans lesquelles plusieurs individus sont évalués dans plusieurs conditions différentes que l'on chercherait à comparer. En sciences du sport, un exemple relativement classique est de tester la performance d'endurance (variable quantitative) en ayant pris (condition de test) ou non (condition contrôle) une substance potentiellement ergonénique (la prise de substance ou non étant les modalités d'une même variable qualitative de type condition). Dans ce cas là, tous les individus auront des données dans les deux conditions et ces données seront donc pairées (dépendantes).

4.3.1 Etudier graphiquement la relation

Lorsque l'on cherche à explorer la relation qu'il peut y avoir entre une variable quantitative et une variable qualitative, il peut être intéressant de visualiser la distribution de la variable quantitative en fonction de chaque modalité de la variable qualitative. Plusieurs possibilités existent afin de faire cela. On peut tout d'abord vouloir visualiser les choses de long de l'axe vertical avec :

- des moyennes et écarts-types (cf. codes et graphiques ci-dessous) ;

```r
# Moyennes et barres d'erreur
g1 <-
  ggplot(data = iris, aes(x = Species, y = Sepal.Length)) +
  stat_summary(fun.data = "mean_sdl", fun.args = list(mult=1), geom = "errorbar", na.rm = TRUE, s
  stat_summary(fun = "mean", geom = "point", na.rm = TRUE, size = 5, color = "red") +
  ggtitle("g1") +
  theme(plot.title = element_text(size = 20),
        axis.title = element_text(size = 17),
        axis.text = element_text(size = 17))

# Moyennes, barres d'erreur, et points (l'installation du package Hmisc est nécessaire pour pouvo
g2 <-
  ggplot(data = iris, aes(x = Species, y = Sepal.Length)) +
  geom_point(size = 3) +
  stat_summary(fun.data = "mean_sdl", fun.args = list(mult=1), geom = "errorbar", na.rm = TRUE, s
  stat_summary(fun = "mean", geom = "point", na.rm = TRUE, size = 5, color = "red") +
  ggtitle("g2") +
  theme(plot.title = element_text(size = 20),
        axis.title = element_text(size = 17),
        axis.text = element_text(size = 17))

# Moyennes, barres d'erreur, et points avec mouvement latéral aléatoire (l'installation du packag
g3 <-
  ggplot(data = iris, aes(x = Species, y = Sepal.Length)) +
  geom_jitter(width = 0.08, height = NULL, size = 3) +
  stat_summary(fun.data = "mean_sdl", fun.args = list(mult=1), geom = "errorbar", na.rm = TRUE, s
  stat_summary(fun = "mean", geom = "point", na.rm = TRUE, size = 5, color = "red") +
  ggtitle("g3") +
  theme(plot.title = element_text(size = 20),
        axis.title = element_text(size = 17),
        axis.text = element_text(size = 17))
```

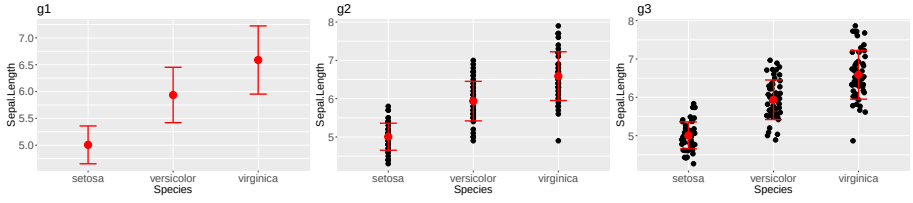


Figure 4.12: Nuage de points avec moyennes et écart-types

- des médianes et intervalles interquartiles (cf. codes et graphiques ci-dessous) ;

```
# Boîtes à moustaches
g4 <-
  ggplot(data = iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot() +
  ggtitle("g4") +
  theme(plot.title = element_text(size = 20),
        axis.title = element_text(size = 17),
        axis.text = element_text(size = 17))

# Boîtes à moustaches et points
g5 <-
  ggplot(data = iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot() +
  geom_point(size = 3) +
  ggtitle("g5") +
  theme(plot.title = element_text(size = 20),
        axis.title = element_text(size = 17),
        axis.text = element_text(size = 17))

# Boîtes à moustaches et points avec mouvement latéral aléatoire
g6 <-
  ggplot(data = iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot() +
  geom_jitter(width = 0.08, height = NULL, size = 3) +
  ggtitle("g6") +
  theme(plot.title = element_text(size = 20),
        axis.title = element_text(size = 17),
        axis.text = element_text(size = 17))
```

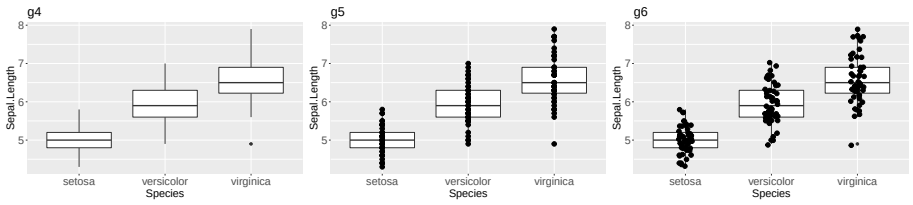


Figure 4.13: Nuage de points avec boîtes à moustaches

- des aires de densité (cf. codes et graphiques ci-dessous).

```
# Aires de densité
g7 <-
  ggplot(data = iris, aes(x = Species, y = Sepal.Length)) +
```

```
  geom_violin() +
  ggtitle("g7") +
  theme(plot.title = element_text(size = 20),
        axis.title = element_text(size = 17),
        axis.text = element_text(size = 17))

# Aires de densité et points
g8 <-
  ggplot(data = iris, aes(x = Species, y = Sepal.Length)) +
  geom_violin() +
  geom_point(size = 3) +
  ggtitle("g8") +
  theme(plot.title = element_text(size = 20),
        axis.title = element_text(size = 17),
        axis.text = element_text(size = 17))

# Aires de densité et points avec mouvement latéral aléatoire
g9 <-
  ggplot(data = iris, aes(x = Species, y = Sepal.Length)) +
  geom_violin() +
  geom_jitter(width = 0.08, height = NULL, size = 3) +
  ggtitle("g9") +
  theme(plot.title = element_text(size = 20),
        axis.title = element_text(size = 17),
        axis.text = element_text(size = 17))
```

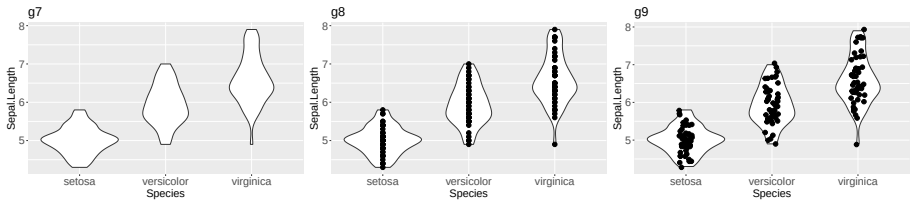


Figure 4.14: Nuages de points avec aires de densité

Les différentes lignes de graphiques ci-avant montrent des graphiques de plus en plus efficaces à mesure que l'on se déplace de la gauche vers la droite. Une bonne pratique est en effet de donner la possibilité de visualiser les données individuelles en plus des statistiques que l'on souhaiterait montrer pour résumer la distribution. Il s'agit d'une bonne pratique car une même statistique peut en réalité cacher des distributions de valeurs très différentes. Ce principe a été illustré notamment par Weissgerber (2015) qui milite pour la disparition des graphiques en forme de bâtons de dynamite, lesquels sont souvent utilisés pour montrer des moyennes et écart-types. La raison est que

ces graphiques en forme de bâtons de dynamite peuvent induire en erreur quant à la réelle forme de la distribution et ils limitent les possibilités du lecteur de juger de la pertinence des choix d'analyse qui seraient faits par la suite au regard des données d'origine.

On remarque que pour pouvoir montrer les moyennes et écart-types en rouge sur les premiers graphiques, il a fallu utiliser la fonction `stat_summary()`. Cette fonction permet de mettre en graphique certaines statistiques en configurant l'argument `fun` ou `fun.data`. Pour pouvoir enrichir les possibilités de mise en graphique de diverses statistiques, il peut falloir installer au préalable le package `Hmisc`, ce qui était nécessaire ici pour montrer les écart-types avec `fun.data = "mean_sdl"`.

Bien que l'on ait l'habitude d'utiliser des graphiques montrant les données de la variable quantitative sur l'axe vertical, il est possible de montrer les données sous formes d'aires de densité le long de l'axe horizontal (cf. graphique g13 ci-dessous) ou encore avec un graphique en lignes de crêtes (*ridgelines plot*, cf. graphique g14 ci-dessous). Ces types de graphiques et leurs intérêts ont été discutés par Wilke (2018). Pour pouvoir faire le graphque g14, il faut installer et charger un nouveau package : `ggridges`.

```
# Aire de densités superposées
g13 <-
  ggplot(data = iris, aes(x = Sepal.Length, fill = Species, color = Species)) +
  geom_density(alpha = 0.3, size = 2) +
  ggtitle("g13") +
  theme(plot.title = element_text(size = 20),
        axis.title = element_text(size = 17),
        axis.text = element_text(size = 17),
        legend.position = "bottom",
        legend.title = element_text(size = 20, face = "bold"),
        legend.text = element_text(size = 20))

# Ridgelines plot

library(ggridges)

g14 <-
  ggplot(data = iris, aes(x = Sepal.Length, y = Species)) +
  geom_density_ridges(size = 2) +
  ggtitle("g14") +
  theme(plot.title = element_text(size = 20),
        axis.title = element_text(size = 17),
        axis.text = element_text(size = 17))
```

Lorsque les données à visualiser sont pairées, c'est-à-dire qu'il s'agit des mêmes individus qui ont des données pour chaque modalité de la variable qualitative

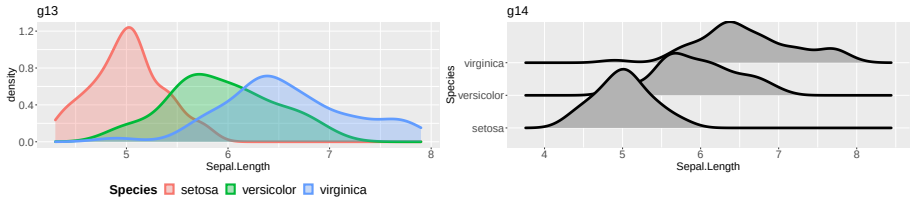


Figure 4.15: Diagrammes en lignes de crête

(e.g., suite à une même mesure qui aurait été réalisée dans différentes conditions), il est possible d'ajouter des éléments graphiques, telles que des lignes, de telle sorte à mettre davantage en évidence le fait que les données soient liées. Un exemple de graphique pour cela est montré ci-dessous avec le jeu de données `mice2` (intégré au package `datarium`). Ce jeu de données contient des données de poids de 10 souris évaluées avant et après un traitement.

```
##   id before after
## 1  1  187.2 429.5
## 2  2  194.2 404.4
## 3  3  231.7 405.6
## 4  4  200.5 397.2
## 5  5  201.7 377.9
## 6  6  235.0 445.8
```

```
mice2 %>%
  pivot_longer(cols = c(before, after), names_to = "treatment", values_to = "score") %>%
  mutate(treatment = fct_relevel(treatment, "before", "after")) %>%
  ggplot(aes(x = treatment, y = score)) +
  geom_line(aes(group = id)) +
  geom_point(size = 3, shape = 21, fill  = "white") +
  stat_summary(aes(group = 1), fun = "mean",  geom = "line", color = "red", size = 1.3) +
  stat_summary(fun.data = "mean_sdl", fun.args = list(mult=1), geom = "errorbar", na.rm = TRUE, s
  stat_summary(fun = "mean", geom = "point", na.rm = TRUE, size = 4, color = "red")
```

Dans le code ayant permis de réaliser ce graphique, il faut comprendre qu'il a été possible de réaliser des lignes reliant les différents points blancs grâce à la fonction `geom_line()` associée à un argument permettant de tracer les lignes en groupant les informations par individu avec la variable `id` (`aes(group = id)`). L'ajout d'une ligne rouge reliant chaque moyenne a été permis par l'écriture d'une ligne de code similaire à celle utilisée pour montrer les moyennes en rouge pour chaque condition avec `stat_summary()`, si ce n'est qu'on a indiqué de vouloir voir une ligne au lieu de points. La configuration de l'argument `aes(group = 1)` à l'intérieur de la fonction `stat_summary()` était nécessaire pour pouvoir montrer la ligne rouge reliant les moyennes.

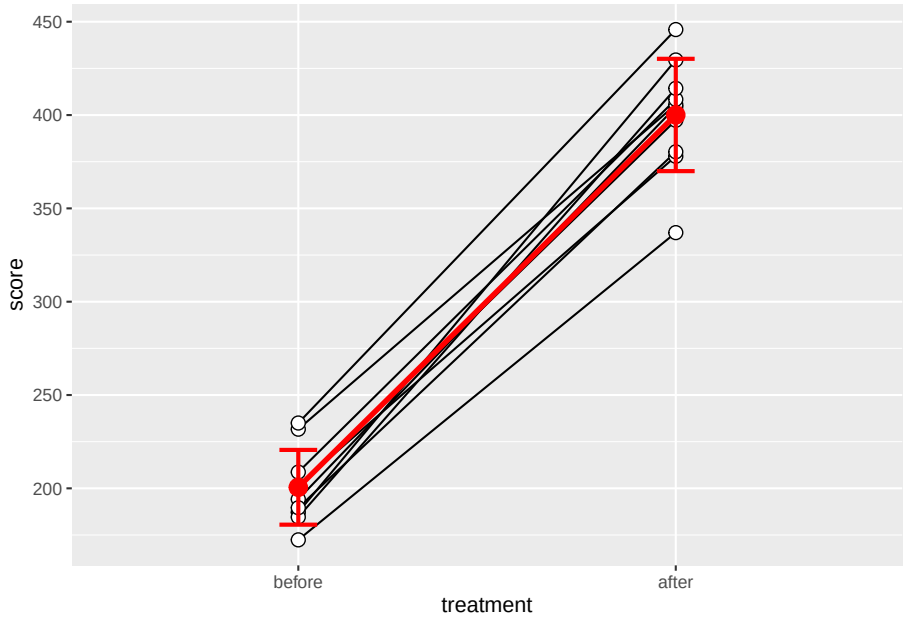


Figure 4.16: Deux groupes avec données appariées

4.3.2 Etudier numériquement la relation

De manière générale, étudier la relation entre une variable quantitative et une variable qualitative revient souvent à comparer les valeurs que prend la variable quantitative en fonction des modalités de la variable qualitative. Les analyses qui peuvent être faites dépendent du nombre de groupes de valeurs (qui dépendent du nombre de modalités) à comparer (2 ou plus), et du caractère pairé (dépendant) ou non des données. L'observation de différences de scores entre les modalités pourrait alors indiquer qu'il y a un lien entre la variable qualitative (qu'on pourrait appeler variable **facteur**) et la variable quantitative (qu'on pourrait appeler **variable réponse**). (A noter que la démonstration d'un lien de cause à effet entre la variable quantitative et la variable qualitative ne pourra être effective que si l'on a sciemment fait varier les modalités de la variable qualitative pour en observer la conséquence sur les valeurs de la variable quantitative.)

Cas de deux groupes de données indépendants (données non pairées)

De prime abord, l'analyse qui pourrait être envisagée pour comparer deux groupes de données quantitatives (e.g., des valeurs de taille) non pairées, qui seraient donc relatives à deux modalités différentes d'une variable qualitative (e.g., le sexe), serait de comparer les moyennes, ou les médianes des groupes. Toutefois, en se restreignant à cela, il pourrait être difficile de porter un

jugement sur la grandeur relative de la différence qui serait observée, qu'on pourrait appeler **taille d'effet**. Il serait également difficile de comparer cette taille d'effet avec celles observées dans d'autres études, car étant calculée de la sorte, elle serait extrêmement inhérente aux variables et valeurs mesurées dans l'étude. Il est donc intéressant, dans ce genre de situation, de standardiser la différence des moyennes ou médianes obtenues pour les deux groupes. Cette procédure de standardisation a été très développée dans le cadre de comparaisons de moyennes, et les calculs suivants, repris de l'article de D. Lakens (2013), s'inscrivent donc dans ce cadre.

ds de Cohen

Classiquement, l'indice statistique utilisé pour calculer une différence de moyennes de manière standardisée entre deux groupes de données non pairées, à partir d'échantillons de population, est le d_s de Cohen. Cette statistique se calcule en faisant la différence entre les moyennes des deux groupes à comparer, et en divisant cette différence par l'écart-type commun des valeurs de chacun des deux groupes. Ce calcul est montré ci-dessous :

$$d_s = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{(N_1-1)s_1^2+(N_2-1)s_2^2}{N_1+N_2-2}}}$$

Dans R, le d_s de Cohen peut être calculé à l'aide de la fonction `cohens_d()` du package `effectsize`, qui nécessite d'être installé puis chargé avant d'être utilisé. Pour illustrer l'utilisation de cette fonction, on peut utiliser le jeu de données `iris_two_species` crée par nos soins à partir du jeu de données `iris` pour l'exemple, qui contient notamment la variable quantitative `Sepal.Length`, et deux modalités de la variable qualitative `Species` (`setosa` et `versicolor`).

On peut alors calculer le d_s de Cohen à l'aide de la fonction `cohens_d()` de la manière suivante :

```
cohens_d(Sepal.Length ~ Species, data = iris_two_species, paired = FALSE, pooled_sd = TRUE)
```

```
## Cohen's d |         95% CI
## -------------------------
##     -2.10 | [-2.59, -1.61]
```

Dans cet exemple, on remarque qu'on a bien cherché à savoir comment les données de la variable `Sepal.Length` pouvaient différer en fonction (`~`) des modalités de la variable `Species`. Si la fonction nous donne un résultat, il faut toutefois bien faire attention au sens du calcul qui a été réalisé. Configurée de la sorte, la fonction `cohens_d()` réalise la différence **modalité 1 - modalité**

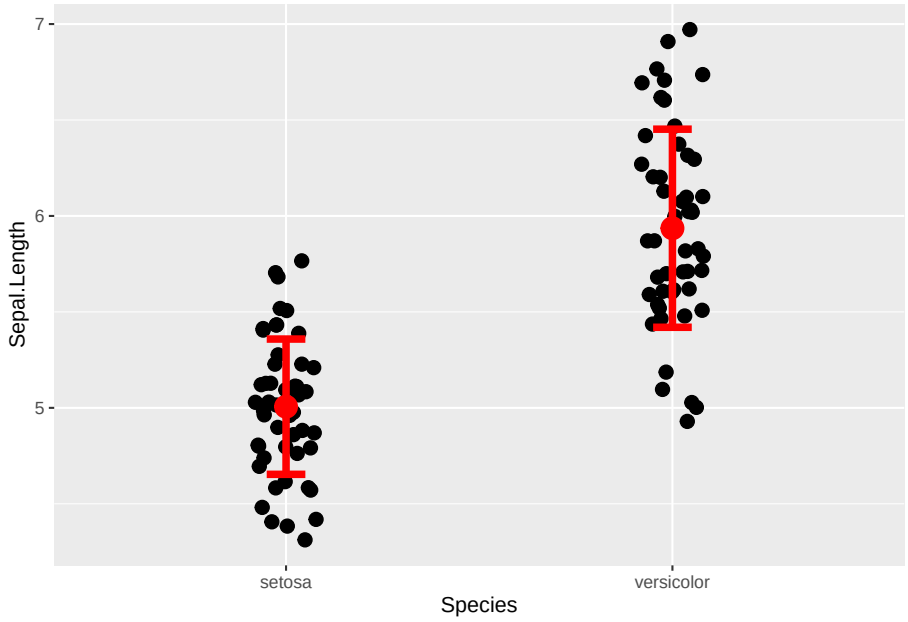


Figure 4.17: Deux groupes avec données non appariées

2. Il faut donc savoir quelle est la modalité 1 et quelle est la modalité 2 dans la variable `Species` pour ensuite pouvoir interpréter le signe du résultat, qui est négatif ici avec la valeur de -2.10. Pour ce faire, on peut utiliser la fonction `levels()` :

```
levels(iris_two_species$Species)
```

```
## [1] "setosa"     "versicolor" "virginica"
```

L'ordre des modalités affichées nous indique que `setosa` est la modalité 1, et que `versicolor` est la modalité 2. (On remarque par ailleurs que l'utilisation précédente de la fonction `filter()` a certes permis de sélectionner les données d'intérêt pour notre nouveau jeu de données `iris_two_species`, mais a malgré cela permis la conservation de toutes les modalités d'origine liées à la variable `Species` dans le jeu de données `iris`.) Par conséquent, le d_s de Cohen de -2.10 obtenu plus haut indique que la longueur des sépals (`Sepal.Length`) pour l'espèce `setosa` est inférieure à celles des sépals de l'espèce `versicolor`. Cette interprétation devrait être en cohérence avec le graphique réalisé au préalable. Si l'on avait voulu avoir le calcul inverse (`versicolor - setosa`), il aurait fallu reconfigurer l'ordre des modalités, par exemple à l'aide de la fonction `fct_relevel()` du package `forcats` comme montré à la fin du chapite 3.

Une fois que l'on a calculé une taille d'effet, il est toujours intéressant d'essayer de formuler un jugement sur l'importance, l'ampleur de l'effet. En ce sens, des valeurs seuils ont été proposées dans la littérature (Cohen, 1988; in Lakens (2013)). Ces valeurs, valables pour interpréter des tailles d'effet dans le cadre d'une étude de type *between-subject design*, sont montrées dans le tableau ci-dessous.

Ces valeurs sont relativement arbitraires, et l'interprétation de la taille de l'effet et des conséquences qu'il peut avoir en pratique ne devrait pas être liée de manière trop rigide à ces valeurs (Lakens, 2013).

Il existe également une autre approche pour interpréter une valeur de taille d'effet : l'approche *Common Language explanation* (Lakens, 2013). Cette approche consiste à faire le lien entre la valeur de la taille d'effet et les probabilités de rencontrer des valeurs similaires ou supérieures dans les groupes comparés. Par exemple, lorsqu'on obtient un d_s de Cohen de 0.80, cela peut se traduire par le fait qu'il y a 71.4 % de chances qu'une personne prise au hasard dans le groupe avec la meilleure moyenne ait un score plus élevé qu'une personne qui serait prise au hasard dans le groupe avec la moyenne la plus basse des deux groupes. Kristoffer Magnusson a réalisé une page web qui permet d'utiliser l'approche *Common Language explanation* pour n'importe quelle valeur de taille d'effet dans le cadre d'une étude de type *between-subject design*. Jetez un oeil ici : https://rpsychologist.com/cohend.

gs de Hedges

Il a été rapporté, dans la littérature, qu'en réalité le d_s de Cohen est relativement biaisé lorsqu'il s'agit d'estimer la taille d'un effet dans la population d'intérêt à partir d'échantillons de population (autrement dit, à l'échelle de nombreux échantillons, on sait qu'avec le d_s de Cohen on aura en moyenne un écart entre l'effet qu'on a trouvé et l'effet réel qui existe dans la population). Cela sera d'autant plus vrai lorsque les calculs seront réalisés à partir de petits échantillons ($N < 20$), selon Hedges et Okins (1985) cités par Lakens (2013). En raison de cela, un autre indice statistique a été proposé pour corriger cette erreur systématique qui sera d'autant plus grande que l'échantillon étudié est petit, à savoir le g_s de Hedges :

$$g_s = d_s(1 - \frac{3}{4(N_1 + N_2) - 9})$$

Dans R, le g_s de Hedges lié à un échantillon de population peut être calculé à l'aide de la fonction `hedges_g()` du package `effectsize` :

```
hedges_g(Sepal.Length ~ Species, data = iris_two_species, paired = FALSE, pooled_sd = TRUE)
```

```
## Hedge's g |         95% CI
```

```
## -------------------------
##     -2.09 | [-2.57, -1.60]
```

Ici, la valeur ne change pas beaucoup par rapport au cas précédent, car l'effectif n'est pas si petit que cela ($N_1 + N_2 = 100$ ici, ce qui fait que la correction appliquée au d_s de Cohen est minime).

Delta de Glass

Dans certains cas où l'on souhaiterait comparer les scores de deux groupes indépendants pour tester l'effet de deux conditions expérimentales différentes, l'expérimentation en tant que telle peut influencer, au-delà de la moyenne, l'écart-type de la variable réponse dans un des deux groupes. On peut se trouver dans ce genre de situation lorsque l'on compare les données post-programme d'un groupe entrainé ou traité à celles d'un groupe contrôle. En effet, le groupe entraîné/traité peut voir son écart-type changé au terme d'un programme en raison d'une réponse individuelle hétérogène à ce programme, ce qui ne sera pas en principe le cas du groupe contrôle. Dans ce genre de situation, des indices statistiques autres que le g_s de Hedges mériteraient d'être calculés, tels que le Δ de Glass (Lakens, 2013) :

$$\Delta = \frac{\overline{X_1} - \overline{X_2}}{s_2}$$

Le calcul du Δ de Glass est dans l'idée le même que celui du d_s de Cohen, sauf que la différence entre les moyennes des deux groupes n'est pas divisée par l'écart-type commun des deux groupes, mais par celui d'un seul des deux groupes, qui serait en principe celui qui représenterait la condition contrôle ou la condition de référence. Une pratique souvent recommandée pour comparer dans ce genre de situation les scores post-programme de deux groupes (un groupe entraîné/traité et un groupe contrôle) serait d'utiliser l'écart-type des scores du groupe contrôle obtenu en pré-programme (Lakens, 2013).

Dans R, le Δ de Glass lié à un échantillon de population peut être calculé à l'aide de la fonction `glass_delta()` du package `effectsize`. Attention, l'écart-type utilisé dans le code suivant est celui de la variable quantitative associée à la modalité 2 de la variable qualitative, qui est toujours `versicolor` dans cet exemple.

```
glass_delta(Sepal.Length ~ Species, data = iris_two_species)
```

```
## Glass' delta |         95% CI
## -----------------------------
##        -1.80 | [-2.26, -1.33]
```

Cas de deux groupes de données dépendants (données pairées)

dz et dav de Cohen

Le calcul classique pour obtenir la taille d'effet désignant l'écart entre deux groupes de données dépendantes est celui du d_z (Lakens, 2013), qui est montré ci-dessous :

$$d_z = \frac{\overline{X}_{diff}}{s_{diff}},$$

$\overline{X}_{diff}$ désignant la moyenne des différences relatives à chaque pair de valeurs obtenues dans les deux conditions comparées, et s_{diff} désignant l'écart-type de ces différences. Dans R, le d_z peut être obtenu à nouveau avec la fonction `cohens_d()` du package `effectsize`. Pour illustrer cela, nous allons cette fois utiliser le jeu de données `Blink` associé au package `PairedData` qui doit être installé et chargé pour être utilisé. Pour information, `Blink` contient les données de taux de clignotement des yeux obtenues chez 12 sujets et dans deux conditions différentes : 1/ tâche où il fallait diriger un stylo selon une trajectoire rectiligne (modalité `Straight`) ; 2/ tâche où il fallait diriger un stylo selon une trajectoire avec des oscillations (`Oscillating`).

```
##   Subject Straight Oscillating
## 1     S01     24.0        15.0
## 2     S02     19.5         6.6
## 3     S03      8.2         1.9
## 4     S04      8.5         1.5
## 5     S05     12.1         1.1
## 6     S06      8.0         2.5
```

Pour la suite de l'analyse, on peut reconfigurer ce jeu de données de telle sorte à bien avoir les conditions testées dans une même colonne et les valeurs correspondantes dans la colonne d'à côté, avec la modalité "Straight" en tant que première modalité à considérer dans les fonctions à suivre. Le nouveau jeu de données s'appellerait alors `Blink2`.

```
## # A tibble: 24 x 3
##    Subject Condition   Blink_rate
##    <fct>   <fct>            <dbl>
##  1 S01     Straight          24
##  2 S01     Oscillating       15
##  3 S02     Straight          19.5
##  4 S02     Oscillating        6.6
##  5 S03     Straight           8.2
```

```
##  6 S03     Oscillating        1.9
##  7 S04     Straight           8.5
##  8 S04     Oscillating        1.5
##  9 S05     Straight          12.1
## 10 S05     Oscillating        1.1
## # ... with 14 more rows
```

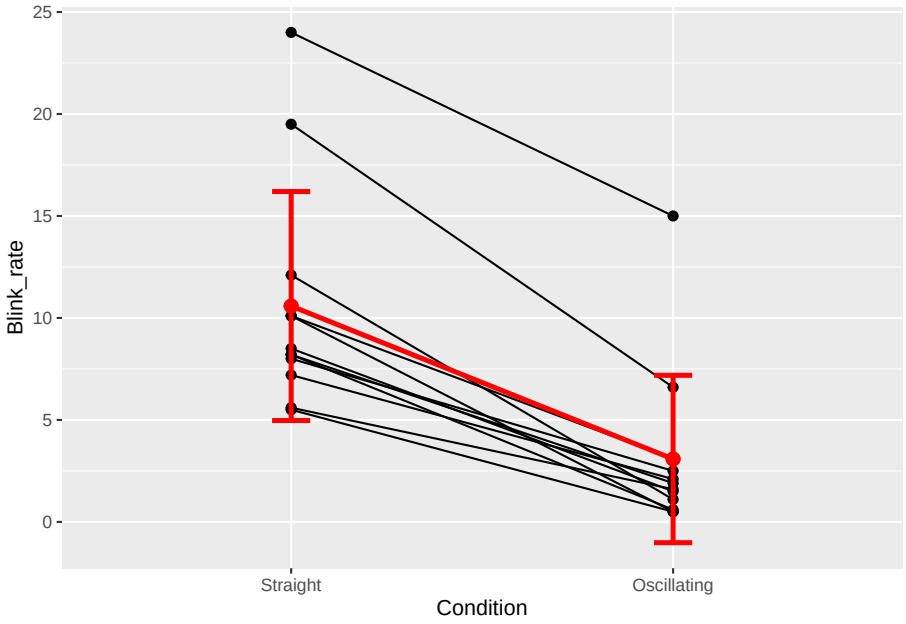


Figure 4.18: Données du jeu de données 'Blink'

Dans R, on peut alors calculer d_z à nouveau à l'aide de la fonction `cohens_d()` du package `effectsize` comme ceci :

```
cohens_d(Blink_rate ~ Condition, data = Blink2, paired = TRUE)
```

```
## Cohen's d |       95% CI
## ------------------------
##      2.81 | [1.58, 4.27]
```

Toutefois, d'après Lakens (2013), il peut être intéressant de rapporter plutôt un indice de taille d'effet qui ne serait pas influencé par le fait que les données des deux groupes à comparer soient corrélées, notamment en vue de rendre comparable cette taille d'effet à celles obtenues dans le cadre d'études de type *between-subject design*, pour plus facilemement conduire des méta-analyses par

la suite par exemple. La recommandation de Lakens (2013) serait alors d'utiliser l'indice d_{av}, dont le calcul serait le suivant :

$$d_{av} = \frac{\overline{X}_{diff}}{\sqrt{\frac{s_1^2+s_2^2}{2}}}$$

Ce calcul reviendrait donc à diviser la moyenne des différences par l'écart-type moyen relatif aux deux groupes de données. Pour obtenir d_{av}, il est possible de repartir de la fonction `cohens_d()` et de sa configuration pour des groupes indépendants, mais en indiquant `pooled_sd = FALSE` dans la fonction, comme ci-dessous :

```
cohens_d(Blink_rate ~ Condition, data = Blink2, paired = FALSE, pooled_sd = FALSE)
```

```
## Cohen's d |       95% CI
## ------------------------
##      1.53 | [0.59, 2.43]
```

En effet, que l'on fasse la moyenne des différences (ce qui serait attendu pour le présent calcul) ou la différence entre deux moyennes (ce qui est finalement fait par la fonction tel que configurée ci-dessus), cela ne change rien avec des données pairées. Le numérateur $\overline{X}_{diff}$ reste donc correctement calculé même avec cette configuration de fonction qui est faite initialement pour des groupes indépendants. Le fait d'indiquer `pooled_sd = FALSE` change le calcul du dénominateur que nous avions rencontré initialement pour le calcul de d_s, cela pour obtenir le calcul attendu pour le d_{av}.

gav de Hedges

Comme dans le cas de l'étude de groupes indépendants, d_{av} et ses dérivés sont biaisés. Il peut donc être préférable à nouveau de calculer le g_{av} de Hedges avec la fonction `hedges_g()` configurée comme ceci :

```
hedges_g(Blink_rate ~ Condition, data = Blink2, paired = FALSE, pooled_sd = FALSE)
```

```
## Hedge's g |       95% CI
## ------------------------
##      1.47 | [0.57, 2.35]
```

Toutefois, comme le rappelle Lakens (2013), le g_{av} de Hedges ne serait pas complètement non biaisé.

Table 4.3: Les rangs d'une variable

hp	hp_rank
52	1.0
62	2.0
65	3.0
66	4.5
66	4.5
91	6.0
93	7.0
95	8.0
97	9.0
105	10.0
109	11.0
110	13.0
110	13.0
110	13.0
113	15.0
123	16.5
123	16.5
150	18.5
150	18.5
175	21.0
175	21.0
175	21.0
180	24.0
180	24.0
180	24.0
205	26.0
215	27.0
230	28.0
245	29.5
245	29.5
264	31.0
335	32.0

Table 4.4: Taille d'effet pour une différence de moyennes

Petit	Moyen	Grand
0.2	0.5	0.8

Chapter 5

Régressions

5.1 Régression linéaire simple

Il est possible d'investiguer l'existence d'une relation linéaire entre deux variables en modélisant cette relation à l'aide d'une équation de type $Y = aX + b$, et en calculant certaines statistiques associées qui rendent compte du niveau de correspondance entre le modèle linéaire et les données étudiées. Ces statistiques sont le coefficient de détermination, noté R^2, et l'erreur typique d'estimation, dont on gardera l'acronyme anglais SEE (pour *Standard Error of Estimate*).

5.1.1 Le coefficient de détermination

Le coefficient de détermination, noté R^2, représente la part de variance de la variable Y expliquée par le modèle linéaire concerné. La formule de ce coefficient peut être présentée comme ceci :

$$R^2 = 1 - \frac{Var(\hat{Y} - Y)}{Var(Y)} = 1 - \frac{Var(RES)}{Var(Y)},$$

où $\hat{Y}$ désigne les prédictions faites à partir du modèle, et Y désigne les valeurs réelles que l'on a cherché à prédire à partir du modèle. Le terme $\hat{Y} - Y$ (ou RES) doit se concevoir comme une variable contenant toutes les différences $\hat{Y}i - Yi$ qu'on appelle des **résidus**. Ainsi, le terme $Var(\hat{Y} - Y)$ désigne la variance des résidus (ou encore la variance des erreurs). Au final, le ratio $\frac{Var(\hat{Y}-Y)}{Var(Y)}$ traduit la part de variance non expliquée (non détectée) par le modèle, et le R^2 se calcule en faisant 1 moins ce ratio. (A noter qu'on peut

trouver ailleurs d'autres manières de présenter ce coefficient R^2, avec des formules initiales différentes, mais mathématiquement, ces différentes manières d'aborder les choses restent équivalentes).

La figure ci-dessous vise à illustrer la notion de **résidu** et ce qu'elle représente dans le calcul du R^2. Sur cette figure, les points représentent les valeurs Yi en fonction des valeurs Xi, la ligne bleue représente le modèle de régression linéaire (i.e., toutes les valeurs $\hat{Y}i$ qui seraient prédites à partir du modèle et des valeurs Xi), et les segments rouges représentent les résidus (i.e., la différence qu'on a à chaque fois entre $\hat{Y}i$ et Yi). Pour un modèle donné, plus ces segments rouges seront nombreux et grands, plus cela signifiera que les erreurs de prédiction du modèle sont nombreuses et grandes, que la part de variance non expliquée par le modèle est grande, et que la valeur du R^2 pour ce modèle est éloignée de 1. Ainsi, le coeffient R^2 peut aller de la valeur 0 (signifiant que le modèle n'explique aucune variation de Y), à la valeur de 1 (signifiant que le modèle explique toute les variations de Y). Plus la valeur de R^2 d'un modèle linéaire se rapprochera de 1, plus cela suggérera que la relation étudiée est effectivement linéaire. (On note par ailleurs que le coefficient de détermination R^2 associé à un modèle linéaire est mathématiquement lié au coefficient de corrélation de Pearson (r), r étant la racine carrée du R^2.)

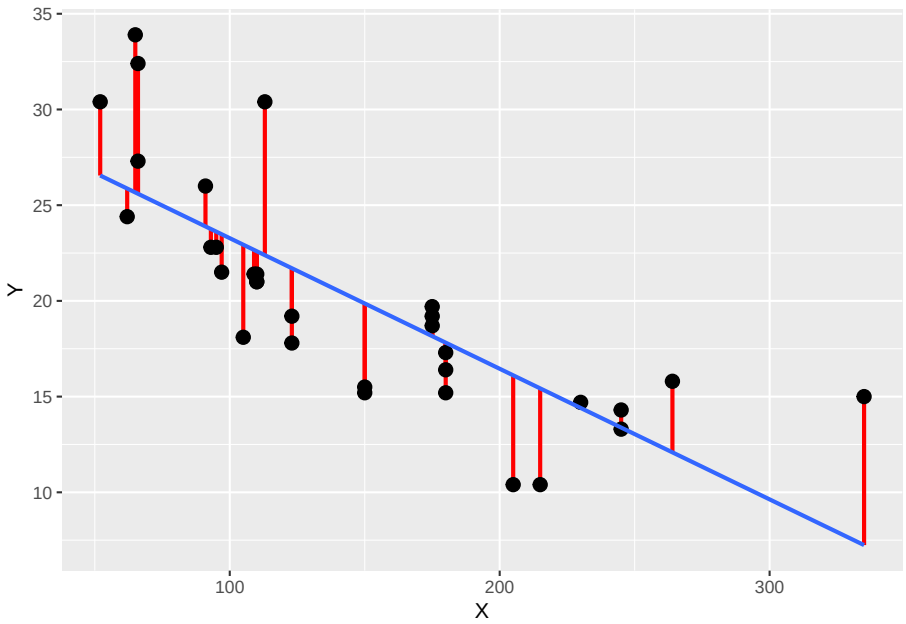


Figure 5.1: Résidus d'un modèle

Pour déterminer le R^2 d'un modèle linéaire avec le logiciel R, il faut d'abord créer ce modèle à l'aide de la fonction `lm()`. L'usage simple de cette fonction, tel que montré ci-dessous, permet de prendre connaissance des coefficients du

modèle. Dans les résultats issus de l'exemple ci-dessous, l'ordonnée à l'origine est située sous `(Intercept)`, et le coefficient directeur est situé sous le nom de la variable X du modèle, ici `hp`. Dans l'exemple ci-dessous, qui utilise à nouveau le jeu de données `mtcars`, le modèle nous indique que lorsque `hp` vaudra 0, l'estimation de `mpg` vaudra 30.09886, et que pour chaque augmentation d'unité de `hp`, on aura une diminution de -0.06823 unité de `mpg`.

```
lm(mpg ~ hp, data = mtcars)
```

```
##
## Call:
## lm(formula = mpg ~ hp, data = mtcars)
##
## Coefficients:
## (Intercept)           hp
##    30.09886     -0.06823
```

Pour plus de confort dans l'écriture de la suite du code, il peut être intéressant d'associer le modèle crée avec la fonction `lm()` à un nom. Pour accéder aux différentes informations statistiques résumant le modèle, on peut alors utiliser la fonction `summary()` avec le nom choisi pour le modèle.

```
model <- lm(mpg ~ hp, data = mtcars)
summary(model)
```

```
##
## Call:
## lm(formula = mpg ~ hp, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.7121 -2.1122 -0.8854  1.5819  8.2360
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 30.09886    1.63392  18.421  < 2e-16 ***
## hp          -0.06823    0.01012  -6.742 1.79e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.863 on 30 degrees of freedom
## Multiple R-squared:  0.6024, Adjusted R-squared:  0.5892
## F-statistic: 45.46 on 1 and 30 DF,  p-value: 1.788e-07
```

Dans la liste d'informations données suite à l'activation du code, on retrouve notamment les coefficients déjà rencontrés plus haut, et on peut trouver le coefficient R^2 en face de l'écriture *Multiple R-squared*. On peut aussi y voir l'erreur typique d'estimation en face de l'écriture *Residual standard error*.

5.1.2 L'erreur typique d'estimation

L'erreur typique d'estimation, ou SEE, représente l'écart-type des erreurs d'estimation associées à l'utilisation d'un modèle. Son unité est donc celle de la variable Y que l'on a cherché à prédire avec le modèle. La formule suivante permet d'expliquer son calcul à partir de données prélevées sur un échantillon :

$$SEE = \sqrt{\frac{\sum_{i=1}^{N}(RESi - \overline{RES})^2}{N-2}},$$

où $RESi$ désigne le résidu pour une observation donnée, $\overline{RES}$ la moyenne des résidus, et N le nombre d'observations.

5.1.3 Graphique récapitulatif

Il est possible d'extraire l'ordonnée à l'origine et la pente (i.e., le coefficient directeur) du modèle de régression, le coefficient R^2, et la statistique SEE, à partir de la liste d'informations obtenue avec la fonction `summary()`. Le code ci-dessous montre comment faire cela avec l'exemple concernant le jeu de données `mtcars` :

```
# Extraction de l'ordonnée à l'origine
intercept <- summary(model)$coefficients[1]
intercept
```

```
## [1] 30.09886
```

```
# Extraction du coefficient directeur
slope <-  summary(model)$coefficients[2]
slope
```

```
## [1] -0.06822828
```

```
# Extraction du R2
R2 <- summary(model)$r.squared
R2
```

```
## [1] 0.6024373
```

```
# Extraction de SEE
SEE <- summary(model)$sigma
SEE
```

```
## [1] 3.862962
```

Une fois extraites et associées à des noms, ces informations peuvent ensuite être réutilisées avec le package `ggplot2` et la fonction `annotate()` pour compléter le graphique initial avec des informations statistiques.

```
ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  annotate("text", label = bquote(paste("Y = ", .(round(slope, digits = 3)), "X + ", .(round(inte
                                        R^2, " = ", .(round(R2, digits = 3)),
                                        " ; SEE = ", .(round(SEE, digits = 3)))),
           x = 50, y = 35, hjust = 0, size = 5)
```

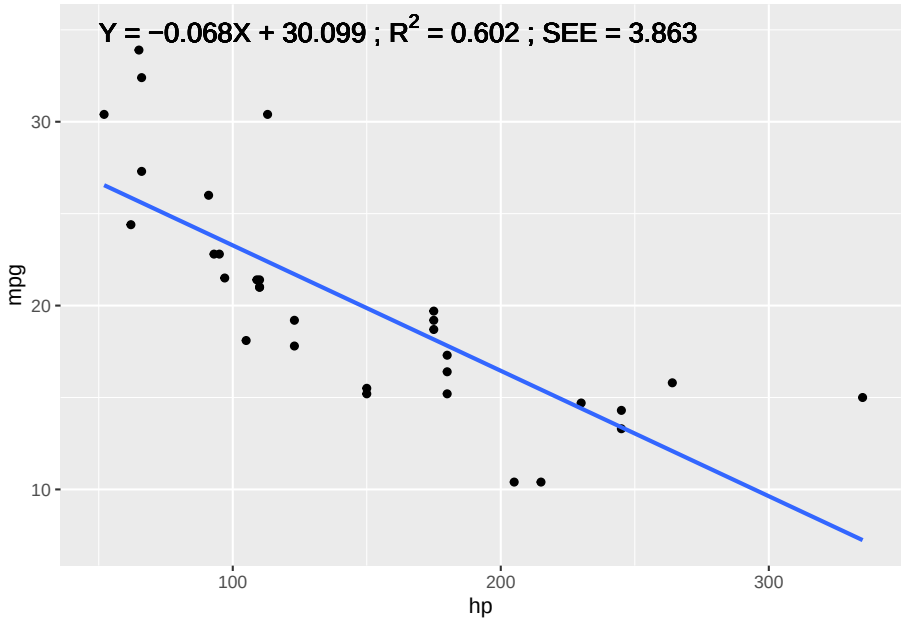


Figure 5.2: Régression linéaire avec les informations correspondantes

La leçon du quartet d'Anscombe

Encore une fois, lorsqu'on étudie un phénomène, ici l'existence d'une relation linéaire, il est important de d'abord faire un graphique montrant les données. Cette première étape graphique est importante car les valeurs numériques qui peuvent être obtenues pour le coefficient R^2 (et donc aussi pour le coefficient de corrélation de Pearson), et la statistique SEE, ne peuvent à elles seules garantir l'aspect linéaire d'une relation. Un exemple qui permet d'illustrer cela est le quartet d'Anscombe (1973). Il s'agit de quatre jeux de données dont les représentations graphiques sont montrées ci-dessous.

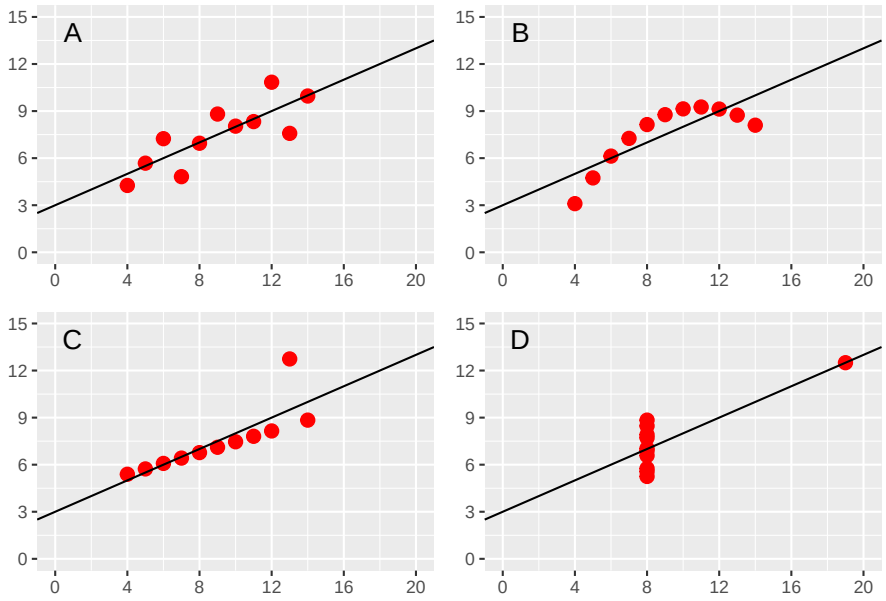


Figure 5.3: Le quartet d'Anscombe

Bien que d'aspects très différents, ces jeux de données montrent pourtant des variables en abscisses qui ont toutes la même moyenne ($\overline{X} = 9$) et le même écart-type ($SD = 3.32$), des variables en ordonnées qui ont elles aussi la même moyenne ($\overline{Y} = 7.5$) et le même écart-type = ($SD = 2.03$), et des lignes de régression linéaire qui ont toutes la même équation ($Y = 0.5X + 3$), le même coefficient de détermination ($R^2 = 0.67$) et la même erreur typique d'estimation ($SEE = 1.24$). Pour autant, on observe que seul le premier jeu de données (cf. graphique A ci-dessus) est associé à un modèle linéaire vraiment pertinent. En effet, le graphique B montre bien que la relation n'est pas linéaire mais plutôt quadratique, le graphique C montre que la régression est anormalement influencée par une valeur extrême, et le graphique D montre qu'il n'y a en réalité pas de relation linéaire entre les deux variables et que celle-ci ne semble exister numériquement que grâce à une seule valeur très extrême. Autant le graphique C invite à conserver une analyse de régression

linéaire avec éventuellement certains ajustements à réaliser, autant les graphiques B et D indiquent qu'un modèle linéaire n'est pas pertinent pertinent en l'état pour caractériser la relation entre les deux variables étudiées.

Anscombe, F. J. (1973). Graphs in Statistical Analysis. *The American Statistician*, *27*(1), 17–21.

Bickel, P. J., Hammel, E. A., & O'Connell J, W. (1975). Sex bias in graduate admissions: data from berkeley. *Science*, *187*(4175), 398–404. https://doi.org/10.1126/science.187.4175.398

Chatellier, G., & Durieux, P. (2003). Moyenne, médiane, et leurs indices de dispersion : quand les utiliser et comment les présenter dans un article scientifique ? *Rev Mal Respir*, *20*(3), 421–424. https://doi.org/RMR-06-2003-20-3-0761-8425-101019-ART17

Dart, T., & Chatellier, G. (2003). Comment décrire la distribution d'une variable ? *Rev Mal Respir*, *20*(6), 946–951. https://doi.org/RMR-12-2003-20-6-0761-8425-101019-ART19

Gonzales, V. A., & Ottenbacher, K. J. (2001). Measures of central tendency in rehabilitation research: what do they mean? *Am J Phys Med Rehabil*, *80*(2), 141–146. https://doi.org/10.1097/00002060-200102000-00014

Grenier, E. (2007). Quelle est la « bonne » formule de l'écart-type ? *Revue MODULAD*, *37*, 102–105.

Halperin, S. (1986). Spurious correlations–causes and cures. *Psychoneuroendocrinology*, *11*(1), 3–13. https://doi.org/10.1016/0306-4530(86)90028-4

Hopkins, W. G., Marshall, S. W., Batterham, A. M., & Hanin, J. (2009). Progressive statistics for studies in sports medicine and exercise science. *Med Sci Sports Exerc*, *41*(1), 3–13. https://doi.org/10.1249/MSS.0b013e31818cb278

Joanes, D. N., & Gill, C. A. (1998). Comparing measures of sample skewness and kurtosis. *The Statistician*, *47*(Part 1), 183–189. https://doi.org/10.1111/1467-9884.00122

Labreuche, J. (2010). Les différents types de variables, leurs représentations graphiques et paramètres descriptifs. *Sang Thrombose Vaisseaux*, *22*(10), 536–543. https://doi.org/10.1684/stv.2010.0541

Lakens, D. (2013). Calculating and reporting effect sizes to facilitate cumulative science: a practical primer for t-tests and ANOVAs. *Front Psychol*, *4*, 863. https://doi.org/10.3389/fpsyg.2013.00863

Navarro, D. (2018). *Learning statistics with R*. UNSW Computational Cognitive Science.

Rousselet, G. A., & Wilcox, R. R. (2020). Reaction times and other skewed distributions: problems with the mean and the median. *Meta-Psychology*, *4*, 1–39. https://doi.org/10.15626/MP.2019.1630

Weissgerber, T. L., Milic, N. M., Winham, S. J., & Garovic, V. D. (2015). Beyond bar and line graphs: time for a new data presentation paradigm. *PLoS Biol*, *13*(4), e1002128. https://doi.org/10.1371/journal.pbio.1002128

Wickham, H. (2016). *ggplot2* (2nd ed.). Springer-Verlag. https://doi.org/10.1007/978-0-387-98141-3

Wickham, H., & Grolemund, G. (2017). *R for Data Science*. O'Reilly.

Wilke, C. O. (2018). *Fundamentals of data visualization*. O'Reilly Media, Inc. Retrieved from https://clauswilke.com/dataviz.