Travaux dirigés d'Informatique

Lience 3 parcours Economie-Finance

Ewen Gallice

2022-2023

Table des matières

Liste des tableaux

Introduction

Ce livret est conçu pour les séances de travaux dirigés d'Informatique en 3e année de Licence mention Economie et Gestion, à Aix-Marseille Université.

Chaque chapitre correspond à une séance ou deux de travaux dirigés. Le contenu alterne entre des courtes présentations de concepts et des exercices pour apprendre à mettre en pratique les concepts présentés.

Pour appréhender de manière aprofondie le contenu de ce livret, le lecteur ou la lectrice est invité(e) à consulter les notes de cours, disponibles à l'adresse suivantes : http://egallic.fr/Enseignement/R/Book/avant-propos.html.

Les conventions de lecture suivantes seront adoptées.

Les boîtes oranges désignent, en début de chapitre, les lectures recommandées des notes de cours.

Les boîtes jaunes indiquent des notes.

Les boîtes rouges indiquent des avertissements.

Les boîtes vertes indiquent des exercices.

Chapitre 1

Initiation à R

1.1 Travailler par projets

Durant vos études universitaires, vous allez réaliser des études en vous appuyant sur des données et sur leur analyse statistique. Il convient de prendre de bonnes habitudes en travaillant avec des projets RStudio. Dans cet exercice, vous allez mettre en place un projet pour la première séance de travaux dirigés.

En respectant l'arborescence présentée sur l'image ci-dessous, il sera aisé de vous repérer et le partage de vos codes avec vos collaborateurs sera simple.

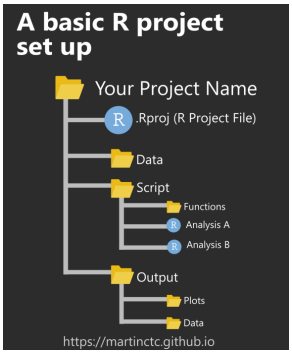


Figure 1.1 – Structure basique pour les projets. Source : https://martinctc.github.io/.

Création d'un projet
Ouvrez le logiciel RStudio :
— Cliquez sur le menu `File`, puis sur `New Project...`.

— Cliquez sur `New Directory`, puis sur `New Project`.
— Donnez un nom au nouveau projet, puis cliquez sur le bouton `Browse...`
— Choisissez l'emplacement du projet, puis appuyez sur le bouton `Open`.
— Cliquez sur le bouton `Create Project` pour créer le projet. Une nouvelle session RStudio s'ouvre alors. Le répertoire courant devient celui dans lequel vous avez créé le projet. Un fichier d'extention `.Rproj` a été créé dans ce répertoire. Il suffira d'ouvrir ce fichier à l'avenir pour ouvrir RStudio pour travailler sur ce projet.
— À présent que le projet est créé, il est temps de créer un fichier de script : dans RStudio, cliquez sur le menu `File`, puis sur `New File` et enfin sur `R Script` (de manière équivalente, vous pouvez cliquer sur l'icône présentant un document blanc et une croix blanche sur fond vert).
— Enfin, enregistrez votre fichier : menu `File`, puis `Save`.

1.2 Création, modification, suppression d'un objet

Pour créer un objet, il est nécessaire de lui donner un nom et un contenu. On écrit le nom en respectant des conventions de nommage. Le nom d'une variable doit être composé de caractères alphanumériques ou du point ou du trait de soulignement uniquement. Il ne doit pas commencer par un chiffre ou contenir d'espace, sauf s'il est entouré par des guillemets. Il est important de noter que ces noms sont sensibles à la casse, c'est à dire qu'une distinction entre les majuscules et les minuscules a lieu.

Dans le cadre de ces travaux dirigés, nous utiliseront une convention supplémentaire, consistant à créer des variables dont on peut facilement deviner le contenu grâce à leur nom, en utilisant uniquement des lettres minuscules, et en ajoutant un trait de soulignement entre les différents termes s'il y a lieu d'utiliser plusieurs termes. Par exemple, une variable qui contiendrait l'année de naissance d'un individu pourrais être nommée `annee_naissance`.

L'affectation du contenu à une variable, une fois son nom indiqué, se fait en ajoutant un symbole d'affectation : `<-` (on peut aussi utiliser le symbole égal (`=`)).

```
x <- 1+1
x
```

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

Exercice

1. Créer un objet que vous nommerez `age` qui contiendra, en valeur numérique,

votre âge.
2. Affichez dans la console la valeur de l'objet `age`.
3. Créez un objet appelé `annee_naissance` qui, en s'appuyant sur la valeur contenue dans `age`, indiquera votre année de naissance (soustrayez la valeur de `age` à l'année actuelle)
4. Modifiez le contenu de l'objet `age` en lui ajoutant 10 ans. Affichez dans la console la valeur de l'objet `age` suite à la modification.
5. Affichez la valeur de l'objet `annee_naissance` : que constatez-vous ?
6. Modifiez le contenu de l'objet `age` en lui soustrayant 10 ans. Affichez dans la console la valeur de l'objet `age` suite à la modification.
7. Calculez, sans modifier le contenu de l'objet `age`, le nombre de jours correspondant à votre âge (en multipliant votre âge par 365,25).
8. En faisant appel à la fonction `ls()`, affichez le nom des objets existants dans l'environnement global.
9. À l'aide de la fonction `rm()` (*remove*), supprimez l'objet `age`, puis tentez d'afficher le contenu de l'objet `age` dans la console.
10. Affichez à nouveau le nom des objets existants dans l'environnement global.
11. Affichez la valeur de l'objet `annee_naissance` : que constatez-vous ?

1.3 Lecture d'une page d'aide

Dans cet exercice, vous allez calculer la racine carrée d'un nombre. Pour cela, vous allez faire appel à une fonction native en R, dont vous ignorez le nom et le fonctionnement (si c'est la première fois que vous faites du R).

Lorsque vous n'arrivez pas à effectuer une opération avec R, réfléchissez **en anglais** à votre question, puis consultez les pages d'aide ou bien un moteur de recherche en ligne.

Exercice

1. Comment dit-on "racine carrée" en anglais ?
2. Sur le moteur de recherche que vous avez l'habitude d'utiliser, inscrivez la traduction de "racine carrée" en anglais, suivi de "in r".
3. Repérez le nom de la fonction qui vous permet de calculer une racine carrée en R, puis affichez (et parcourez) la page d'aide de cette fonction dans RStudio.
4. Calculez la racine carrée de 9 à l'aide de la fonction dont vous venez de consulter la page d'aide.

1.4 Installation et chargement d'un package

Les *packages* sont des jeux de fonctions, accompagnés de fichiers d'aides, parfois de jeux de données. Pour pouvoir faire appel à une fonction contenue dans un *package*, il faut soit :

— indiquer le nom du *package* suivi de deux fois deux-points puis du nom de la fonction (*e.g.*, `dplyr::select(iris, "Species")` qui fait appel à la fonction `select()` du *package* {dplyr})
— charger le *package* en mémoire à l'aide de la fonction `library()` puis faire appel au nom de la fonction (*e.g.*, `library(dplyr)` puis dans une seconde instruction, `select(iris, "Species")`).

— Lorsque l'on charge un *package* à l'aide de la fonction `library()`, l'ensemble des fonctions contenues dans ce package sont chargées en mémoire et les pages d'aide associées deviennent consultables.
— Lorsque l'on quitte R et qu'on le relance, les *packages* chargés à l'aide de la fonction `library()` ne sont plus en mémoire, et il faut de nouveau les charger pour pouvoir accéder aux fonctions qu'ils contiennent.

Pour pouvoir charger un *package*, il est nécessaire que celui-ci soit installé sur votre machine. Pour installer un package disponible sur le dépôt CRAN (*The Comprehensive R Archive Network*), il suffit d'utiliser la fonction `install.packages()` et de lui donner le nom du *package* sous la forme d'une chaîne de caractères (en entourant le nom par des guillemets doubles ou simples).

— L'installation d'un *package* (`install.packages()`) se fait une seule fois, contrairement au chargement d'un *package* (`library()`) qui s'effectue à chaque fois que l'on relance R.
— L'appel de la fonction `install.packages()` sert également à mettre à jour un *package*.

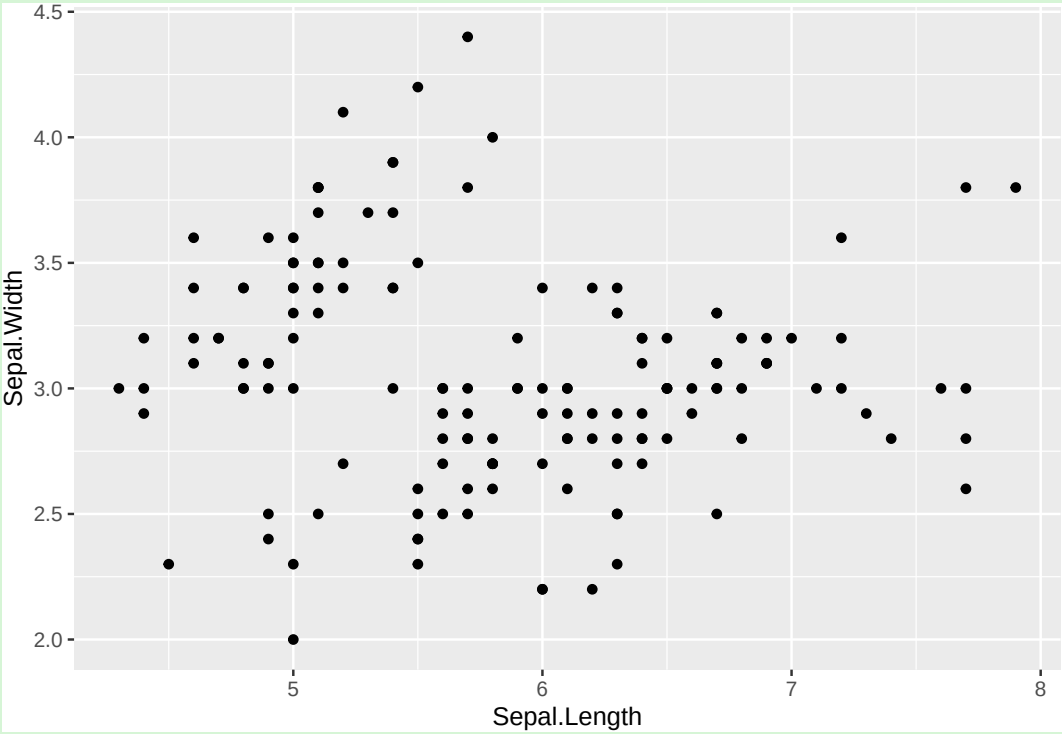
Exercice

1. Évaluez le coode suivant, et lisez le message d'erreur :

```
ggplot(data = iris, mapping = aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point()
```

2. Les fonctions `ggplot()`, `aes()` et `geom_point()` sont contenues dans le *package* {ggplot2}. **Sans charger** le *package*, et en utilisant la syntaxe avec les

deux-points (`::`), modifiez l'instruction de la question précédente pour afficher un nuage de point comme ci-dessous :



3. Chargez le *package* {ggplot2} à l'aide de la fonction `library()` et évaluez à nouveau le même code qu'en première question.

Chapitre 2

Type des données et première structure

2.1 Types de données

La réalisation de cet exercice nécessite la lecture des sections 2.1 Types de données du cours et 2.2.1.1 Vecteurs

R gère des objets qui disposent de 4 caractéristiques :

- un nom ;
- un type : la nature des éléments ; les 4 types principaux sont les suivants :
  - *numeric* : des nombres entiers (on ajoute un L majuscule après le nombre pour le déclarer comme entier `1L`) ou réels (`1.4` ou `1` ou `1.0`),
  - *character* : des chaînes de caractère (que l'on créé à l'aide de guillemets doubles ou de guillemets simples),
  - *logical* : des valeurs logiques (`TRUE`, `FALSE` et `NA`),
  - *complex* : des nombres complexes (`2+3i`) ;
- une longueur ;
- un contenu.

Par ailleurs, R propose un objet `NULL`, de longueur 0 et de type `NULL`. Il représente le vide.

Enfin, R propose de marquer les valeurs manquantes avec le mot réservé `NA`. Comme indiqué plus haut, il s'agit d'un objet de type logique.

Il est important de connaître le type des données que l'on manipule : chaque objet dispose d'une classe qui dépend du type. Or, certaines opérations sont réalisables pour certaines

classes mais pas pour d'autres. Par exemple, la fonction `abs()` qui calcule la valeur absolue, est applicable à des objets de type numérique mais ne fonctionne pas pour des chaînes de caractères.

```
x <- 3
abs(x)
```

```
## [1] 3
```

```
y <- "toto"
abs(y)
```

```
## Error in abs(y): argument non numérique pour une fonction mathé
    matique
```

Note :
Quelques fonctions à connaître concernant les types de données :

— Pour savoir le type d'un objet : `typeof()`

```
x <- 3
typeof(x)
```

```
## [1] "double"
```

```
y <- "hello"
typeof(y)
```

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

— Pour savoir si un objet est d'un type spécifique, R propose des fonctions retournant une valeur logique. La syntaxe de ces fonctions est simple : on ajoute le suffixe `is.` au nom du type.

```
is.character(x)
```

```
## [1] FALSE
```

```
is.numeric(x)
```

```
## [1] TRUE
```

```
is.na(x)
```

```
## [1] FALSE
```

— La longueur d'un objet est obtenue avec la fonction `length()`. Pour le moment, les objets que nous créons dans cette sectionsont tous de longueur 1. Nous verrons dans la section suivante qu'il s'agit simplement de vecteurs de longueur 1.
— Pour forcer le changement de type d'un objet, R propose des fonctions à la syntaxe simple : il suffit d'ajouter le préfixe `as.` au type voulu. Lorsqu'une conversion n'est pas possible, R retourne la valeur `NA`

```
z <- as.character(x)
z
```

```
## [1] "3"
```

```
typeof(z)
```

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

```
as.numeric("hello")
```

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

Exercice

1. Créez les objets suivants :
   — `x_1` : un objet contenant la valeur `10`,
   — `x_2` : un objet contenant la valeur `20`,
   — `y` : un objet contenant la chaîne de caractères `10`,
   — `z` : un objet contenant la valeur `NULL`,
   — `t` : un objet contenant la valeur `NA`,
   — `v` : un objet contenant la valeur logique `TRUE`,
   — `f` : un objet contenant la valeur logique `FALSE`.

2. Dans un objet que vous nommerez `s`, additionnez `x_1` et `x_2`, puis affichez la valeur de `s` dans la console.
3. Dans un objet que vous nommerez `s_2`, additionnez `x_1` et `y`, puis affichez la valeur de `s_2` dans la console. Lisez l'erreur.
4. Changez le type de `y` de sorte que l'objet ne soit plus une chaîne de caractères mais une valeur numérique.
5. Dans un objet que vous nommerez `s_3`, additionnez `x_1` et `z`, puis affichez la valeur de `s_3` dans la console. Vérifiez, à l'aide de la fonction adéquate, la longueur de `s_3`.
6. Dans un objet que vous nommerez `s_4`, additionnez `x_1` et `t`, puis affichez la valeur de `s_4` dans la console.
7. Divisez la valeur `x_1` par 0 et regardez la valeur retournée.
8. Affichez, sans modifier le contenu de `v` et `f`, le résultat de l'application de la fonction `as.numeric()` aux objets `v` puis `f`.
9. Dans un objet que vous nommerez `s_5`, additionnez `v` et `f`, puis affichez la valeur de `s_5` dans la console.
10. Dans un objet que vous nommerez `s_6`, additionnez `v` et `v`, puis affichez la valeur de `s_6` dans la console.
11. Dans un objet que vous nommerez `s_7`, multipliez `v` et `f`, puis affichez la valeur de `s_7` dans la console.

2.2 Structure de données : les vecteurs

Les vecteurs sont les objets les plus communs en R. Il s'agit d'une collection d'éléments à une dimension. Les éléments contenus dans un vecteur doivent nécessairement être **de même type**. Lorsque nous avons créé des objets contenant un numérique ou une chaîne de caractère, R a stocké l'information sous la forme d'un vecteur de longueur 1 (vecteur contenant un seul élément).

La fonction `c()` (pour *combine*) permet de créer un vecteur, on lui donne les éléments en argument de la fonction, en les séparant par une virgule (et éventuellement d'une espace).

Le tableau Table 2.1 reporte les valeurs du PIB nominal de la France, en fréquence annuelle.

Table 2.1 – PIB nominal annuel de la France, en millions d'Euros (série ajustée de la saisonnalité) (Source : Fred Economic Data).

Année	PIB
2015	549901,2
2016	558184,9
2017	574852,5
2018	591119,6
2019	610003,3
2020	577434,8
2021	624705,6

Utilisons la fonction `c()` pour créer un vecteur contenant les valeurs du PIB pour les années 2015 à 2019 :

```
pib_1 <- c(549901.2, 558184.9, 574852.5, 591119.6, 610003.3)
pib_1
```

```
## [1] 549901.2 558184.9 574852.5 591119.6 610003.3
```

L'objet `pib_1` que nous venons de créer est un vecteur de numériques de longueur 5 :

```
class(pib_1)
```

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

```
length(pib_1)
```

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

La fonction `c()` permet également de combiner plusieurs vecteurs :

```
pib_2 <- c(577434.8, 624705.6)
pib <- c(pib_1, pib_2)
pib
```

```
## [1] 549901.2 558184.9 574852.5 591119.6 610003.3 577434.8 624705.6
```

Pour **sélectionner** des sous-ensembles dans un vecteur, R propose l'indexation. Trois indexations différentes sont possibles : par position, par nom, et par condition.

2.2.1 Indexation par position

Chaque élément d'un vecteur possède une position, qui débute en R à 1. On utilise des crochets dans lesquels les positions des éléments à conserver sont données dans un vecteur. Pour extraire le premier élément de du vecteur de numériques `pib` :

```
pib[1]
```

```
## [1] 549901.2
```

Pour extraire le 1er et le 3e éléments :

```
pib[c(1,3)]
```

```
## [1] 549901.2 574852.5
```

Il est possible d'exclure les éléments dont les positions sont données dans le vecteur dans les crochets :

```
pib[-c(1,3)]
```

```
## [1] 558184.9 591119.6 610003.3 577434.8 624705.6
```

Exercice

1. Créez un vecteur que vous nommerez `annees` qui contiendra les valeurs des années du tableau Table 2.1.
2. En utilisant l'indexation par position, extrayez les années aux positions 2 et 4.
3. En utilisant l'indexation par position, récupérez le dernier élément du vecteur `annees`.
4. Si, à la question précédente, vous avez inscrit à la main la valeur de la position du dernier élément, recommencez la question en obtenant la position du dernier élément à l'aide d'une fonction (pour que votre instruction retourne le dernier élément du vecteur, peu importe sa taille).

2.2.2 Indexation par nom

Il est possible de nommer les éléments d'un vecteur lors de sa création :

```
notes <- c(maths = 10, francais = 12, anglais = 15)
notes
```

```
##    maths francais  anglais
##       10       12       15
```

Ou bien une fois que le vecteur a été créé, en utilisant la fonction `name()` :

```
notes_2 <- c(10, 12, 15)
names(notes_2) <- c("maths", "francais", "anglais")
notes_2
```

```
##    maths francais  anglais
##       10       12       15
```

La fonction `names()` permet d'obtenir la liste des noms (et dans l'intruction précédente, les noms ont été remplacés avec une nouvelle assignation) :

```
names(notes_2)
```

```
## [1] "maths"    "francais" "anglais"
```

Pour extraire des sous-éléments d'un vecteur à l'aide des noms, le fonctionnement est similaire à l'indexation par position :

```
notes_2["maths"]
```

```
## maths
##    10
```

```
notes_2[c("maths", "anglais")]
```

```
##   maths anglais
##      10      15
```

2.2.3 Indexation par condition

Plutôt que d'indiquer les positions ou les noms des éléments, il est possible de fournir un vecteur de logique précisant, pour chaque élément, si celui-ci doit être inclus (`TRUE`) ou exclus (`FALSE`) :

```
pib[c(TRUE, TRUE, FALSE, TRUE, FALSE, FALSE, TRUE)]
```

```
## [1] 549901.2 558184.9 591119.6 624705.6
```

Evidemment, écrire un vecteur de logique à la main est assez fastidieux. Imaginons que nous disposons d'un vecteur indiquant pour chaque année correspondant à nos observations de PIB s'il s'agit d'une année touchée par une crise économique :

```
crise <- c(FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, TRUE)
```

Ce vecteur peut être utilisé pour filtrer les valeurs du PIB lors des crises :

```
pib[crise]
```

```
## [1] 577434.8 624705.6
```

Les vecteurs de logiques peuvent de surcroit être obtenus en faisant appel à des opérateurs de comparaison reportés dans le tableau ci-dessous.

Table 2.2 – Opérateurs de comparaison

Opérateur	Description
`<`	inférieur à
`<=`	inféfieur ou égal à
`>`	supérieur à
`>=`	supérieur ou égal à
`==`	égal à
`!=`	différent de

Par exemple, pour obtenir un vecteur qui indique pour chaque élément contenu dans le vecteur numérique `pib` si la valeur est inférieure à 560 000 :

```
pib < 560000
```

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

Aussi, pour obtenir les éléments de `pib` dont les valeurs sont inférieures à 560 000, on peut écrire :

```
pib[pib < 560000]
```

```
## [1] 549901.2 558184.9
```

Exercice

1. Créez un vecteur de logiques à partir du vecteur `annees` créé lors du précédent exercice indiquant si chaque valeur du vecteur `annees` est égale à 2017.
2. Utilisez le vecteur de logiques de la question précédente pour extraire la valeur du vecteur `annees` correspondant à l'année 2017.
3. Utilisez le vecteur de logiques de la question précédente pour extraire la valeur du vecteur `pib` correspondant à l'année 2017.
4. En consultant un moteur de recherche, cherchez l'opérateur mathématique permettant de calculer le modulo d'un nombre.
5. En utilisant l'opérateur trouvé en première question, créez un vecteur logique permettant d'obtenir les années paires du vecteur `annees`.

Les vecteurs de logiques peuvent être le résultat de l'évaluation de plusieurs conditions logiques, en s'appuyant sur les opérateurs logiques.

Table 2.3 – Opérateurs logiques

Opérateur	Description
`&`	et logique
`|`	ou logique
`!`	négation logique

```
pib < 560000 | pib > 600000
```

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

```
pib[pib < 560000 | pib > 600000]
```

```
## [1] 549901.2 558184.9 610003.3 624705.6
```

Exercice

1. À l'aide d'une condition logique multiple, extrayez les valeurs du vecteur `pib` pour lesquelles les deux conditions suivantes sont remplies : l'année est inférieure ou égale à 2018, et la valeur est inférieure à 560 000.

2. À quelles années les valeurs extraites correspondent-elles ?

Notes :
Il existe en R un opérateur permettant de tester l'appartenance à un ensemble discret de valeurs : `%in%`.

```
x <- c("A", "B", "C", "D")
x %in% c("D", "B")
```

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

```
x[x %in% c("D", "B")]
```

```
## [1] "B" "D"
```

Par ailleurs, R propose deux fonctions très pratiques pour les vecteurs de logiques :
- `any()` : au moins un des éléments du vecteur vaut `TRUE` ;
- `all()` : tous les éléments du vecteur valent `TRUE`.

```
x <- c(TRUE, FALSE)
any(x)
```

```
## [1] TRUE
```

```
y <- c(FALSE, FALSE)
all(!y)
```

```
## [1] TRUE
```

2.2.4 Gestion des valeurs manquantes

Admettons que nous disposions d'une valeur manquante pour une des années.

```
pib <- c(549901.2, 558184.9, NA, 591119.6,
         610003.3, 577434.8, 624705.6)
pib
```

```
## [1] 549901.2 558184.9       NA 591119.6 610003.3 577434.8 624705.6
```

Lorsque l'on fait un test logique sur une valeur `NA`, le résultat est `NA` :

```
pib > 550000
```

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

Les valeurs `NA` d'un vecteur de logiques utilisé dans le cadre d'une indexation par condition retournent `NA`

```
pib[pib > 550000]
```

```
## [1] 558184.9       NA 591119.6 610003.3 577434.8 624705.6
```

Ce qui peut être pénible… Si on souhaite obtenir les années pour lesquelles le PIB est supérieur à 550 000 :

```
annees <- 2015:2022
annees
```

```
## [1] 2015 2016 2017 2018 2019 2020 2021 2022
```

```
annees[pib > 550000]
```

```
## [1] 2016   NA 2018 2019 2020 2021
```

On note que l'année pour laquelle la valeur du PIB est manquante a été transformée en `NA`.

Si on souhaite exclure les valeurs manquantes, on peut utiliser la fonction `is.na()` à laquelle on applique l'opérateur de négation logique :

```
annees[pib > 550000 & !is.na(pib)]
```

```
## [1] 2016 2018 2019 2020 2021
```

2.2.5 Remplacement et ajout de valeurs

Pour effectuer un remplacement de valeurs à l'intérieur d'un vecteur, on peut utiliser la fonction crochet (`"["()` que nous utilisons dans sa version plus pratique `[]`) et la flèche d'affectation (`<-`).

```
pib[3] <- 574852.5
```

Plusieurs éléments peuvent être modifiés en une seule instruction.

```
pib[pib < 560000 | pib > 600000] <- -1000
pib
```

```
## [1]  -1000.0  -1000.0 574852.5 591119.6  -1000.0 577434.8  -1000.0
```

Nous avons vu précédemment que pour ajouter des valeurs, il est possible de combiner plusieurs vecteurs. Il est aussi possible d'utiliser l'indexation.

```
pib[8] <- 10
pib
```

```
## [1]  -1000.0  -1000.0 574852.5 591119.6  -1000.0 577434.8  -1000.0
       10.0
```

2.2.6 Exercice de synthèse

Reprenons les données de PIB nominal de la France, en fréquence annuelle :

```
pib <- c(549901.2, 558184.9, 574852.5, 591119.6, 610003.3, 577434.8, 624705.6)
annees <- 2015:2021
```

1. En utilisant la fonction appropriée, retournez le nombre d'éléments contenus dans le vecteur numérique `pib`.
2. Donnez la valeur du PIB en milliards plutôt qu'en millions, et placez le resultat dans un vecteur que vous nommerez `pib_milliards`.
3. Quelles sont les années pour lesquelles le PIB est compris entre 560 000 et 590 000 (pensez à effectuer un test logique contenant deux conditions logiques et un opérateur logique).
4. Créez un indice pour le PIB, en prenant comme référence l'année 2020 (valeur 100 en 2020). Stockez le résultat dans un objet que vous nommerez `pib_ind_2020`. Pour ce faire :
   — créez un objet que vous nommerez `ref` qui contiendra la valeur du PIB en 2020 (utilisez une extraction par condition)
   — divisez les valeurs de `pib` par la valeur stockée dans `ref`, puis multipliez le résultat par 100.

5. Regardez le fonctionnement de la fonction `lag()` dans le fichier d'aide et en observant le résultat dans la console lorsque vous l'appliquez au vecteur `pib`.
6. En utilisant la fonction `lag()`, calculez le taux de croissance, en pourcentage du PIB d'une année à l'autre.
7. Admettons que la réserve Fédérale de Saint Louis vous informe que la valeur du PIB en 2016 est finalement corrigée et est maintenant estimée à 55963.2. Corrigez la valeur à l'aide d'une affectation de remplacement.

Chapitre 3

Matrices et listes

Ce chapitre s'appuie sur les sections 2.2.1.4 Matrices et 2.2.1.5 Listes du chapitre 2 des notes de cours, ainsi que sur la Section 2.4 Manipulation des données.

3.1 Matrice

En R, les matrices sont des vecteurs auxquels un attridut `dim` de dimension est ajouté (cet attribut est de longueur 2). La fonction `matrix()` permet de créer une matrice. Son premier argument, `data` reçoit un vecteur de valeurs. Les arguments `ncol` et `nrow` reçoivent quant à eux le nombre de colonnes et de lignes, respectivement. Il est possible de ne préciser qu'un seul de ces deux derniers arguments (en fonction de la longueur du vecteur donné à l'argument `data`, R devine).

Créons la matrice suivante en R :

$$A = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}$$

```
A <- matrix(data = c(1,2,3,4,5,6, 7, 8, 9, 10, 11, 12), ncol = 4)
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
```

```
## [3,]    3    6    9   12
```

Notes

- Le remplissage de la matrice s'est effectué colonne par colonne, de sorte à avoir 4 colonnes dans le résultat final (`ncol=4{.R}`).
- Les données fournies à l'argument `data` doivent être de même type (puisqu'elles sont contenues dans un vecteur).
- Nous pouvons ne pas écrire `data=` et simplement écrire `matrix(c(1,2,3,4,5,6,7,8,9,10,11,12), ncol = 4)`.
- Au lieu d'écrire `c(1,2,3,4,5,6,7,8,9,10,11,12)`, il est possible de générer la même séquence de nombres à l'aide d'une syntaxe plus courte : `seq(1,12)` ou encore plus court : `1:12`.

Pour remplir la matrice en procédant ligne par ligne plutôt que colonne par colonne, on peut ajouter `byrow = TRUE` en argument à la fonction `matrix()`

```
matrix(1:12, ncol = 4, byrow = TRUE)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

3.1.1 Accès aux éléments d'une matrice

3.1.1.1 Indexation par position

Avec les vecteurs, nous avons extrait des sous-ensembles à l'aide de la fonction crochet, en utilisant l'indexation. Les vecteurs étant des objets en dimension 1, nous n'avons donné à la fonction crochet qu'un seul argument : `x[c(2,3)]` permet d'extraire les éléments aux positions 2 et 3 de `x`. Les matrices étant des objets à deux dimensions, il convient de fournir deux arguments à la fonction crochet, en les séparant par une virgule : `x[i,j]`. Le premier argument, `i` correspond aux indices de la première dimension (les lignes) et `j` aux indices de la deuxième dimension (les colonnes).

Revenons à notre matrice `A`. Extrayons les observations des lignes 1 et 3 aux colonnes 2 et 4 :

```
A[c(1,3), c(2,4)]
```

```
##      [,1] [,2]
```

```
## [1,]    4   10
## [2,]    6   12
```

En ommettant l'argument `i` ou l'argument `j`, l'ensemble des lignes ou des colonnes respectivement est retourné.

Les éléments de l'ensemble des lignes aux colonnes 2 et 4 :

```
A[, c(2,4)]
```

```
##      [,1] [,2]
## [1,]    4   10
## [2,]    5   11
## [3,]    6   12
```

Les éléments des lignes 1 et 3 pour l'ensemble des colonnes :

```
A[c(1,3), ]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    3    6    9   12
```

Pour exclure des éléments, comme pour les vecteurs, on utilise le symbole moins (`-`). Par exemple, pour extraire les éléments pour toutes les lignes, pour toutes les colonnes sauf la 2e et la 4e :

```
A[, -c(2,4)]
```

```
##      [,1] [,2]
## [1,]    1    7
## [2,]    2    8
## [3,]    3    9
```

Note
Rappelez-vous que les matrices sont stockés sous forme de vecteurs. Il est donc possible d'utiliser la fonction crochet en lui fournissant uniquement l'argument `i`. L'instruction suivante retourne, sous la forme d'un vecteur, les éléments aux positions 1, 4, et 3 :

```r
A[c(1,4,3)]
```

```
## [1] 1 4 3
```

Exercice

1. Créez la matrice suivante :

$$B = \begin{bmatrix} -2 & 2 & -3 \\ -1 & 1 & 3 \\ 4 & 0 & -1 \end{bmatrix}.$$

2. Extrayez la 2e colonne de la matrice B.
3. Extrayez la sous-matrice dans le coin inférieur de B :

$$C = \begin{bmatrix} 1 & 3 \\ 0 & -1 \end{bmatrix}.$$

4. Extrayez toutes les colonnes de B sauf la 2e.
5. Extrayez le terme en dernière ligne et en dernière colonne de B.

3.1.1.2 Indexation par condition

Comme pour les vecteur, nous pouvons utiliser une matrice de logique indiquant si les éléments doivent être retournés (`TRUE`) ou non(`FALSE`).

```r
masque <- matrix(c(TRUE, FALSE, FALSE, FALSE, TRUE, FALSE,
                   TRUE, TRUE, FALSE, FALSE, FALSE, FALSE), ncol = 4)
masque
```

```
##       [,1]  [,2]  [,3]  [,4]
## [1,]  TRUE FALSE  TRUE FALSE
## [2,] FALSE  TRUE  TRUE FALSE
## [3,] FALSE FALSE FALSE FALSE
```

En appliquant ce masque de valeurs logiques à la matrice A, on récpuère dans un vecteur les valeurs pour lesquelles la valeur dans le masque à la même position vaut `TRUE` :

```r
A[masque]
```

```
## [1] 1 5 7 8
```

Evidemment, écrire de tels masques à la main n'est pas très utile. En revanche, ils peuvent être obtenus suite à l'évaluation d'une condition logique. Par exemple, si on souhaite obtenir les éléments de la matrice A dont la valeur est inférieure à 5, on peut créer le masque suivant :

```
A<5
```

```
##       [,1]  [,2]  [,3]  [,4]
## [1,] TRUE  TRUE FALSE FALSE
## [2,] TRUE FALSE FALSE FALSE
## [3,] TRUE FALSE FALSE FALSE
```

Et ensuite appliquer ce masque avec la fonction crochet :

```
A[A<5]
```

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

Exercice

1. Récupérez les valeurs de la matrice A qui sont inférieures à 3 ou supérieures à 7.
2. Récupérez les valeurs paires de la matrice A (utilisez le modulo).

3.1.2 Remplacement et ajout de valeurs

Comme pour les vecteurs, le remplacement de valeurs peut s'effectuer avec la flèche d'affectation `<-`. On écrit à gauche de cette flèche l'extraction réalisée par indexation et à droite les valeurs de remplacement.

Reprenons notre matrice A :

$$A = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}$$

```
A <- matrix(1:12, ncol = 4)
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

Remplaçons les valeurs 10 et 11 par -10 et -11

```
A[c(1,2),4] <- c(-10, -11)
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7  -10
## [2,]    2    5    8  -11
## [3,]    3    6    9   12
```

Cela fonctionne aussi avec un indexation par condition. Créons un masque indiquant si les valeurs sont négatives :

```
A<0
```

```
##       [,1]  [,2]  [,3]  [,4]
## [1,] FALSE FALSE FALSE  TRUE
## [2,] FALSE FALSE FALSE  TRUE
## [3,] FALSE FALSE FALSE FALSE
```

Puis affectons la valeur `NA` à la place des valeurs négatives de la matrice `A` :

```
A[A<0] <- NA
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   NA
## [2,]    2    5    8   NA
## [3,]    3    6    9   12
```

Exercice

1. Créez la matrice suivante :

$$X = \begin{bmatrix} -2 & 2 & -3 \\ -1 & 1 & 3 \\ 4 & 0 & -1 \end{bmatrix}.$$

2. Remplacez les valeurs positives de `X` par 1000.

3. Remplacez les éléments aux positions (1,1) et (2,3) par `NA`.

3.1.3 Calculs matriciels

En économie, et particulièrement en économétrie, nous manipulons beaucoup de matrices. Il est important de savoir effectuer avec R les différents calculs matriciels les plus courants dans notre discipline.

Considérons quelques matrices :

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 4 & 3 & -1 \end{bmatrix}$$

$$B = \begin{bmatrix} 5 & 1 \\ 2 & 3 \\ 3 & 4 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} -2 & 2 & -3 \\ -1 & 1 & 3 \\ 4 & 0 & -1 \end{bmatrix}$$

```
A <- matrix(c(1,4,2,3,0,-1), nrow = 2)
B <- matrix(c(5,2,3, 1,3,4), ncol = 2)
C <- matrix(c(1,0,0,0,1,0,0,0,1), ncol = 3)
D <- matrix(c(-2,-1,4,2,1,0,-3,3,-1), ncol = 3)
```

Considérons par ailleurs le scalaire $a = 3$.

```
a <- 3
```

3.1.3.1 Addition, soustraction

Pour additionner ou soustraire un scalaire ou une matrice à une matrice, on utilise simplement les opérateurs d'addition (`+`) ou de soustraction (`-`), respectivement :

```
A+a
```

```
##      [,1] [,2] [,3]
## [1,]    4    5    3
## [2,]    7    6    2
```

```
A-a
```

```
##      [,1] [,2] [,3]
## [1,]   -2   -1   -3
## [2,]    1    0   -4
```

```
C+D
```

```
##      [,1] [,2] [,3]
## [1,]   -1    2   -3
## [2,]   -1    2    3
## [3,]    4    0    0
```

```
C-D
```

```
##      [,1] [,2] [,3]
## [1,]    3   -2    3
## [2,]    1    0   -3
## [3,]   -4    0    2
```

Lorsqu'on essaie d'additionner ou de soustraire deux matrices non compatibles, R retourne une erreur :

```
A+C
```

```
## Error in A + C: tableaux de tailles inadéquates
```

3.1.3.2 Multiplication

La multiplication par un scalaire s'effectue avec l'étoile `*` :

```
A*a
```

```
##      [,1] [,2] [,3]
## [1,]    3    6    0
```

```
## [2,]   12    9   -3
```

Pour multiplier deux matrices (compatibles) entre-elles, on utilise l'opérateur de produit matriciel `%*%` :

```
A %*% B
```

```
##      [,1] [,2]
## [1,]    9    7
## [2,]   23    9
```

En cas d'incompatibilité, R nous prévient :

```
C %*% A
```

```
## Error in C %*% A: arguments inadéquats
```

Attention, si on utilise l'étoile au lieu de l'opérateur de produit matriciel, si les deux matrices ont la même dimension, alors R effectue une multiplication terme à terme :

```
C * D
```

```
##      [,1] [,2] [,3]
## [1,]   -2    0    0
## [2,]    0    1    0
## [3,]    0    0   -1
```

3.1.3.3 Transposée

La transposée d'une matrice s'obtient en R avec la fonction `t()` :

```
t(A)
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    3
## [3,]    0   -1
```

3.1.3.4 Inversion

L'inversion d'une matrice est calculée à l'aide de la fonction `solve()` :

```
solve(D)
```

```
##              [,1]       [,2] [,3]
## [1,] -0.02777778 0.05555556 0.25
## [2,]  0.30555556 0.38888889 0.25
## [3,] -0.11111111 0.22222222 0.00
```

En économétrie, l'estimateur des moindres carrés ordinaires est donné par $(X^\top X)^{-1}(X^\top y)$, où X est la matrice des variables explicatives et y le vecteur de la variable endogène.

```
solve(t(X) %*% X) %*% (t(X) %*% y)
```

Qui peut aussi se calculer de manière équivalente (nous le ferons dans un prochain exercice) :

```
crossprod(X,X) %*% crossprod(X,y)
```

3.1.3.5 Déterminant

Le déterminant s'obtient avec la fonction `det()` :

```
det(D)
```

```
## [1] 36
```

3.1.3.6 Diagonale

La diagonale d'une matrice peut être obtenue avec la fonction `det()`{.R :

```
diag(C)
```

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

Note
La fonction `diag()` permet par ailleurs de créer une matrice identité :

```
diag(3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

Pour obtenir la trace d'une matrice carrée, on peut simplement sommer les éléments de la diagonale :

```
sum(diag(D))
```

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

3.1.4 Combiner des matrices

Pour ajouter des **colonnes** à une matrice, la fonction `cbind()` peut être utilisée. Il s'agit de combiner (*bind*) les colonnes (*columns*) de deux matrices ou d'une matrice et d'un vecteur colonne.

```
cbind(B, C)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    5    1    1    0    0
## [2,]    2    3    0    1    0
## [3,]    3    4    0    0    1
```

```
cbind(B, c(10,11,12))
```

```
##      [,1] [,2] [,3]
## [1,]    5    1   10
## [2,]    2    3   11
## [3,]    3    4   12
```

Pour ajouter des **lignes** à une matrice, la fonction `rbind()` peut être utilisée. Il s'agit de combiner (*bind*) les lignes (*rows*) de deux matrices ou d'une matrice et d'un vecteur ligne.

```
rbind(C, D)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
## [4,]   -2    2   -3
## [5,]   -1    1    3
## [6,]    4    0   -1
```

```
rbind(C, c(10,11,12))
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
## [4,]   10   11   12
```

Note
Si on ajoute un vecteur colonne (ligne) à une matrice, et que la taille du vecteur est inférieure au nombre de colonnes (lignes), R effectue ce que l'on appelle un **recyclage**. C'est-à-dire que R va compléter le vecteur plus court à l'aide des valeurs de ce même vecteur, en repartant au début de celui-ci, afin d'obtenir deux objets de même taille.

```
cbind(B, c(10, 11))
```

```
##      [,1] [,2] [,3]
## [1,]    5    1   10
## [2,]    2    3   11
## [3,]    3    4   10
```

Exercice
Dans cet exercice, nous allons considérer un modèle de régression linéaire. Vous allez utiliser vos connaissances acquises sur les matrices pour estimer les coefficients du modèle par la méthode des moindres carrés ordinaires.
Considérons le modèle suivant, pour les observations $i = 1, \dots, n$:

$$\text{consommation}_i = \beta_0 + \beta_1 \text{puissance}_i + \beta_2 \text{masse}_i + \varepsilon_i,$$

où consommation est un vecteur de valeurs de consommation de carburant (miles par gallons), puissance et masse des vecteurs de puissance (nombre de chevaux) et

de masse du véhicule (en millième de livres) respectivement. Le vecteur ε est un terme d'erreurs que l'on suppose distribué selon une normale d'espérance nulle.
L'objectif de l'exercice est d'estimer les paramètres β_0, β_1 et β_2.
Dans un premier temps, évaluez les instructions suivantes pour récupérer les valeurs (qui sont stockées dans un tableau de données nommé `mtcars` qui est fourni dans le *package* {datasets}).

```
consommation <- mtcars$mpg
puissance <- mtcars$hp
masse <- mtcars$wt
```

Créez également un vecteur pour la constante (ce vecteur ne contient que des 1) :

```
constante <- rep(1, nrow(mtcars))
```

2. Créez la matrice `X` qui doit contenir 3 colonnes : une pour la constante (`constante`), une autre pour la puissance (`puissance`) et une dernière pour la masse (`masse`) ;
3. Calculez la transposée de X : $X^\top$;
4. Calculez le produit matriciel : $X^\top X$;
5. Calculez l'inverse de ce produit matriciel : $(X^\top X)^{-1}$;
6. Calculez le produit matriciel $X^\top y$;
7. Donnez les estimations des paramètres : $(X^\top X)^{-1} X^\top y$.

3.2 Listes

Tandis que les éléments d'un vecteur ou d'une matrice doivent nécessairement être de même type, les listes proposent une structure de données moins rigide à cet égard. Les types des élément d'une liste en R peuvent être différents.

```
l_1 <- list(c(10,12,15), c("Janvier", "Fevrier"))
l_1
```

```
## [[1]]
## [1] 10 12 15
##
## [[2]]
## [1] "Janvier" "Fevrier"
```

Les classes d'objets à l'intérieur d'une liste ne sont pas non plus nécessairement identiques.

Dans l'exemple qui suit, la liste `l_2` contient un vecteur de numériques en premier élément, et une matrice de numériques en second :

```
l_2 <- list(c(1,2,3), matrix(c(1,2,3,4), ncol = 2))
l_2
```

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

Les éléments peuvent être, comme dans un vecteur, nommés. Les noms peuvent être indiqués au moment de la création…

```
l_1 <- list(notes = c(10,12,5), mois = c("Janvier", "Fevrier"))
l_1
```

```
## $notes
## [1] 10 12  5
##
## $mois
## [1] "Janvier" "Fevrier"
```

… ou une fois la liste créée, à l'aide de la fonction `names()` :

```
l_1 <- list(c(10,12,5),  c("Janvier", "Fevrier"))
names(l_1) <- c("notes", "mois")
l_1
```

```
## $notes
## [1] 10 12  5
##
## $mois
## [1] "Janvier" "Fevrier"
```

3.2.1 Accès aux éléments

Comme pour les vecteurs, il est possible d'utiliser les crochets pour extraire des éléments d'une liste. **La fonction crochet appliquée à une liste retourne une liste**. On donne

en arguments de cette fonction, comme pour les vecteurs, soit des positions, soit des noms, soit des valeurs logiques.

Créons une liste contenant 4 éléments :

```
l_3 <- list(nombres = 1:5, alphabet_min = letters,
            alphabet_maj = LETTERS, mois = month.name)
l_3
```

```
## $nombres
## [1] 1 2 3 4 5
##
## $alphabet_min
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p"
    "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
##
## $alphabet_maj
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P"
    "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
##
## $mois
##  [1] "January"   "February"  "March"     "April"     "May"       "
   June"
##  [7] "July"      "August"    "September" "October"   "November"  "
   December"
```

Note
Il existe 5 constantes intégrées en R :
- `LETTERS` : les 26 lettres de l'alphabet occidental, en majuscules ;
- `letters` : les 26 lettres de l'alphabet occidental, en minuscules ;
- `month` : les noms des 12 mois de l'année du calendrier grégorien ;
- `month.abb` : les abbréviations en 3 lettres des 12 mois du calendrier grégorien ;
- `pi` : le ratio de la circonférence d'un cercle sur son diamiètre (pi).

L'indexation par position fonctionne comme pour un vecteur :

```
l_3[1]
```

```
## $nombres
## [1] 1 2 3 4 5
```

```
l_3[c(1,3)]
```

```
## $nombres
## [1] 1 2 3 4 5
##
## $alphabet_maj
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P"
     "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
l_3[-c(2,3)]
```

```
## $nombres
## [1] 1 2 3 4 5
##
## $mois
##  [1] "January"   "February"  "March"     "April"     "May"       "
    June"
##  [7] "July"      "August"    "September" "October"   "November"  "
    December"
```

L'indexation par nom fonctionne aussi comme pour les vecteurs :

```
l_3[c("mois", "nombres")]
```

```
## $mois
##  [1] "January"   "February"  "March"     "April"     "May"       "
    June"
##  [7] "July"      "August"    "September" "October"   "November"  "
    December"
##
## $nombres
## [1] 1 2 3 4 5
```

Il en est de même pour l'indexation par condition :

```
l_3[c(TRUE, FALSE, FALSE, TRUE)]
```

```
## $nombres
## [1] 1 2 3 4 5
##
## $mois
```

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

Note
Nous verrons lors de la séance de travaux dirigés sur les boucles comment parcourir les éléments d'une liste tout en appliquant un test logique à chaque élément.

On note bien que le résultat retourné est une liste :

```
class(l_3[1])
```

```
## [1] "list"
```

```
class(l_3[c(1,3)])
```

```
## [1] "list"
```

```
class(l_3["nombres"])
```

```
## [1] "list"
```

Si l'on souhaite accéder au **contenu** d'un élément particulier, il faut utiliser la fonction double crochets (`"[["()`). De manière analogue à la fonction crochets (`"["()`), une syntaxe allégée est proposée.

```
l_3[[1]]
```

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

Le résultat retourné est bien le contenu, ici un vecteur d'entiers :

```
class(l_3[[1]])
```

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

La fonction `str()`, qui affiche la structure d'un objet de manière compacte, permet de mieux se fixer les idées :

```
str(l_3[1])
```

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

```
str(l_3[[1]])
```

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

Note
Lordque les éléments de la liste son nommés et que l'on désire extraire le contenu d'un seul élément, il est possible d'utiliser le dollar en lieu et place des crochets, en indiquant le nom de l'élément.

```
l_3$nombres
```

Créons une liste comprenant trois éléments, dont le 3e est une liste de deux éléments :

```
l_4 <-
  list(c(1,2,3),
     c("Bonjour", "Hello"),
     list(c(1,2,3), month.name))
l_4
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "Bonjour" "Hello"
##
## [[3]]
## [[3]][[1]]
## [1] 1 2 3
##
## [[3]][[2]]
##  [1] "January"   "February"  "March"     "April"     "May"       "
    June"
##  [7] "July"      "August"    "September" "October"   "November"  "
    December"
```

Si on souhaite extraire le premier élément du troisième élément de `l_4`, on utilise dans un premier temps les doubles crochets (pour extraire le contenu du 3e élément de `l_4`), puis on utilise la fonction crochet sur le résultat (pour extraire le premier élément). Il s'agit d'une composition de fonctions, et la fonction appliquée en dernier étant la fonction crochet, le résultat est… une liste !

```
l_4[[3]][1]
```

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

Si l'on souhaite extraire le **contenu** du 1er élément du 3e élément de `l_4`, on utilisera deux fois la fonction double crochets : la première pour extraire le contenu du 3e élément (on récupère l'objet en 3e position, qui est une liste) puis la seconde pour extraire le contenu du premier élément de ce que l'on vient d'extraire :

```
l_4[[3]][[1]]
```

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

3.2.2 Remplacement et ajout d'éléments

Pour remplacer un élément dans une liste, comme pour un vecteur ou une matrice, on utilise une nouvelle affectation.

```
l_3[["nombres"]] <- c(6,5,4,3,2,1,0)
l_3$mois <- c("January", "February")
l_3
```

```
## $nombres
## [1] 6 5 4 3 2 1 0
##
## $alphabet_min
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p"
    "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
##
## $alphabet_maj
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P"
    "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
##
```

```
## $mois
## [1] "January"  "February"
```

Pour ajouter un élément, on effectue simplement une affectation. Soit on nomme le nouvel élément :

```
l_3["autres_nombres"] <- c(1,2,3)
l_3
```

```
## $nombres
## [1] 6 5 4 3 2 1 0
##
## $alphabet_min
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p"
     "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
##
## $alphabet_maj
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P"
     "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
##
## $mois
## [1] "January"  "February"
##
## $autres_nombres
## [1] 1
```

Si l'élément n'as pas vocation à être nommé, on utilise la fonction `c()` et on place le nouvel élément au sein d'une liste :

```
l_3 <- c(l_3, list(c(4,5,6)))
l_3
```

```
## $nombres
## [1] 6 5 4 3 2 1 0
##
## $alphabet_min
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p"
     "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
##
## $alphabet_maj
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P"
    "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
##
## $mois
## [1] "January"  "February"
##
## $autres_nombres
## [1] 1
##
## [[6]]
## [1] 4 5 6
```

Exercice

1. Extrayez le 3e élément de la liste suivante :

```
ma_liste <- list(rnorm(10), LETTERS[1:26], month.abb)
```

2. Extrayez tous les éléments de la liste `ma_liste` sauf le second.
3. Extrayez le troisième élément de la liste `ma_liste`, en utilisant le nom de l'élément.

```
ma_liste <- list(nombre = rnorm(10),
                 lettre = LETTERS[1:26], mois = month.abb)
```

4. Toujours en accédant par le nom, ajouter la valeur 10 aux valeurs du premier élément de `ma_liste`. Le résultat doit altérer `ma_liste`.

Chapitre 4

Importation et exportation de donnée

Ce chapitre s'appuie sur la Section 2.3 Importation, exportation et création de données des notes de cours.

4.1 Chargement de données : chemins

Lorsque l'on souhaite accéder à des données avec Excel, il suffit de double cliquer sur le fichier dans l'explorateur de fichiers pour que le logiciel se lance et que le tableur nous révèle son contenu. Avec R, il est nécessaire de payer un petit coût d'entrée pour pouvoir charger les données. Mais une fois ce coût payé, il nous est possible de réaliser des opérations avancées en fournissant des efforts moindres. Par ailleurs, le traitement des gros volumes de données est facilité avec R, comparativement à Excel.

Dans cet exercice, vous allez apprendre à importer des données dans R. Cela nécessite au préalable d'avoir connaissance de l'environnement de travail (en anglais, *working directory*). Lorsqu'on lance RStudio en double cliquant sur un fichier de projet (dont l'extension est `.Rproj`), comme ce que l'on a fait jusqu'à présent, l'environnement de travail de R est le répertoire dans lequel se situe le fichier `.Rproj`. Aussi, lorsque nous donnons comme instruction à R d'importer ou d'exporter des informations, R se réfère à ce répertoire comme point d'ancrage, comme répertoire courant.

1. En choisissant l'option "Enregistrer le fichier" (**et non pas en choisissant "Ouvrir avec"), téléchargez l'archive `exercice_importation.zip` à

l'adresse suivante : https://egallic.fr/Enseignement/R/Exercices/exercice_importation.zip.

Par défaut, sur les ordinateurs de l'Université, les fichiers téléchargés sont enregistrés dans le dossier `Téléchargement`. Pour être plus précis, les fichiers téléchargés sont enregistrés sur le disque dur nommé `C:`, dans un répertoire nommé `Downloads` qui se situe lui-même dans un répertoire nommé selon votre numéro d'étudiant, situé lui-même dans un répertoire nommé `Users`. Pour accéder à une ressource, le système d'exploitation utilise des chemins d'accès, qui sont constitués à l'aide des noms de répertoires séparés par des barres obliques inverse sous Windows (\) (et des barres obliques `/` sous Unix) et qui se terminent soit par un nom de répertoire, soit par un nom de fichier. Le chemin qui permet de localiser le répertoire de téléchargement sur la machine de l'Université s'écrit ainsi : `C:\Users\votre_numero_etudiant\Downloads`, tandis que le chemin qui permet de localiser le fichier que vous venez de télécharger s'écrit `C:\Users\votre_numero_etudiant\Downloads\exercice_importation.zip`.

2. Ouvrez l'explorateur de fichiers et déplacez vous dans le dossier de téléchargement.
3. Faites un clic droit sur le fichier `exercice_importation.zip`, et dans le menu contextuel qui s'affiche, choisissez l'option `7-Zip`, puis `Extraire vers exercice_importation`. Un répertoire nommé `exercice_importation` a alors été extrait.

Note
Dans votre répetoire `Téléchargement` vous pouvez à présent voir deux ressources portant le nom `exercice_importation`. Si vous faites bien attention, l'icône de ces ressources diffère : l'une représente un porte document jaune ouvert (et désigne le répertoire que vous venez d'extraire de l'archive), tandis que l'autre présente un porte document jaune fermé et muni d'une fermeture éclair (un *zip*).
Afin d'éviter de confondre les deux, un moyen simple consiste à afficher l'extension des fichiers. Il sera alors aisé de reconnaître les fichiers des répertoires : les dossiers n'auront pas d'extention. Pour afficher les extensions :

1. Dans l'explorateur de fichiers, atteignez l'onglet `Affichage` ;
2. Cliquez sur le bouton `Afficher/Masquer` ;
3. Cochez la case `Extensions de noms de fichiers`.

À présent, vous pourrez bien voir la différence entre le répertoire `exercice_importation` et le fichier ZIP `exercice_importation.zip` (l'archive).

La hiérarchie du répertoire `exercice_importation` que vous venez d'extraire peut être représentée à l'aide d'un arbre (on parle d'arborescence).

```
exercice_importation
    Dossier_C
        fichier_donnees_C.csv
    Dossier_D
        Dossier_D_1
|   |       fichier_donnees_D_1.csv
    Exercice
|       Dossier_A
|   |       Dossier_A_1
|   |   |       fichier_donnees_A_1.csv
|   |       Dossier_A_2
|   |   |       fichier_donnees_A_2.csv
|   |       fichier_donnees_A.csv
|       Dossier_B
|   |       fichier_donnees_B.csv
|       fichier_donnees.csv
|       L3_info.Rproj
```

Mise en garde
Faites attention lorsque vous ouvrez des fichiers CSV avec Excel. Dès lors que vous enregistrerez des modifications, Excel ajoutera des informations dans le fichier. Ces ajouts peuvent rendre vos fichiers illisibles par d'autres logiciels ensuite.

4. Depuis l'explorateur de fichiers, descendez dans le répertoire `exercice_importation`, puis dans `Exercice` et ouvrez le fichier `L3_info.Rproj` en double cliquant dessus.

En ayant ouvert le fichier `L3_info.Rproj`, RStudio s'est lancé, en définissant le répertoire `Exercice` comme répertoire de travail.

5. Dans la console R de RStudio, évaluez l'instruction suivante : `getwd()` et observez le chemin retourné.
6. Créez un nouveau script R.

Le répertoire courant est donc `Exercice`. Lorsque nous allons charger ou exporter dans ou depuis R des données, nous pourrons créer des chemins relatifs à ce répertoire courant. Il convient ainsi de noter la distinction entre chemin absolu et chemin relatif :

1. **Chemin absolu** : le chemin absolu décrit l'accès à une ressource sans se référer au chemin courant, mais en se rapportant à la place à la racine du disque ou du serveur.

— exemple : le chemin absolu vers le fichier `fichier_donnees.csv` sur **windows** sera : `C:\Users\votre_numero_etudiant\Downloads\exercice_importation\Exercice\fichier_don`
— Note : sous Unix, la racine du système de fichiers se note simplement `/`. Le lien absolu vers le fichier ressemblera à celui-ci : `/home/votre_nom_user/Downloads/fichier_donnees.csv`.

2. **Chemin relatif** : le chemin relatif décrit quant à lui l'accès à une ressource en se rapportant au répertoire courant.

— le répertoire courant étant `Exercice`, pour accéder au fichier csv, le chemin est le suivant, sous **windows** : `fichier_donnees.csv` (on peut aussi écrire `.\fichier_donnees.csv`, où `.` se réfère au **répertoire courant**)
— sous Unix : `fichier_donnees.csv` ou `./fichier_donnees.csv`.

Note
La barre oblique `/` (inversée `\` sous Windows) est le séparateur de chemin.

Nous allons à présent importer dans R le contenu du fichier `fichier_donnees.csv`. Ce fichier contient une première ligne définissant les en-têtes, et une seconde ligne contenant une observation. Dans le fichier, les différents champs sont séparés par un point-virgule (`;`).

Pour importer le contenu de ce fichier CSV dans R, nous allons nous appuyer sur la fonction `read_csv2()` du *package* {readr}. Nous allons indiquer le chemin vers le fichier à l'argument `file` de cette fonction. Le tableau de données sera stocké dans un objet que nous allons appeler `df_exercice` :

```
df_exercice <- read_csv2(file = "fichier_donnees.csv")
df_exercice
```

```
## # A tibble: 1 x 2
##   Dossier          Valeur
##   <chr>            <chr>
## 1 Dossier_Exercice Bravo
```

L'importation a bien fonctionné, R nous prévient avoir deviné que les valeurs dans les colonnes `Dossier` et `Valeur` sont des chaînes de caractères (la fonction `col_character()` a été utilisée pour ces deux colonnes).

Nous pouvons obtenir le même résultat en écrivant le chemin absolu vers le fichier :

```
path <- paste0("C:/Users/votre_numero_etudiant/Downloads/",
               "exercice_importation/Exercice/fichier_donnees.csv")
read_csv2(file = path)
```

Note : le chemin étant trop long pour tenir sur une page PDF, la fonction `paste0` a été utilisée ici pour concaténer deux chaînes de caractères en une seule. Sur votre machine, vous pouvez écrire le chemin complet sans passer par l'utilisation de la fonction `paste0`.

Nous verrons plus loin que l'utilisation des chemins absolus dans les scripts posent des soucis lorsque l'on change d'ordinateur ou que l'on partage son code avec d'autres personnes.

Le fichier `fichier_donnees_A.csv` n'est pas situé dans le répertoire courant : il se situe dans le répertoire `Dossier_A`, contenu dans le répertoire courant (`Exercice`). Pour pouvoir charger le contenu de ce fichier dans R avec un lien relatif, il faut constuire le chemin en indiquant que l'on souhaite descendre dans le répertoire `Dossier_A` dans un premier temps :

```
df_exercice_A <- read_csv2(file = "Dossier_A/fichier_donnees_A.csv")
df_exercice_A
```

```
## # A tibble: 1 x 2
##   Dossier   Valeur
##   <chr>     <chr>
## 1 Dossier_A Bravo
```

7. Dans un objet que vous nommerez `df_exercice_A_1`, chargez dans R le contenu du fichier `fichier_donnees_A_1.csv`.
8. Dans un objet que vous nommerez `df_exercice_A_2`, chargez dans R le contenu du fichier `fichier_donnees_A_2.csv`.
9. Dans un objet que vous nommerez `df_exercice_B`, chargez dans R le contenu du fichier `fichier_donnees_B.csv`.

Si la ressource que l'on souhaite atteindre nécessite de remonter dans un répertoire parent (un répertoire qui contient un autre est appelé "répertoire parent"), il suffit d'utiliser sa désignation : `..` Par exemple, pour accéder au fichier `fichier_donnees_C.csv`, qui se trouve dans le répertoire `Dossier_C` contenu dans le répertoire `exercice_importation` (qui contient le répertoire `Exercice` et est donc le répertoire parent de `Exercice`), on peut écrire :

```
df_exercice_C <- read_csv2(file = "../Dossier_C/fichier_donnees_C.csv")
df_exercice_C
```

```
## # A tibble: 1 x 2
##   Dossier   Valeur
##   <chr>     <chr>
```

```
## 1 Dossier_C Bravo
```

10. Dans un objet que vous nommerez `df_exercice_D`, chargez dans R le contenu du fichier `fichier_donnees_D_1.csv`.
11. Enregistrez votre script dans le répertoire `Exercice` (en le nommant par exemple `importer_donnees.R`) et fermez RStudio.

4.1.1 Avantages de l'utilisation de chemins relatifs

Pour finir cette section, nous allons illustrer le grand avantage que confère l'emploi de chemins relatifs.

Mise en situation : admettons que vous souhaitez partager votre projet R avec une autre personne. Une manière simple de le faire consiste à créer une archive de l'ensemble de votre projet, pour fournir à la fois les scripts et les données.

Le répertoire `exercice_importation` se situe depuis le début de ce chapitre dans votre dossier de téléchargement. Dans l'explorateur de fichiers, allez dans le répertoire de téléchargements et créez une archive du répertoire `exercice_importation` :

12. clic droit sur le répertoire `exercice_importation` ;
13. dans le menu contextuel qui s'affiche, cliquez sur `7-Zip`, puis sur `Ajouter à "exercice_importation.zip"` ;
14. dans votre répetoire du cours d'informatique (dans vos documents), créez un répertoire intitulé `TD_4` ;
15. dans le répetoire `TD_4`, collez l'archive `exercice_importation.zip` que vous venez de créer ;
16. dans le répertoire de **téléchargement**, **supprimez** le répertoire `exercice_importation`.
17. dans le répetoire `TD_4`, extrayez le contenu de l'archive (clic droit, `7-Zip > Extraire vers "exercice_importation\"`).
18. descendez dans le répertoire `exercice_importation` qui vient d'être extrait, puis dans `Exercice` ;
19. Lancez le fichier `L3_info.Rproj` ;
20. Dans la session RStudio qui vient de se lancer, ouvrez le script `importer_donnees.R` que vous avez créé précédemment.

Si vous avez écrit les chemins vers les fichiers en utilisant des chemins absolus, vous serez obligé de les modifier un à un…

```
path <- paste0("C:/Users/votre_numero_etudiant/Downloads/",
               "exercice_importation/Exercice/fichier_donnees.csv")
read_csv2(file = path)
```

Cette instruction qui permettait de charger le fichier `fichier_donnees.csv` ne fonctionne plus. R retourne un message d'erreur indiquant "*No such file or directory*" (un tel fichier ou répertoire n'existe pas).

Si vous avez employé des chemins relatifs, vous n'aurez rien à modifier.

```
read_csv2(file = "./fichier_donnees.csv")
```

4.2 Chargement de formats différents

Cet exercice s'appuie sur la section 2.3 Importation, exportation et création de données des notes de cours.

Dans cet exercice, vous allez apprendre à importer des données dans R lorsqu'elles proviennent de différents types de fichiers. Au programme :

- des fichiers texte (plus particulièrement, CSV :*comma separated values*, et TSV, *tab-separated values*) ;
- des fichiers Excel ;
- des fichiers de données R.

4.2.1 Fichiers texte

Dans un premier temps, concentrons-nous sur les fichiers texte contenant des données tabulaires. Chaque ligne de ces fichiers correspond à une ligne d'un tableau, et un caractère spécifique appelé le séparateur de champ permet de séparer les colonnes. À la fin de chaque ligne, un caractère (non visible) permet de passer à la ligne suivante (fin de ligne ou *line break*). Parfois, les champs sont également placés entre double quotes (`"valeur"`).

Un format très répandu de ce type de fichiers s'appelle CSV, ou *comma separated values* : la virgule sépare les champs. Souvent, la première ligne sert à nommer les colonnes. Voici un exemple :

```
"nom","prenom","id","profile","matiere","note"
"Amsberry","Evan",1,1,"Microéconomie 1",11.5
"Basurto","Anthony",2,2,"Microéconomie 1",9
```

```
"Begaye","Kai",3,3,"Microéconomie 1",16.5
```

Dans certaines administrations françaises, les données au format CSV sont fournies avec une variante dans laquelle le séparateur de champ est le point-virgule plutôt que la virgule, cette dernière étant utilisée comme séparateur décimal.

```
"nom";"prenom";"id";"profile";"matiere";"note"
"Amsberry";"Evan";1;1;"Mathématiques 1";5
"Basurto";"Anthony";2;2;"Mathématiques 1";6
"Begaye";"Kai";3;3;"Mathématiques 1";18
```

Face à la possibilité d'utiliser différents caractères pour séparer les champs ainsi que pour séparer la partie entière de la partie décimale des nombres, R propose tout un attirail de fonctions. Nous allons en utiliser une qui permet d'être polyvalent : `read_delim()`. Cette fonction est proposée dans le *package* {readr}.

```
library(readr)
```

1. Affichez la page d'aide de la fonction `read_delim`.
2. Lisez les descriptions des arguments suivants : `file`, `delim`, `col_names`, `skip`, `n_max`.

Vous allez devoir charger dans R 4 fichiers texte contenant des notes d'étudiants, selon les matières qu'ils et elles ont suivies. Pour les besoins de l'exercice, les séparateurs de champs et les séparateurs décimaux varient d'un fichier à l'autre. Le tableau 4.1 reporte les valeurs utilisées.

Table 4.1 – Caractéristiques des fichiers texte avec séparateur de champ

Nom de fichier	Séparateur de champs	Séparateur décimal	En-têtes
`Microeconomie_1.csv`	Virgule (`,`)	Point (`.`)	Oui
`Macroeconomie_1.csv`	Virgule (`,`)	Point (`.`)	Non
`Mathematiques_1.csv`	Point-virgule (`;`)	Virgule (`,`)	Oui
`Informatique_1.tsv`	Taquet de tabulation (`\t`)	Point (`.`)	Oui

Ces fichiers sont disponibles dans une archive disponible à l'adresse suivante : https://egallic.fr/Enseignement/R/Exercices/exercice_importation.zip.

3. Téléchargez l'archive, extrayez son contenu, puis déplacez le répertoire `importation_formats` qui vient d'être extrait dans votre répertoire `TD_4` créé lors du précédent exercice.
4. En utilisant la fonction `read_delim()`, chargez le contenu du fichier `Microeconomie_1.csv` dans R. Ce fichier est situé dans le répertoire `L1`. Vous prendrez soin de stocker le tableau qui sera importé dans un objet que vous nommerez `microeconomie_1`.
5. En utilisant la fonction `read_delim()`, chargez le contenu du fichier `Macroeconomie_1.csv` dans R. Vous prendrez soin de stocker le tableau qui sera importé dans un objet que vous nommerez `macroeconomie_1`.

Lorsque le séparateur décimal n'est pas le point, il est nécessaire de renseigner à l'argument `locale` la valeur `locale(decimal_mark = ",")`.

6. En utilisant la fonction `read_delim()`, chargez le contenu du fichier `Mathematiques_1.csv` dans R. Vous prendrez soin de stocker le tableau qui sera importé dans un objet que vous nommerez `mathematiques_1`. Attention, vérifiez bien les valeurs des notes.
7. En utilisant la fonction `read_delim()`, chargez le contenu du fichier `Informatique_1.tsv` dans R. Vous prendrez soin de stocker le tableau qui sera importé dans un objet que vous nommerez `informatique_1`.

Le *package* {readr} propose également les fonctions `read_csv()` et `read_csv2()` qui sont spécifiquement créées pour des fichiers `texte/csv` répondant exactement au standard anglo-saxon (virgule en séparateur de champs, point en séparateur décimal) et français (point-virgule en séparateur de champs, virgule en séparateur décimal).

4.2.2 Fichier Excel

L'archive `importation_formats.zip` contient dans le répertoire `L1`, un fichier Excel intitulé `notes_L1.xlsx`.

8. Ouvrez ce fichier avec Excel, pour observer sa structure : le classeur est composé de 8 feuilles, une feuille par matière. Dans chaque feuille, les notes obtenues par les étudiants dans la matière correspondante sont reportées. La première ligne indique systématiquement les noms de clonnes.

Pour importer le contenu de fichiers Excel dans R, il existe plusieurs *packages*. Nous allons utiliser {readxl}.

9. Après avoir chargé le *package* {readxl}, affichez la page d'aide de la fonction `read_excel{.R}`. Lisez les descriptions des arguments suivants : `path`, `sheet`, `col_names`, `na`, `skip`, `n_max`.
10. En utilisant le numéro d'index de la feuille `info` (il s'agit de la 4e feuille), chargez son contenu, à l'aide de la fonction `read_excel` dans un objet que vous nommerez `info`.
11. En utilisant le nom de la feuille `stats`, chargez son contenu, à l'aide de la fonction `read_excel` dans un objet que vous nommerez `stats`.

4.2.3 Fichiers R

Il existe deux formats de fichier de données R : les fichiers `.RDS` et `.RData`. Les premiers contiennent un seul objet, tandis que les seconds peuvent en contenir plusieurs.

Dans le dossier `L1` de l'archive `importation_formats.zip` que dont vous avez extrait le contenu se trouve deux fichiers de données R :

— `anglais.RDS` : tableau de données contenant les notes d'anglais‘ ;
— `notes.RData` : 8 tableaux de données (`micro`, `macro`, `maths`, `info`, `stats`, `anglais`, `eco_envir`, et `finance`).

Pour importer un fichier `.RDS`, on utilise la fonction `readRDS()`. De la même manière que pour `read_delim()`, il est nécessaire de stocker le résultat dans un objet.

```
notes_anglais <- readRDS(file = "importation_formats/L1/anglais.RDS")
notes_anglais
```

```
## # A tibble: 32 x 5
##    nom        prenom     id matiere    note
##    <chr>      <chr>   <dbl> <chr>     <dbl>
##  1 Amsberry   Evan        1 Anglais 1   4
##  2 Basurto    Anthony     2 Anglais 1   5.5
##  3 Begaye     Kai         3 Anglais 1  11
##  4 Brack      Michael     4 Anglais 1   1.5
##  5 Burgess    Jamie       5 Anglais 1   7
##  6 Christian  Colton      6 Anglais 1   4
##  7 Cly        Jesse       7 Anglais 1   5
##  8 Coberley   Rebecca     8 Anglais 1   6
##  9 De Venecia Madison     9 Anglais 1   6
## 10 Gurule     Elliott    12 Anglais 1   6
## # ... with 22 more rows
```

Pour charger les objets d'un fichier `.RData` dans votre session R, il faut utiliser la fonction `load()`. Attention, si un objet est nommé `x` dans votre fichier `.RData` et qu'un objet existant dans votre session R s'appelle également `x`, ce dernier sera remplacé par l'objet qui vient d'être chargé avec la fonction `load()`.

```
load(file = "importation_formats/L1/notes.RData")
```

4.3 Exportation

4.3.1 Format texte

Pour enregistrer des données depuis un tableau de données, un vecteur ou une matrice, la fonction `write_delim()` peut être utilisée.

12. Affichez la page d'aide de la fonction `write_delim()`, et lisez les descriptions des arguments `x`, `file`, `delim` et `append`.
13. Dans un fichier que vous nommerez `notes_finance.csv`, exportez le tableau de données `finance` (qui a été chargé avec la fonction `load()`) dans le dossier `L1`.

4.3.2 Format R

Pour sauvegarder un seul objet, on peut utiliser la fonction `saveRDS()` et exporter les données au format `.RDS` :

```
saveRDS(maths, file = "importation_formats/L1/maths.RDS")
```

Pour sauvegarder plusieurs objets, on peut utiliser la fonction `save()`, en indiquant les noms des objets à sauvegarder en argument de la fonction.

```
save(micro, macro, maths, info, stats, anglais, eco_envir, finance,
     file = "importation_formats/L1/notes_L1.RData")
```

Chapitre 5

Tableaux de données

Ce chapitre s'appuie sur les Sections 2.2.2 Tableaux de données et 2.4.6 Tableaux de données des notes de cours.

Dans les chapitres Matrices et listes et Importation de données, vous avez appris à manipuler des matrices et à importer des données tabulaires dans R. Dans ce document, vous allez dans un premier temps utliser des techniques similaires à celles permettant de manipuler les matrices pour les appliquer à des tableaux que vous aurez importé depuis un fichier CSV dans R. Ensuite, vous apprendrez d'autres techniques proposées par l'environnement {tidyverse} pour manipuler des tableaux de données. Vous serez capable, une fois que vous aurez terminé de parcourir ce document, d'effectuer des opérations basiques de manipulation de tableau pour produire des statistiques descriptives.

5.1 Importation des données de la séance

Dans un premier temps, vous allez importer les données de notes des étudiant•es pour l'ensemble des matières, pour la première année (L1) et la deuxième année (L2). Ces données sont disponibles au format CSV, dans l'archive téléchargeable à l'adresse suivante : https://egallic.fr/Enseignement/R/Exercices/exercice_notes.zip.

1. Créez un nouveau projet RStudio dans un dossier que nous intitulerez `TD_5`, au sein de votre espace étudiant sur les serveurs de l'Université.
2. Dans le répertoire `TD_5`, créez trois répertoires : `Data`, `Script` et `Output`.
3. Créez un script R que vous appellerez `notes.R` et enregistrez-le dans le répertoire `Script`.

4. Téléchargez l'archive `exercice_notes.zip` en suivant le lien donné plus haut, et extrayez son contenu.
5. Placez les deux fichiers CSV (`notes_L1.csv` et `notes_L2.csv`) et le fichier de données R (`moyennes.RData`) dans le répertoire `Data`.

L'arborescence de votre dossier de travaux dirigés pour cette séance doit être comme suit :

```
TD_5
     notes.RProj
    Data
         notes_L1.csv
         notes_L2.csv
         moyennes.RData
    Script
|       notes.R
    Output
```

6. À présent, dans R, dans un objet que vous appellerez `notes_L1`, importez les données dans un tableau depuis le fichier `notes_L1.csv`.
7. Dans un objet que vous appellerez `notes_L2`, importez les données dans un tableau depuis le fichier `notes_L2.csv`.

En appelant les deux tableaux dans la console, vous devez voir les aperçus suivants :

```
notes_L1
```

```
## # A tibble: 224 x 5
##    nom        prenom     id matiere          note
##    <chr>      <chr>   <dbl> <chr>           <dbl>
##  1 Amsberry   Evan        1 Microéconomie 1  11.5
##  2 Basurto    Anthony     2 Microéconomie 1   9
##  3 Begaye     Kai         3 Microéconomie 1  16.5
##  4 Brack      Michael     4 Microéconomie 1   4
##  5 Burgess    Jamie       5 Microéconomie 1   6.5
##  6 Christian  Colton      6 Microéconomie 1  14.5
##  7 Cly        Jesse       7 Microéconomie 1   2.5
##  8 Coberley   Rebecca     8 Microéconomie 1  20
##  9 De Venecia Madison     9 Microéconomie 1   6
## 10 Gurule     Elliott    12 Microéconomie 1   7.5
## # ... with 214 more rows
```

```
notes_L2
```

```
## # A tibble: 217 x 5
##    nom         prenom     id matiere          note
##    <chr>       <chr>   <dbl> <chr>           <dbl>
##  1 Amsberry    Evan        1 Microéconomie 2   5.5
##  2 Basurto     Anthony     2 Microéconomie 2   5
##  3 Begaye      Kai         3 Microéconomie 2   8.5
##  4 Brack       Michael     4 Microéconomie 2   5
##  5 Burgess     Jamie       5 Microéconomie 2   9
##  6 Christian   Colton      6 Microéconomie 2  20
##  7 Coberley    Rebecca     8 Microéconomie 2  15.5
##  8 De Venecia  Madison     9 Microéconomie 2   5.5
##  9 Dewitt      Amelia     10 Microéconomie 2   6.5
## 10 Eisenberg   Kelly      11 Microéconomie 2   7
## # ... with 207 more rows
```

5.2 Tibbles

Les tableaux de données sont traditionnellement des objets à deux dimensions, comme les matrices. Une dimension pour les colonnes, et une autre pour les lignes. En R, les tableaux de données pricipaux sont appelés *data frames*. Nous allons, dans le cadre de ce cours, utiliser majoritairement des tableaux de données de classe `tbl_df`, une sous classe de `data.frame`, que l'on nommera des *tibbles*. Il s'agit de la structure de données centrale dans l'environnement tidyverse. Pour pouvoir créer ou manipuler des tibbles, il est nécessaire de charger le *package* {tibble}. Ce *package* est automatiquement chargé lorsque vous chargez {tidyverse} :

```
library(tidyverse)
```

Avant de manipuler les tableaux de données que vous venez d'importer, vous allez apprendre quelques notions élémentaires au sujet des tibbles. La fonction `tibble()` permet de créer un tableau de données. On lui donne en arguments les colonnes, en les séparant par une virgule. Pour une facilité d'utilisation, il est préférable de nommer ces colonnes, en inscrivant simplement le nom suivi du symbole `=` avant le contenu de la colonne :

```
tableau <- tibble(
  x = c(1,2,3,4, 5),
  y = c(10, 20, 30, 40, 50),
  z = c("A","B","C","D", "E"),
```

```
  t = x + y)
tableau
```

```
## # A tibble: 5 x 4
##       x     y z         t
##   <dbl> <dbl> <chr> <dbl>
## 1     1    10 A        11
## 2     2    20 B        22
## 3     3    30 C        33
## 4     4    40 D        44
## 5     5    50 E        55
```

Une colonne doit contenir des observations de même type. Comme vous l'avez vu dans le chapitre 2, les éléments d'un vecteur sont tous de même type. Aussi, dans l'exemple précédent, où chaque colonne est un vecteur, l'ensemble des éléments de chaque colonne est bien de même type (*double* pour `x`, `y` et `t`, *character* pour `z`).

Il est possible de donner des noms non conventionnels aux colonnes. En pratique, lorsque nous importons des données issues d'un fichier CSV, il n'est pas rare de faire face à des noms de colonnes non conventionnels (contenant des espaces par exemple, ou commençant par un nombre ou bien contenant des caractères spéciaux). Aussi, il est important de savoir comment faire face à cette situation : en utilisant des accents graves (`, *backtick*) :

```
tibble(
  `nom de colonne` = c(1,2,3)
)
```

```
## # A tibble: 3 x 1
##   `nom de colonne`
##              <dbl>
## 1                1
## 2                2
## 3                3
```

5.2.1 Dimensions

Lors de l'affichage dans la console de l'aperçu d'un tableau de données, les premières colonnes, leur type, et leur contenu s'affichent. Les dimensions sont données dans la première ligne d'affichage :

```
tableau
```

```
## # A tibble: 5 x 4
##       x     y z          t
##   <dbl> <dbl> <chr>  <dbl>
## 1     1    10 A         11
## 2     2    20 B         22
## 3     3    30 C         33
## 4     4    40 D         44
## 5     5    50 E         55
```

La première valeur correspond au nombre de lignes, tandis que la deuxième valeur correspond au nombre de colonnes. Pour accéder à ces valeurs, R propose la fonction `dim()`, qui retourne les dimensions sous la forme d'un vecteur :

```
dim(tableau)
```

```
## [1] 5 4
```

On peut ensuite, comme expliqué dans le chapitre introductif accéder à une dimension spécifique avec la fonction crochet :

```
dim(tableau)[1] # nombre de lignes
```

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

```
dim(tableau)[2] # nombre de colonnes
```

```
## [1] 4
```

Il est également possible d'utiliser les fonctions `nrow()` et `ncol()` qui donnent le nombre de lignes (*rows* en anglais) et de colonnes (*columns* en anglais), respectivement :

```
nrow(tableau)
```

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

```
ncol(tableau)
```

```
## [1] 4
```

5.2.2 Accès aux éléments

Pour accéder aux éléments d'un tibble, on peut utiliser les crochets simples (`[]`) ou doubles (`[[]]`), en utilisant l'indexation (par position, nom, ou condition). Avec les crochets simples, un tibble est retourné :

```
tableau[c(1,3),] # Lignes 1 et 3
```

```
## # A tibble: 2 x 4
##       x     y z         t
##   <dbl> <dbl> <chr> <dbl>
## 1     1    10 A        11
## 2     3    30 C        33
```

```
tableau[,c(2,4)] # Colonnes 2 et 4
```

```
## # A tibble: 5 x 2
##       y     t
##   <dbl> <dbl>
## 1    10    11
## 2    20    22
## 3    30    33
## 4    40    44
## 5    50    55
```

```
tableau[, c("x", "t")]
```

```
## # A tibble: 5 x 2
##       x     t
##   <dbl> <dbl>
## 1     1    11
## 2     2    22
## 3     3    33
## 4     4    44
## 5     5    55
```

Avec les crochets doubles, le contenu de la colonne que l'on souhaite extraire est retourné :

```
tableau[[2]]
```

```
## [1] 10 20 30 40 50
```

```
tableau[["z"]]
```

```
## [1] "A" "B" "C" "D" "E"
```

Pour extraire le contenu d'une colonne, R propose aussi d'utiliser le dollar plutôt que les crochets doubles, pour une écriture plus compacte :

```
tableau$z
```

```
## [1] "A" "B" "C" "D" "E"
```

Exercice

1. Affichez, à l'aide d'une fonction, le nombre de lignes du tableau `notes_1`.
2. Idem avec le nombre de colonnes.
3. Extrayez les valeurs de la colonne `note` du tableau `notes_1`.
4. Sur un moteur de recherche, cherchez comment afficher les valeurs distinctes d'un vecteur, puis appliquez les fruits de votre recherche pour afficher les valeurs distinctes de la colonne `matiere` du tableau `notes_1`. Stockez le résultat dans un objet que vous nommerez `matieres`.
5. Combien de matières différentes y a-t-il dans le tableau `notes_1` ? Utilisez le vecteur `matieres` et une fonction retournant le nombre d'éléments dans un vecteur pour répondre à cette question.

5.3 Résumés statistiques sur les colonnes d'un tableau

Des statistiques descriptives sur un vecteur peuvent facilement être obtenues à l'aide des fonctions suivantes :

— somme : `sum()`
— moyenne : `mean()`
— écart-type : `sd()`
— min : `min()`
— max : `max()`
— médiane : `median()`
— quantile quelconque : `quantile()`

Il suffit de donner un vecteur de valeurs numériques à ces fonctions pour obtenir la statistique calculée sur la base de l'échantillon de valeurs fournies.

La moyenne et l'écart-type de la colonne `x` du tableau `tableau` :

```
mean(tableau$x)
```

```
## [1] 3
```

```
sd(tableau$x)
```

```
## [1] 1.581139
```

Attention, la fonction `sd()` retourne la valeur de l'estimateur de l'écart-type d'une population : $\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$, où $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$.

```
n <- length(tableau$x)

moy <- sum(tableau$x) / n
moy
```

```
## [1] 3
```

```
variance <- sum((tableau$x - moy)^2) / (n-1)
variance
```

```
## [1] 2.5
```

```
ecart_type <- sqrt(variance)
ecart_type
```

```
## [1] 1.581139
```

Le minimum et le maximum :

```
min(tableau$x)
```

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

```
max(tableau$x)
```

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

La médiane :

```
median(tableau$x)
```

```
## [1] 3
```

Pour d'autres quantiles que la médiane, on utilise la fonction `quantile()`, en précisant le quantile à l'argument `probs` :

```
quantile(tableau$x, probs = .25) # Premier quartile
```

```
## 25%
##   2
```

```
quantile(tableau$x, probs = .75) # Troisième quartile
```

```
## 75%
##   4
```

```
quantile(tableau$x, probs = .90) # Neuvième décile
```

```
## 90%
## 4.6
```

En cas de présence de valeurs manquantes (`NA`), les fonctions `sum()` `mean()`, `sd()`, `min()`, etc. retournent la valeur `NA` :

```
x <- c(1,2,3,NA,4,NA)
mean(x)
```

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

Pour que la fonction soit appliquée en retirant au préalable du calcul les observations manquantes, on peut ajouter l'argument `na.rm=TRUE` (avec `rm` pour *remove*, c'est-à-dire retirer) :

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

```
## [1] 2.5
```

Exercice

1. À partir du tableau `notes_L1`, calculez la moyenne et l'écart-type des valeurs dans la colonne `note`.
2. Extrayez le vecteur de notes du tableau `notes_L1`, et conservez uniquement les 32 premières valeurs (ce qui correspond aux notes de microéconomie), à l'aide des crochets. Calculez la moyenne en microéconomie, ainsi que les premier et troisième quartiles.

5.4 Sélection d'une ou plusieurs colonnes

La fonction `select()` du *package* {dplyr} (qui est chargé lorsque {tiduverse} est chargé) permet de sélectionner une ou plusieurs colonnes. On donne un tableau de données comme premier argument à cette fonction. On indique ensuite la ou les noms des colonnes à conserver, en les séparant par une virgule. Admettons par exemple que l'on souhaite conserver uniquement les colonnes `x` et `z` de `tableau` :

```
tableau_2 <- select(tableau, x, z)
tableau_2
```

```
## # A tibble: 5 x 2
##       x z
##   <dbl> <chr>
## 1     1 A
## 2     2 B
## 3     3 C
## 4     4 D
## 5     5 E
```

Pour conserver l'ensemble des colonnes **excepté** certaines, il faut faire précéder le nom de ces colonnes que l'on souhaite ne pas conserver par le signe moins (`-`) dans la fonction `select`

```
select(tableau_2, -z)
```

```
## # A tibble: 5 x 1
##       x
##   <dbl>
```

```
## 1      1
## 2      2
## 3      3
## 4      4
## 5      5
```

L'ordre dans lequel sont écrits les noms de colonnes lors de l'appel à la fonction `select()` définit l'odre d'apparition des colonnes dans le résultat. Pour changer la position des colonnes d'un tibble, il existe une fonction dans {tidyverse} : `relocate()`. Si on souhaite voir apparaître certaines colonnes sans se soucier de l'ordre d'autres colonnes, il suffit de nommer les colonnes à positionner dans les premières positions :

```
tableau <- relocate(tableau, z, x)
tableau
```

```
## # A tibble: 5 x 4
##   z         x     y     t
##   <chr> <dbl> <dbl> <dbl>
## 1 A         1    10    11
## 2 B         2    20    22
## 3 C         3    30    33
## 4 D         4    40    44
## 5 E         5    50    55
```

Il est également possible de préciser que l'on désire placer une colonne avant ou après une autre colonne, en renseignant les arguments `.before` ou `.after` :

```
tableau <- relocate(tableau, z, .before = y)
tableau
```

```
## # A tibble: 5 x 4
##       x z         y     t
##   <dbl> <chr> <dbl> <dbl>
## 1     1 A        10    11
## 2     2 B        20    22
## 3     3 C        30    33
## 4     4 D        40    44
## 5     5 E        50    55
```

Exercice

1. À partir du tableau `notes_L1`, créez le tableau `eleves` qui contiendra uniquement les colonnes `nom`, `prenom` et `id`.
2. Appliquez la fonction `unique()` au tableau `eleves` pour ne conserver dans le tableau `eleves` que les enregistrements distincts (de sorte à avoir 1 ligne correspondant à une seule personne).

5.5 Création d'une nouvelle colonne

Le *package* {tidyverse} offre une fonction permettant de créer ou de modifier une colonne d'un tableau de données : `mutate()`. Son premier argument est le tableau de données, le ou les arguments suivants correspondent aux colonnes à créer ou modifier.

Illustrons le fonctionnement de cette fonction en ramenant les notes de la colonne `note` de `tableau` sur 10 plutôt que sur 20. Appelons `notes_10` la colonne contenant les résultats de la division par deux des valeurs de la colonne `note` :

```
notes_L1 <- mutate(notes_L1, notes_10 = note/2)
notes_L1
```

```
## # A tibble: 224 x 6
##    nom        prenom     id matiere          note notes_10
##    <chr>      <chr>   <dbl> <chr>           <dbl>    <dbl>
##  1 Amsberry   Evan        1 Microéconomie 1  11.5     5.75
##  2 Basurto    Anthony     2 Microéconomie 1   9       4.5
##  3 Begaye     Kai         3 Microéconomie 1  16.5     8.25
##  4 Brack      Michael     4 Microéconomie 1   4       2
##  5 Burgess    Jamie       5 Microéconomie 1   6.5     3.25
##  6 Christian  Colton      6 Microéconomie 1  14.5     7.25
##  7 Cly        Jesse       7 Microéconomie 1   2.5     1.25
##  8 Coberley   Rebecca     8 Microéconomie 1  20      10
##  9 De Venecia Madison     9 Microéconomie 1   6       3
## 10 Gurule     Elliott    12 Microéconomie 1   7.5     3.75
## # ... with 214 more rows
```

Admettons à présent que l'on souhaite ajouter 1 point à l'ensemble des élèves :

```
notes_L1 <- mutate(notes_L1, notes_10_augmentees = notes_10 +1)
```

Regardons les valeurs, en utilisant la fonction `select()` pour éviter d'afficher toutes les colonnes :

```
select(notes_L1, nom, note, notes_10, notes_10_augmentees)
```

```
## # A tibble: 224 x 4
##    nom          note notes_10 notes_10_augmentees
##    <chr>       <dbl>    <dbl>                <dbl>
##  1 Amsberry     11.5     5.75                 6.75
##  2 Basurto       9       4.5                  5.5
##  3 Begaye       16.5     8.25                 9.25
##  4 Brack         4       2                    3
##  5 Burgess       6.5     3.25                 4.25
##  6 Christian    14.5     7.25                 8.25
##  7 Cly           2.5     1.25                 2.25
##  8 Coberley     20       10                   11
##  9 De Venecia    6       3                    4
## 10 Gurule        7.5     3.75                 4.75
## # ... with 214 more rows
```

Ces opérations en plusieurs étapes peuvent se réaliser à l'aide d'une seule instruction :

```
notes_L1 <- mutate(notes_L1,
                   notes_10 = note / 2,
                   notes_10_augmentees = notes_10 + 1)
select(notes_L1, nom, note, notes_10, notes_10_augmentees)
```

```
## # A tibble: 224 x 4
##    nom          note notes_10 notes_10_augmentees
##    <chr>       <dbl>    <dbl>                <dbl>
##  1 Amsberry     11.5     5.75                 6.75
##  2 Basurto       9       4.5                  5.5
##  3 Begaye       16.5     8.25                 9.25
##  4 Brack         4       2                    3
##  5 Burgess       6.5     3.25                 4.25
##  6 Christian    14.5     7.25                 8.25
##  7 Cly           2.5     1.25                 2.25
##  8 Coberley     20       10                   11
##  9 De Venecia    6       3                    4
## 10 Gurule        7.5     3.75                 4.75
## # ... with 214 more rows
```

Exercice
Dans le tableau `notes_L1`, créez la colonne `notes_100` qui contiendra les notes exprimées sur 100 points et non pas sur 20.

5.6 Renommer une colonne

Pour renommer une colonne, le *package* {dplyr} propose la fonction `rename()`. Elle s'applique directement à un tableau de données et retourne un tableau de données. Il suffit d'indiquer le nouveau nom suivi du symbole égal (=) puis de l'ancien nom. Lorsqu'un nom n'est pas conventionnel, on utilise l'accent grave (`, *backtick*) pour entourer le nom :

```
notes_L1 <-
  rename(notes_L1,
         notes_dix = notes_10,
         `notes 10 augmentees` = notes_10_augmentees)
notes_L1
```

```
## # A tibble: 224 x 7
##    nom         prenom     id matiere          note notes_dix `notes
   10 augmente~`
##    <chr>       <chr>   <dbl> <chr>           <dbl>     <dbl>
                  <dbl>
##  1 Amsberry    Evan        1 Microéconomie 1  11.5      5.75
                    6.75
##  2 Basurto     Anthony     2 Microéconomie 1   9        4.5
                     5.5
##  3 Begaye      Kai         3 Microéconomie 1  16.5      8.25
                    9.25
##  4 Brack       Michael     4 Microéconomie 1   4        2
                       3
##  5 Burgess     Jamie       5 Microéconomie 1   6.5      3.25
                    4.25
##  6 Christian   Colton      6 Microéconomie 1  14.5      7.25
                    8.25
##  7 Cly         Jesse       7 Microéconomie 1   2.5      1.25
                    2.25
##  8 Coberley    Rebecca     8 Microéconomie 1  20       10
                      11
##  9 De Venecia  Madison     9 Microéconomie 1   6        3
                       4
## 10 Gurule      Elliott    12 Microéconomie 1   7.5      3.75
                    4.75
```

```
## # ... with 214 more rows
```

5.7 Filtrage

Pour conserver uniquement les lignes d'un tableau qui remplissent les conditions d'un filtre, la fonction `filter()` du package {dplyr} s'avère être très pratique. Cette fonction qui fait égalemnt partie de l'environnement {tidyverse}, s'applique directement à un tableau de données, et retourne un tableau de données comprenant les lignes pour lesquels le filtre retourne une valeur logique `TRUE`.

Le premier argument de la fonction `filter()` est le tableau de données. Les arguments suivants correspondent au(x) filtre(s) à appliquer. Par exemple, pour conserver uniquement les lignes du tableau `notes_L1` pour lesquelles la note est inférieure à 10 :

```
filter(notes_L1, note < 10)
```

```
## # A tibble: 148 x 7
##    nom        prenom     id matiere          note notes_dix `notes
   10 augmente~`
##    <chr>      <chr>   <dbl> <chr>           <dbl>     <dbl>
                  <dbl>
##  1 Basurto    Anthony     2 Microéconomie 1   9         4.5
                     5.5
##  2 Brack      Michael     4 Microéconomie 1   4         2
                     3
##  3 Burgess    Jamie       5 Microéconomie 1   6.5       3.25
                     4.25
##  4 Cly        Jesse       7 Microéconomie 1   2.5       1.25
                     2.25
##  5 De Venecia Madison     9 Microéconomie 1   6         3
                     4
##  6 Gurule     Elliott    12 Microéconomie 1   7.5       3.75
                     4.75
##  7 Israel     Joshua     14 Microéconomie 1   4         2
                     3
##  8 Jacket     Kanae      15 Microéconomie 1   6.5       3.25
                     4.25
##  9 Kinney     Jonah      17 Microéconomie 1   2         1
                     2
## 10 Ledbetter  Conor       18 Microéconomie 1   9         4.5
                     5.5
## # ... with 138 more rows
```

> Pour chaque ligne, le test logique consistant à regarder si la valeur de la colonne `note` est strictement supérieure à 10 est réalisé. Si la valeur retournée est `TRUE`, la ligne est retournée. Dans tous les autres cas, la ligne est retirée du résultat.

Des filtres plus complexes peuvent être réalisés, à l'aide des opérateurs logiques présentés dans la section portant sur l'indexation par condition (ET logique `&`, OU logique `|`, `!` négation logique).

Par exemple, pour obtenir les lignes du tableau `notes_L1` dont la valeur de la colonne `note` est comprise dans l'intervalle $[10; 12]$, on écrira :

```
filter(notes_L1, note >= 10 & note <= 12)
```

```
## # A tibble: 32 x 7
##    nom              prenom     id matiere         note notes_dix `
   notes 10 augm~`
##    <chr>            <chr>   <dbl> <chr>          <dbl>     <dbl>
              <dbl>
##  1 Amsberry         Evan        1 Microéconomi~  11.5       5.75
               6.75
##  2 Medicine Blanket Robert     23 Microéconomi~  11         5.5
                6.5
##  3 Thiede           Tyler      33 Microéconomi~  10.5       5.25
               6.25
##  4 Gurule           Elliott    12 Macroéconomi~  12         6
                 7
##  5 Medicine Blanket Robert     23 Macroéconomi~  10.5       5.25
               6.25
##  6 Pereyra          Emilie     27 Macroéconomi~  11.5       5.75
               6.75
##  7 Jacket           Kanae      15 Mathématique~  11.5       5.75
               6.75
##  8 Oyebi            Rachael    26 Mathématique~  12         6
                 7
##  9 Joy              Michael    16 Informatique~  10         5
                 6
## 10 Kinney           Jonah      17 Informatique~  10         5
                 6
## # ... with 22 more rows
```

Pour conserver les lignes pour lesquelles la valeur d'une colonne fait partie d'un ensemble de valeurs, on utilise l'opérateur `%in%`. Par exemple, pour conserver les observations du tableau `notes_L1` pour lesquelles le nom est `"Amsberry"` ou `"Basurto 1"`, on écrira :

```
filter(notes_L1, nom %in% c("Amsberry", "Basurto"))
```

```
## # A tibble: 14 x 7
##    nom      prenom     id matiere                note notes_dix `
   notes 10 augm~`
##    <chr>    <chr>   <dbl> <chr>                 <dbl>     <dbl>
            <dbl>
##  1 Amsberry Evan        1 Microéconomie 1        11.5      5.75
              6.75
##  2 Basurto  Anthony     2 Microéconomie 1         9        4.5
               5.5
##  3 Amsberry Evan        1 Macroéconomie 1         7.5      3.75
              4.75
##  4 Basurto  Anthony     2 Macroéconomie 1         4.5      2.25
              3.25
##  5 Amsberry Evan        1 Mathématiques 1         5        2.5
               3.5
##  6 Basurto  Anthony     2 Mathématiques 1         6        3
                 4
##  7 Amsberry Evan        1 Informatique 1          5.5      2.75
              3.75
##  8 Basurto  Anthony     2 Informatique 1          3.5      1.75
              2.75
##  9 Amsberry Evan        1 Statistiques 1          9        4.5
               5.5
## 10 Basurto  Anthony     2 Statistiques 1          4        2
                 3
## 11 Amsberry Evan        1 Anglais 1               4        2
                 3
## 12 Basurto  Anthony     2 Anglais 1               5.5      2.75
              3.75
## 13 Amsberry Evan        1 Economie environneme~   6        3
                 4
## 14 Basurto  Anthony     2 Economie environneme~  12        6
                 7
```

Exercice

1. Filtrez le tableau `notes_L1` pour ne conserver que les lignes pour lesquelles la valeur de la colonne `note` est supérieure ou égale à 10.
2. Filtrez le tableau `notes_L1` pour ne conserver que les lignes concernant la matière “Microéconomie 1”.
3. Filtrez le tableau `notes_L1` pour ne conserver que les lignes pour lesquelles la

valeur de la colonne `matiere` est soit `Microéconomie 1`, soit `Macroéconomie 1`. Stockez le résultat dans un objet que vous appellerez `eco`.

5.8 Agrégation par groupes

La colonne `matiere` du tableau de données `tableau` indique la matière dans lequel la note de l'élève est obtenue. Si l'on souhaite obtenir les moyennes de chaque matière, il est nécessaire de réaliser des agrégations par matière. Commençons par réaliser le calcul à la main.

Regardons les différentes matières :

```
unique(notes_L1$matiere)
```

```
## [1] "Microéconomie 1"             "Macroéconomie 1"
## [3] "Mathématiques 1"             "Informatique 1"
## [5] "Statistiques 1"              "Anglais 1"
## [7] "Economie environnementale 1" "Finance 1"
```

Calculons la note moyenne dans le groupe pour la microéconomie :

```
micro <- filter(notes_L1, matiere == "Microéconomie 1")
moyenne_micro <- mean(micro$note)
moyenne_micro
```

```
## [1] 8.96875
```

Pour la macroéconomie :

```
macro <- filter(notes_L1, matiere == "Macroéconomie 1")
moyenne_macro <- mean(macro$note)
moyenne_macro
```

```
## [1] 8.484375
```

Bien qu'il soit possible de continuer, on se rend rapidement compte que ce calcul répétitif est pénible. Heureusement, {tidyverse} propose une combo de fonctions qui permet de grouper les observations d'un tableau dans un premier temps, puis d'appliquer une fonction à une ou plusieurs colonnes à chacun des sous groupes, et enfin de retourner le résultat. Ces fonctions sont les suivantes :

1. `group_by()` : dans un premier temps, les données du tableau sont regroupées selon les modalités d'une ou de plusieurs colonnes. Il s'agit de créer des sous-groupes dans le tableau de données. Dans notre cas, nous allons effectuer le regroupement selon les valeurs prises par la colonne `matiere`.
2. `summarise()` : dans un deuxième temps, pour chaque groupe, les observations des lignes sont agrégées, résumées.

Regroupons les observations de `notes_L1` en fonction des valeurs de la colonne `matière` (la microéconomie 1 avec la microéconomie 1, la macroéconomie 1 avec la macroéconomie 1, etc.) :

```
group_by(notes_L1, matiere)
```

```
## # A tibble: 224 x 7
## # Groups:   matiere [8]
##    nom        prenom     id matiere          note notes_dix `notes
   10 augmente~`
##    <chr>      <chr>   <dbl> <chr>           <dbl>     <dbl>
                  <dbl>
##  1 Amsberry   Evan        1 Microéconomie 1  11.5      5.75
                    6.75
##  2 Basurto    Anthony     2 Microéconomie 1   9        4.5
                     5.5
##  3 Begaye     Kai         3 Microéconomie 1  16.5      8.25
                    9.25
##  4 Brack      Michael     4 Microéconomie 1   4        2
                       3
##  5 Burgess    Jamie       5 Microéconomie 1   6.5      3.25
                    4.25
##  6 Christian  Colton      6 Microéconomie 1  14.5      7.25
                    8.25
##  7 Cly        Jesse       7 Microéconomie 1   2.5      1.25
                    2.25
##  8 Coberley   Rebecca     8 Microéconomie 1  20       10
                      11
##  9 De Venecia Madison     9 Microéconomie 1   6        3
                       4
## 10 Gurule     Elliott    12 Microéconomie 1   7.5      3.75
                    4.75
## # ... with 214 more rows
```

On note dans la sortie précédente que cela correspond à 8 groupes (il y a en effet 8 valeurs distinctes dans la colonne `matiere` du tableau de données).

Une fois les groupes désignés, il reste à effectuer le calcul de résumé (la moyenne des valeurs

de la colonne `note`) sur chacun des groupes, à l'aide de la fonction `summarise()`.Le résultat du calcul sera donné dans une colonne que l'on pourra nommer, par exemple, `moyenne`.

```
moyennes_matieres <-
  summarise(group_by(notes_L1, matiere), moyenne = mean(note))
moyennes_matieres
```

```
## # A tibble: 8 x 2
##   matiere                      moyenne
##   <chr>                          <dbl>
## 1 Anglais 1                       7.11
## 2 Economie environnementale 1     8.56
## 3 Finance 1                       8.75
## 4 Informatique 1                  8.38
## 5 Macroéconomie 1                 8.48
## 6 Mathématiques 1                 9.05
## 7 Microéconomie 1                 8.97
## 8 Statistiques 1                  9
```

Pour éviter d'écrire les compositions de fonctions de manière condensée comme dans l'exemple précédent, le *package* {magrittr} propose un opérateur nommé "pipe", dont la syntaxe est la suivante `%>%`.

Cet opérateur fournit le résultat de l'évaluation de ce qui se trouve avant lui en premier argument de la fonction qui se situe immédiatement après.

```
moyennes_matieres <-
  notes_L1 %>%
  group_by(matiere) %>%
  summarise(moyenne = mean(note))
```

Le code devient plus facile à lire :

- on part du tableau de données `notes_L1` ;
- ce tableau est donné en premier argument de la fonction `group_by()` ;
- le regroupement est fait selon la colonne `matiere` ;
- une fois le regroupement effectué, le résultat est donné comme premier argument de la fonction `summarise()` ;
- le calcul de la moyenne des valeurs de la colonne `note` est effectué pour chaque groupe du tableau de données, et le résultat est indiqué dans une colonne nommée `moyenne`.

Si on souhaite effectuer des regroupements en fonction des valeurs de plusieurs colonnes,

il suffit d'ajouter les noms de colonnes dans la fonction `group_by()`. La syntaxe est la suivante :

```
# ne pas évaluer
group_by(tableau, colonne_1, colonne_2)
```

L'exercice qui suit vous permettra d'essayer un regroupement selon plusieurs colonnes.

Exercice

1. Intéressez-vous aux résultats de Rebecca Coberley. Créez un tableau que vous appellerez `notes_coberley` qui contiendra les observations de `notes_L1` concernant l'élève Rebecca Coberley.
2. Calculez la moyenne des notes de cette élève à partir du tableau `notes_coberley`.
3. À partir de `notes_L1`, calculez la moyenne générale de chaque élève. Pour ce faire :

a. regroupez les valeurs par les colonnes `nom`, `prenom`, `id`
b. calculez la moyenne de la colonne `note` pour chaque sous-groupe.

4. Retrouvez dans le tableau ainsi obtenu la moyenne de Rebecca Corbeley.

5.9 Tri

La fonction `arrange()` du *package* {dplyr} permet d'ordonner les observations par valeurs croissantes ou décroissantes d'une ou de plusieurs colonnes. À nouveau, le premier argument attendu est un tableau de données, et le résultat est un tableau de données. Les arguments suivants sont les noms des colonnes sur lesquels effectuer le tri. Par défaut, le tri se fait par valeurs **croissantes**.

Par exemple, pour trier les valeurs par valeurs croissantes de la colonne `notes`, on écrit simplement :

```
arrange(notes_L1, note)
```

```
## # A tibble: 224 x 7
##    nom       prenom      id matiere            note notes_dix `
   notes 10 augm~`
##    <chr>     <chr>    <dbl> <chr>             <dbl>     <dbl>
             <dbl>
##  1 Lehi      Savannah    19 Informatique 1       1       0.5
               1.5
```

```
##  2 Cly        Jesse        7 Mathématiques 1      1.5      0.75
               1.75
##  3 De Venecia Madison      9 Mathématiques 1      1.5      0.75
               1.75
##  4 Brack      Michael      4 Anglais 1            1.5      0.75
               1.75
##  5 Lucero     Cole        21 Economie environn~   1.5      0.75
               1.75
##  6 Brack      Michael      4 Finance 1            1.5      0.75
               1.75
##  7 Kinney     Jonah       17 Microéconomie 1      2        1
                 2
##  8 Cly        Jesse        7 Macroéconomie 1      2        1
                 2
##  9 Slovonsky  Tevin       32 Informatique 1       2        1
                 2
## 10 Thiede     Tyler       33 Informatique 1       2        1
                 2
## # ... with 214 more rows
```

Pour trier par valeurs décroissantes, on applique la fonction `desc()` sur le nom de la colonne, à l'intérieur de la fonction `arrange` :

```
arrange(notes_L1, desc(note))
```

```
## # A tibble: 224 x 7
##    nom              prenom     id matiere        note notes_dix `
   notes 10 augm~`
##    <chr>            <chr>   <dbl> <chr>         <dbl>     <dbl>
              <dbl>
##  1 Coberley         Rebecca     8 Microéconomi~    20      10
               11
##  2 Joy              Michael    16 Microéconomi~    20      10
               11
##  3 Massey           Taylor     22 Mathématique~    20      10
               11
##  4 Medicine Blanket Robert     23 Mathématique~    20      10
               11
##  5 Scholer          Julia      31 Informatique~    20      10
               11
##  6 Riley            Paige      28 Statistiques~    20      10
               11
##  7 Rosen            Richard    29 Anglais 1        20      10
               11
##  8 Rosen            Richard    29 Economie env~    20      10
```

```
                11
##  9 Begaye           Kai          3 Statistiques~    19       9.5
              10.5
## 10 Scholer          Julia       31 Statistiques~    19       9.5
              10.5
## # ... with 214 more rows
```

Lorsque une colonne utilisée pour faire un tri est une chaîne de caractères, l'ordre alphanumérique est utilisé :

```
arrange(notes_L1, nom)
```

```
## # A tibble: 224 x 7
##    nom      prenom     id matiere                note notes_dix `
   notes 10 augm~`
##    <chr>    <chr>   <dbl> <chr>                 <dbl>     <dbl>
             <dbl>
##  1 Amsberry Evan        1 Microéconomie 1        11.5      5.75
              6.75
##  2 Amsberry Evan        1 Macroéconomie 1         7.5      3.75
              4.75
##  3 Amsberry Evan        1 Mathématiques 1         5        2.5
              3.5
##  4 Amsberry Evan        1 Informatique 1          5.5      2.75
              3.75
##  5 Amsberry Evan        1 Statistiques 1          9        4.5
              5.5
##  6 Amsberry Evan        1 Anglais 1               4        2
                3
##  7 Amsberry Evan        1 Economie environneme~   6        3
                4
##  8 Basurto  Anthony     2 Microéconomie 1         9        4.5
              5.5
##  9 Basurto  Anthony     2 Macroéconomie 1         4.5      2.25
              3.25
## 10 Basurto  Anthony     2 Mathématiques 1         6        3
                4
## # ... with 214 more rows
```

> Selon l'ordre alphanumérique, les caractères spéciaux apparaissent en premier, suivis des valeurs numériques, puis par les lettres.

Pour trier par une colonne, puis par une autre, il suffit d'ajouter les noms des colonnes en arguments. L'ordre d'énumération définit l'ordre du tri. Par exemple, pour trier par

valeurs croissantes des matières (colonne `matiere`), puis, pour chaque matière, par valeurs décroissantes des notes (colonne `note`) :

```
arrange(notes_L1, matiere, desc(note))
```

```
## # A tibble: 224 x 7
##    nom              prenom     id matiere    note notes_dix `notes
   10 augmente~`
##    <chr>            <chr>   <dbl> <chr>     <dbl>     <dbl>
                  <dbl>
##  1 Rosen            Richard    29 Anglais 1  20        10
                    11
##  2 Joy              Michael    16 Anglais 1  13.5       6.75
                  7.75
##  3 Massey           Taylor     22 Anglais 1  13         6.5
                   7.5
##  4 Begaye           Kai         3 Anglais 1  11         5.5
                   6.5
##  5 Medicine Blanket Robert     23 Anglais 1  10.5       5.25
                  6.25
##  6 Pereyra          Emilie     27 Anglais 1  10         5
                    6
##  7 Mullins          Brenna     24 Anglais 1   9.5       4.75
                  5.75
##  8 Slovonsky        Tevin      32 Anglais 1   9.5       4.75
                  5.75
##  9 Schlaver         Lexa       30 Anglais 1   9         4.5
                   5.5
## 10 Kinney           Jonah      17 Anglais 1   8.5       4.25
                  5.25
## # ... with 214 more rows
```

Pour avoir une meilleure idée de ce qu'il se passe, regardons avec un plus petit tableau. Considérons le tableau suivant :

```
tableau <-
  tibble(
    numero_ligne_initial = c(1,2,3,4),
    matiere = c("Microéconomie 1", "Anglais 1",
                "Microéconomie 1", "Anglais 1"),
    notes = c(20,18,15,5))
tableau
```

```
## # A tibble: 4 x 3
##   numero_ligne_initial matiere          notes
##                  <dbl> <chr>            <dbl>
## 1                    1 Microéconomie 1    20
## 2                    2 Anglais 1          18
## 3                    3 Microéconomie 1    15
## 4                    4 Anglais 1           5
```

Si on ordonne uniquement par la colonne `matiere`, par valeurs croissantes :

```
arrange(tableau, matiere)
```

```
## # A tibble: 4 x 3
##   numero_ligne_initial matiere          notes
##                  <dbl> <chr>            <dbl>
## 1                    2 Anglais 1          18
## 2                    4 Anglais 1           5
## 3                    1 Microéconomie 1    20
## 4                    3 Microéconomie 1    15
```

Les lignes où `matiere` vaut `"Anglais 1"` sont remontées avant celles où `matiere` vaut `"Microéconomie 1"`.
Si on veut, une fois ce tri effectué, trier par valeurs croissantes de `notes`, en conservant les observations pour l'anglais avant celles de la microéconomie :

```
arrange(tableau, matiere, notes)
```

```
## # A tibble: 4 x 3
##   numero_ligne_initial matiere          notes
##                  <dbl> <chr>            <dbl>
## 1                    4 Anglais 1           5
## 2                    2 Anglais 1          18
## 3                    3 Microéconomie 1    15
## 4                    1 Microéconomie 1    20
```

À votre tour.

1. Trier le tableau `notes_L1` par valeurs décroissante des noms (colonne `nom`) .
2. Trier le tableau `notes_L1` par valeurs croissante des matières (colonne `matiere`) et valeurs croissante des noms (colonne `nom`).
3. Affichez le top 3 des élèves par matière. Pour ce faire :

— regroupez les les observations par matiere (colonne `matiere`),
— triez les observations par valeurs décroissantes des notes (colonne `note`),
— utilisez la fonction `slice_head()` sur le tableau trié (et groupé) pour extraire uniquement les 3 premières observations de chaque groupe.

5.10 Jointures

Il n'est pas rare d'avoir deux tableaux de données qui disposent d'une ou de plusieurs colonnes communes, et qui peuvent être joints. Dans l'archive téléchargée au début de ce chapitre, vous avez extrait un fichier intitulé `moyennes.RData` que vous avez placé dans le répertoire `Data` de votre projet. Ce fichier contient deux tableaux de données :

1. `moyennes_L1` : les moyennes à l'année de L1, sur 20, pour les élèves ;
2. `moyennes_L2` : les moyennes à l'année de L2, sur 20, pour les élèves.

Chargez dans R les données contenues dans le fichier `moyennes.RData`

La plupart des élèves de L1 sont aussi en L2 et vice-versa, mais certaines personnes étaient en L1 et n'ont pas été en L2, tandis que d'autres sont arrivées en L2 mais sont absentes du tableau concernant la L1.

Admettons que l'on souhaite mettre en commun les deux tableaux. Ils diposent tous deux d'une colonne indiquant les noms, une autre indiquant les prénoms, une autre indiquant l'identifiant, et enfin une dernière indiquant la moyenne annuelle. Cependant, le nom des colonnes de noms et prénoms diffère d'une année à l'autre : la colonne `nom` du tableau `moyennes_L1` s'appelle `laste_name` dans le tableau `moyennes_L2` et la colonne `prenom` dans `moyennes_L1` est désignée par `first_name` dans `moyennes_L2` :

```
moyennes_L1
```

```
## # A tibble: 32 x 4
##    nom        prenom     id moyenne_L1
##    <chr>      <chr>   <dbl>      <dbl>
##  1 Amsberry   Evan        1       6.93
##  2 Basurto    Anthony     2       6.36
##  3 Begaye     Kai         3      13.4
##  4 Brack      Michael     4       5
##  5 Burgess    Jamie       5       9.36
##  6 Christian  Colton      6      10.3
##  7 Cly        Jesse       7       5.29
##  8 Coberley   Rebecca     8       9.5
```

```
##  9 De Venecia Madison      9        4.07
## 10 Gurule     Elliott     12        7.71
## # ... with 22 more rows
```

```
moyennes_L2
```

```
## # A tibble: 31 x 4
##    last_name  first_name    id moyenne_L2
##    <chr>      <chr>      <dbl>      <dbl>
##  1 Amsberry   Evan           1       4.21
##  2 Basurto    Anthony        2       8.21
##  3 Begaye     Kai            3      15.6
##  4 Brack      Michael        4       6.36
##  5 Burgess    Jamie          5       9.21
##  6 Christian  Colton         6      10.2
##  7 Coberley   Rebecca        8      10.9
##  8 De Venecia Madison        9       5.79
##  9 Dewitt     Amelia        10       6.93
## 10 Eisenberg  Kelly         11      12.1
## # ... with 21 more rows
```

Si l'on souhaite réaliser une jointure entre les deux tableaux, une multitude de fonctions sont disponibles. Elles partagent une syntaxe commune :

`xx_join(x, y, by = NULL, copy = FALSE, ...)`,

où `x` et `y` sont les tableaux à joindre, `by` est un vecteur de chaînes de caractères contenant les noms des variables permettant la jointure (si la valeur est `NULL` – par défaut – la jointure se fera à l'aide des variables portant le même nom dans les deux tables).

Si la jointure s'effectue à l'aide de deux colonnes ou plus, on donner au paramètre `by` de la fonction de jointure un vecteur comportant plusieurs éléments, chaque élément indiquant la correspondance entre les noms de colonnes dans les deux tableaux.
La syntaxe est la suivante, si la colonne `nom_colonne_1_x` du tableau `x` correspond à la colonne `nom_colonne_1_y` du tableau `y`, et si la colonne `nom_colonne_2_x` du tableau `x` correspond à la colonne `nom_colonne_2_y` du tableau `y`.

```
xx_join(x, y, by = c("nom_colonne_1_x" = "nom_colonne_1_y",
                     "nom_colonne_2_x" = "nom_colonne_2_y"))
```

Pour illustrer les différentes jointures, travaillons sur un plus petit jeu de données. Vous aurez l'occasion de vous exercer sur le jeu complet par la suite.

```
moy_l1 <- tibble(
  nom = c("Fuentes Gomez", "Kohyann", "Zagoury", "Souvestre"),
  prenom = c("Sofia", "Hugo", "David", "Pierre"),
  id = c(1, 2, 3, 4),
  moyenne_L1 = c(12, 14, 12, 12),
  groupe_TD = c("A", "B", "A", "A")
)

moy_l2 <- tibble(
  last_name = c("Fuentes Gomez", "Kohyann", "Lang", "Souvestre"),
  first_name = c("Sofia", "Hugo", "Lea", "Pierre"),
  id = c(1, 2, 5, 4),
  moyenne_L2 = c(12, 14, 16, 13),
  groupe_TD = c("A", "A", "B", "B")
)

moy_l1
```

```
## # A tibble: 4 x 5
##   nom           prenom    id moyenne_L1 groupe_TD
##   <chr>         <chr>  <dbl>      <dbl> <chr>
## 1 Fuentes Gomez Sofia      1         12 A
## 2 Kohyann       Hugo       2         14 B
## 3 Zagoury       David      3         12 A
## 4 Souvestre     Pierre     4         12 A
```

```
moy_l2
```

```
## # A tibble: 4 x 5
##   last_name     first_name    id moyenne_L2 groupe_TD
##   <chr>         <chr>      <dbl>      <dbl> <chr>
## 1 Fuentes Gomez Sofia          1         12 A
## 2 Kohyann       Hugo           2         14 A
## 3 Lang          Lea            5         16 B
## 4 Souvestre     Pierre         4         13 B
```

Les différentes fonctions de jointure sont les suivantes :

— `inner_join()` : toutes les lignes de `x` pour lesquelles il y a des valeurs correspondantes dans `y`, et toutes les colonnes de `x` et `y`. S'il y a plusieurs correspondances dans les noms entre `x` et `y`, toutes les combinaisons possibles sont retournées ;

```
inner_join(moy_l1, moy_l2,
           by = c("nom" = "last_name", "prenom" = "first_name", "id"))
```

```
## # A tibble: 3 x 7
##   nom           prenom    id moyenne_L1 groupe_TD.x moyenne_L2
   groupe_TD.y
##   <chr>         <chr>  <dbl>      <dbl> <chr>            <dbl> <chr>
## 1 Fuentes Gomez Sofia      1         12 A                   12 A
## 2 Kohyann       Hugo       2         14 B                   14 A
## 3 Souvestre     Pierre     4         12 A                   13 B
```

— `left_join()` : toutes les lignes de `x`, et toutes les colonnes de `x` et `y`. Les lignes dans `x` pour lesquelles il n'y a pas de correspondance dans y auront des valeurs `NA` dans les nouvelles colonnes. S'il y a plusieurs correspondances dans les noms entre `x` et `y`, toutes les combinaisons sont retournées ;

```
left_join(moy_l1, moy_l2,
          by = c("nom" = "last_name", "prenom" = "first_name", "id"))
```

```
## # A tibble: 4 x 7
##   nom           prenom    id moyenne_L1 groupe_TD.x moyenne_L2
   groupe_TD.y
##   <chr>         <chr>  <dbl>      <dbl> <chr>            <dbl> <chr>
## 1 Fuentes Gomez Sofia      1         12 A                   12 A
## 2 Kohyann       Hugo       2         14 B                   14 A
## 3 Zagoury       David      3         12 A                   NA <NA>
## 4 Souvestre     Pierre     4         12 A                   13 B
```

— `right_join()` : toutes les lignes de `y`, et toutes les colonnes de `x` et `y`. Les lignes dans `y` pour lesquelles il n'y a pas de correspondance dans `x` auront des valeurs `NA` dans les nouvelles colonnes. S'il y a plusieurs correspondances dans les noms entre `x` et `y`, toutes les combinaisons sont retournées ;

```
right_join(moy_l1, moy_l2,
           by = c("nom" = "last_name", "prenom" = "first_name", "id"))
```

```
## # A tibble: 4 x 7
##   nom           prenom    id moyenne_L1 groupe_TD.x moyenne_L2
   groupe_TD.y
##   <chr>         <chr>  <dbl>      <dbl> <chr>            <dbl> <chr>
## 1 Fuentes Gomez Sofia      1         12 A                   12 A
## 2 Kohyann       Hugo       2         14 B                   14 A
```

```
## 3 Souvestre     Pierre     4          12 A                    13 B
## 4 Lang          Lea        5          NA <NA>                 16 B
```

— `semi_join()` : toutes les lignes de `x` pour lesquelles il y a des valeurs correspondantes dans `y`, en ne conservant uniquement les colonnes de `x` ;

```
semi_join(moy_l1, moy_l2,
          by = c("nom" = "last_name", "prenom" = "first_name", "id"))
```

```
## # A tibble: 3 x 5
##   nom           prenom    id moyenne_L1 groupe_TD
##   <chr>         <chr>  <dbl>      <dbl> <chr>
## 1 Fuentes Gomez Sofia      1         12 A
## 2 Kohyann       Hugo       2         14 B
## 3 Souvestre     Pierre     4         12 A
```

— `anti_join()` : toutes les lignes de `x` pour lesquelles il n'y a pas de correspondances dans `y`, en ne conservant que les colonnes de `x`.

```
anti_join(moy_l1, moy_l2,
          by = c("nom" = "last_name", "prenom" = "first_name", "id"))
```

```
## # A tibble: 1 x 5
##   nom     prenom    id moyenne_L1 groupe_TD
##   <chr>   <chr>  <dbl>      <dbl> <chr>
## 1 Zagoury David      3         12 A
```

Exercice

1. À partir des tableaux `moyennes_L1` et `moyennes_L2` (qui sont importés dans R en chargeant le fichier `moyennes.RData` que vous avez placé dans le répertoire `Data` de votre projet), effectuez une jointure permettant d'avoir dans un même tableau l'ensemble des élèves (ayant suivi ou non les deux années) et leur moyenne en L1 et en L2.
2. Identifiez à l'aide d'une jointure des tableaux `moyennes_L1` et `moyennes_L2` les élèves présents en L1 mais pas en L2.
3. Identifiez à l'aide d'une jointure des tableaux `moyennes_L1` et `moyennes_L2` les élèves présents en L2 mais pas en L1.

Chapitre 6

R Markdown

Ce chapitre vise à présenter R Markdown, un environnement de travail permettant de créer des documents en *data science*.

Dans un unique fichier R Markdown, il est possible d'écrire du code et de l'exécuter, puis de produire des rapports (mêlant textes, codes, et résultats des évaluations des codes) visant à être partagés. Les codes peuvent être du R, mais pas uniquement : il est possible d'évaluer des instructions d'autres langages tels python ou SQL, entre autres. Les formats de sortie possible sont nombreux. Parmi les plus utilisés : html, pdf, Word, notebook, ioslides.

Le *package* {rmarkdown} peut être installé à l'aide de l'instruction suivante.

```
install.packages("rmarkdown")
```

Le document de référence principal sur lequel s'appuie cette fiche est l'ouvrage R Markdown Cookbook, rédigé par Yihui Xie, Christophe Dervieux et Emily Riederer (Chapman & Hall/CRC, 2020). Une version électronique est disponible gratuitement à l'adresse suivante : https://bookdown.org/yihui/rmarkdown-cookbook/.

Une antisèche de deux pages sur le R Markdown a été réalisée par RStudio : https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf.

6.1 Créer un document R Markdown avec RStudio

Pour conserver la prise de bonnes habitudes initiée lors des premières séances, créez dans un premier temps un projet RStudio, en respectant l'arborescence présentée sur l'image ci-dessous.

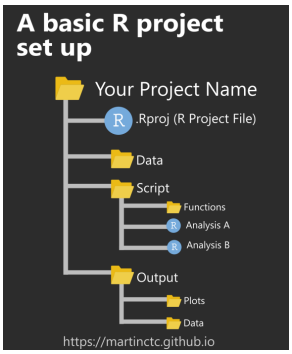


Figure 6.1 – Structure basique pour les projets. Source : https://martinctc.github.io/.

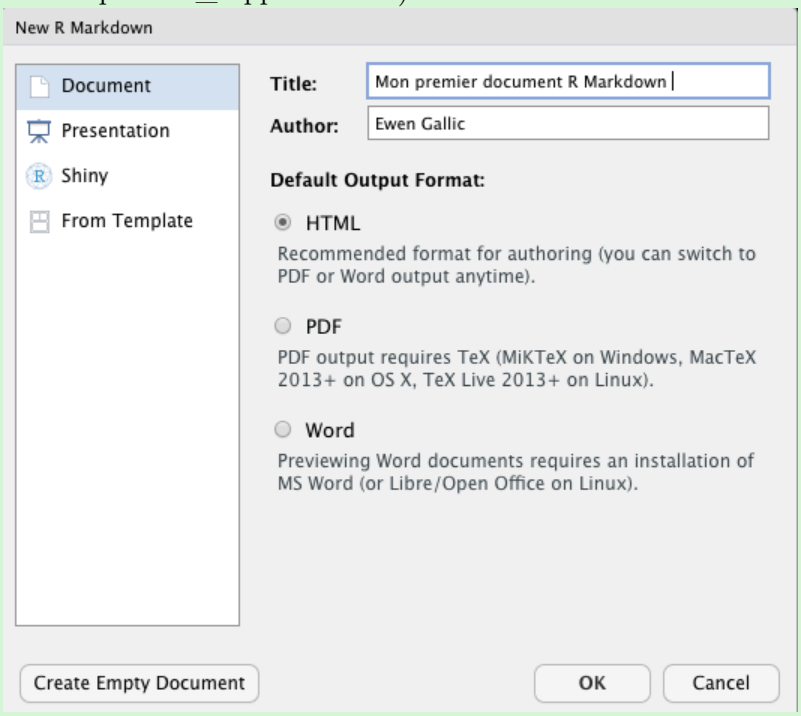
Création d'un projet
Dans RStudio :
- Cliquez sur le menu `File`, puis sur `New Project...`.
- Cliquez sur `New Directory`, puis sur `New Project`.
- Donnez un nom au nouveau projet, puis cliquez sur le bouton `Browse...`
- Choisissez l'emplacement du projet, puis appuyez sur le bouton `Open`.
- Cliquez sur le bouton `Create Project` pour créer le projet. Une nouvelle session RStudio s'ouvre alors. Le répertoire courant devient celui dans lequel vous avez créé le projet. Un fichier d'extention `.Rproj` a été créé dans ce répertoire. Il suffira d'ouvrir ce fichier à l'avenir pour ouvrir RStudio pour travailler sur ce projet.

Sur les ordinateurs de l'Université, vous veillerez à créer le projet dans le dossier `C:/Workout`. La compilation sur les VDI (*virtual desktop infrastructure*) semble être impossible pour le moment.
Attention : à la fin de la séance, il faudra penser à copier/coller l'ensemble du répertoire contenant votre projet dans votre dossier `Documents`. Les contenus du dossier `C:/Workout` sont effacés à la fermeture de session.

À présent que le projet est créé, il est temps de créer un document R Markdown.

Création d'un document R Markdown
Dans RStudio :
- Cliquez sur le menu `File`, puis sur `New File...`.
- Cliquez sur `R Markdown...`.
- Dans la fenêtre qui s'affiche :

  - assurez-vous d'être dans l'onglet "Document"
  - donnez un titre au document que vous allez créer
  - renseignez le champ "Author" avec vos noms et prénom(s)
  - laissez le bouton radio sur l'option `HTML` pour que le rapport qui sera créé après soit un document html (langage conçu pour présenter des pages web).
  - créer le document en cliquant sur le bouton `OK`.
- Sauvegardez le fichier créé en lui donnant le nom de votre choix (par exemple : 'premier_rapport.Rmd").



Un document R Markdown, dont l'extension est `.Rmd` est alors créé. Ce document est composé de trois parties :

- du YAML (des méta-données) ;
- du texte ;
- des morceaux de code (*code chunks*).

6.1.1 YAML

Dans le document que vous venez de créer, l'en-tête YAML indique :

```
title: "Mon premier document R Markdown"
author: "Ewen Gallic"
date: "11/2/2022"
output: html_document
```

Le titre du document, l'auteur et la date sont précisés dans cette en-tête. Lors de la conversion du fichier Rmd en fichier html par Pandoc (un logiciel de conversion de documents), ces informations seront stockées dans des variables et apparaîtront à un ou plusieurs endroits du fichier html (selon le modèle – *template* – utilisé). La ligne `output: html_document` indique quant à elle que le document de sortie sera un document html. D'autres éléments peuvent être indiqués, notamment dans la partie `output` : présence d'une table des matières, numérotation des sections, ajout d'une feuille de style, etc.

En quelques mots, les étapes de conversion sont les suivantes :

— la fonction `knit()` du *package* {`knitr`} exécute les codes contenus dans les *chunks* et prépare un fichier Markdow (`.md`)
— le logiciel Pandoc convertit le fichier `.md` dans un format de sortie (html, pdf, etc.)

Si le format de sortie est le pdf, une étape supplémentaire est ajoutée : le fichier `.md` est converti en un fichier LaTeX (`.tex`). Une compilation du fichier `.tex` est alors effectuée par LaTeX pour obtenir le fichier pdf final. Cela nécessite de fait d'installer LaTeX ou TinyTeX sur son système.

> Dans le cadre de ce cours, nous ne produirons pas de documents au format html sur les ordinateurs de l'Université.

6.1.1.1 Table des matières

Pour ajouter une table des matières dans un fichier html, on ajoute des paires de clés-valeurs :

— `toc: yes` : on désire la création d'une table des matières (*table of contents*) ;
— `toc_depth: 3` : l'entier donné en valeur définit la profondeur de la table des matières (1 : uniquement les sections, 2 : sections et sous-sections, 3 : sections, sous-sections et sous-sous sections, etc.)
— `toc_float: true` : la table des matières sera insérée comme objet flotant et visible en permanence tout le long du document.

```
---
title: "Mon premier document R Markdown"
```

```
author: "Ewen Gallic"
date: "11/7/2022"
output: html_document:
    toc: yes
    toc_depth: 3
    toc_float: true
---
```

> Attention, il faut respecter les indentations, comme dans l'exemple précédent.

Pour une table des matières sur un document final en pdf, le YAML doit contenir les paires `toc: yes` et `toc_depth:3`.

```
---
title: "Mon premier document R Markdown"
author: "Ewen Gallic"
date: "11/7/2022"
output: pdf_document:
    toc: yes
    toc_depth: 3
---
```

6.1.2 Textes

Le fichier `.Rmd` peut contenir du texte que l'on rédige en markdown. Davantage d'informations seront données dans la suite de cette fiche concernant la syntaxe markdown (qui est très simple).

6.1.3 Morceaux de code

Les morceaux de code contiennent deux parties :

— du code à évaluer, dans un langage donné (nous utiliserons du R) ;
— une liste d'options pour le morceau de code.

Pour être exécuté, le code fait appel à un environnement (dans lequel des variables peuvent être créées). Cet environnement peut être modifié après l'exécution du code.

6.1.4 Compilation

Pour compiler un document R Markdown, une fois le YAML bien spécifié, il suffit au choix :

- d'appeler la fonction `render()` du *package* {`rmarkdown`} :
  - en lui fournissant le chemin vers le fichier `Rmd` via l'argument `input` : `rmarkdown::render(input = "votre_document_rmarkdown.Rmd")`
  - il faut ensuite aller dans l'explorateur de fichier / dans Finder pour ouvrir le fichier converti
- de cliquer sur le bouton `Knit` (on le repère facilement avec son icone d'aiguille à tricoter et sa pelote de laine) ;
- d'appuyer simultanément sur les touches du clavier `Ctrl / Cmd + Shift + K`.

Les deux dernières solutions conduisent à un affichage du résultat dans une fenêtre qui s'ouvre à l'issue de la compilation.

Compilez votre premier fichier R Markdown avec l'une des trois manières indiquées, puis regardez le résultat.

6.2 Ecriture du texte : syntaxe en Markdown

Les parties textuelles qui permettent d'ajouter une narration aux rapports peuvent être rédigées en markdown. La sytaxe est très simple.

6.2.1 Texte

Il suffit d'écrire comme dans un bloc note pour que le texte soit affiché dans le rapport final.
Terminer une ligne par deux espaces permet d'aller à la ligne.

Laisser une ligne vide entre deux textes permet de créer un nouveau paragraphe.

6.2.2 Styles de texte

Style	Syntaxe	Exemple	Rendu
Gras	`** **` ou `__ __`	`Du texte **en gras**`	Du texte **en gras**
Italique	`* *` ou `_ _`	`Un mot en *italique*`	Un mot en *italique*
Barré	`~~ ~~`	`J'~~aime~~ adore R`	J'~~aime~~ adore R
Une partie en italique dans du gras	`**- -**`	`Un **texte _très_ important**`	Un **texte *très* important**
Tout en gras et italique	`*** ***`	`*** 新年快樂 *** (Xin nian kuai le)`	***新年快樂*** (Xin nian kuai le)
Exposant	`^ ^`	`Le 1^er^ janvier`	Le 1^er^ janvier

6.2.3 Titres

Il existe six niveaux de titres dans les documents R Markdown. On fait précéder le texte du titre par autant de croisillons (`#`) que de niveau désiré.

```
# Titre de niveau 1

## Titre de niveau 2

### Titre de niveau 3

#### Titre de niveau 4

##### Titre de niveau 5

###### Titre de niveau 6
```

Notes :
— il faut bien faire suivre le croisillon d'une espace ;
— il est nécessaire d'insérer une ligne vide avant et après le titre.

6.2.4 Citations

Pour effectuer une citation dans un bloc, il convient de faire précéder la citation du symbole >, que l'on place en début de ligne. Exemple avec`> chúc mừng năm mới` :

> chúc mừng năm mới

Pour faire en sorte qu'une citation comporte plusieurs paragraphes, un chevron (`>`) doit être ajouté au début de lignes vides.

```
>  “How can two people hate so much without knowing each other?”
>
> --- Alan Moore, _The Killing Joke_
```

> “How can two people hate so much without knowing each other ?”
> — Alan Moore, *The Killing Joke*

6.2.5 Tirets longs, tirets moyens

Pour insérer un tiret long (cadratin), on utilise trois tirets : `---` ; pour un tiret court (ou demi-cadratin), on utilise deux tirets : `--`.

Symbole voulu	Syntaxe	Exemple	Rendu
Tiret long (cadratin)	`---`	`--- une réplique`	— une réplique
Tiret moyen (demi-cadratin)	`--`	`La frontière France--Italie`	La frontière France–Italie

6.2.6 Lignes horizontale

En inscrivant trois tirets `---` et en passant immédiatement à la ligne, une ligne horizontale est insérée.

6.2.7 Points de suspension

Pour écrire des points de suspension, il suffit d'écrire trois points (`...`) à la suite…

6.2.8 Liens hypertextes

La création d'un lien hypertexte se fait en utilisant deux éléments : un texte sur lequel on clique, qui doit être entouré par des crochets `[]` et une adresse vers laquelle le lien point, qui doit être entourée par des parenthèses (`()`).

```
Regardez cette [vidéo épatante](https://www.youtube.com/watch?v=dQw4w9WgXcQ).
```

Regardez cette vidéo épatante.

Pour créer un lien sans définir de texte spécifique en remplacement de l'URL, il est possible de simplement écrire l'URL. Il est toutefois préférable d'encadrer l'URL par des chevrons. Il en va de même pour une adresse email.

```
<https://www.youtube.com/watch?v=oavMtUWDBTM>
<ewen.gallic@univ-amu.fr>
```

https://www.youtube.com/watch?v=oavMtUWDBTM
ewen.gallic@univ-amu.fr

Pour créer une ancre (un lien vers un endroit précis de la page déjà affichée) **vers un titre** du document, il faut connaître la référence vers le point d'ancrage. Un moyen simple consiste à la définir soi-même, en utilisant la syntaxe suivante :

```
# Le titre {#nom-de-la-reference}
```

Le nom de la reference ne doit pas contenir d'espaces ni de traits de soulignement (_). Il peut en revanche, comme dans l'exemple, contenir des tirets (-).

Dans ce document, la sous section dans laquelle ce texte est inscrit, est définie comme suit :

```
## Liens hypertextes {#liens-hypertextes}
```

```
On peut ainsi faire un lien vers cette [section](#liens-hypertextes).
```

On peut ainsi faire un lien vers cette section.

6.2.9 Notes de bas de page

L'insertion de notes de bas de page numérotées s'effectue à l'aide de crochets ([]) contenant un accent circonflèxe et une référence qui peut être soit un nombre, soit un texte (mais sans espace ou autre caractère blanc).

Le numéro de la note de bas de page est un lien qui renvoie vers la note. Une flèche de retour est proposée pour ramener au texte lorsque le document créé est un document html.

```
Une note de bas de page simple[^1] suivie d'une plus longue[^longue-note].

[^1]: la note de bas de page.

[^longue-note]: une note de bas de page plus longue.

    Dans laquelle on peut écrire un paragraphe.

    `{ du code }`

    Plusieurs paragraphes, même.
```

Une note de bas de page simple[1] suivie d'une plus longue[2].

1. la note de bas de page.
2. une note de bas de page plus longue.
   Dans laquelle on peut écrire un paragraphe.
   `{ du code }`
   Plusieurs paragraphes, même.

6.2.10 Listes

On distingue deux types de listes : les listes ordonnées et les listes non ordonnées.

6.2.10.1 Listes ordonnées

Pour créer une liste ordonnée, on place en début de ligne, devant chaque élément de la lique, un nombre immédiatement suivi d'un point et d'une espace.

```
1. Un premier élément ;
2. Un deuxième élément ;
3. Un troisième élément.
```

1. Un premier élément ;
2. Un deuxième élément ;
3. Un troisième élément.

Il n'est pas nécessaire de respecter la numérotation :

```
1. Un premier élément ;
10. Un deuxième élément ;
5. Un troisième élément.
```

1. Un premier élément ;
2. Un deuxième élément ;
3. Un troisième élément.

Le numéro du premier élément de la liste ordonnée définit la valeur du compteur :

```
4. Un premier élément ;
10. Un deuxième élément ;
5. Un troisième élément.
```

4. Un premier élément ;
5. Un deuxième élément ;
6. Un troisième élément.

6.2.10.2 Listes non ordonnées

Pour insérer une liste non ordonnée, on fait précéder chaque élément du symbole `-` ou du symbole `*`.

```
Une liste comprenant :

* un premier élément ;
* un deuxième élément ;
* un troisième élément.
```

Une liste comprenant :

— un premier élément ;
— un deuxième élément ;
— un troisième élément.

6.2.11 Listes imbriquées

Pour ajouter une liste à l'intérieur d'une liste, il faut ajouter avant le tiret ou l'étoile soit un taquet de tabulation, soit 4 espaces.

```
- un premier élément ;
- un deuxième élément :
    - qui contient un sous-élement,
    - ainsi qu'un deuxième sous-élément,
    - et un troisième ;
- puis un troisième élément.
```

— un premier élément ;
— un deuxième élément :
  — qui contient un sous-élement,
  — ainsi qu'un deuxième sous-élément,
  — et un troisième ;
— puis un troisième élément.

Pour écrire un paragraphe à l'intérieur d'une liste, une identation (taquet de tabulation ou 4 espaces) doit être ajoutéé, pour préserver la continuité de la liste. Il faut également faire précéder le paragraphe d'une liste vide (on peut aussi ajouter une ligne vide après le paragraphe, mais celle-ci est optionnelle).

```
- un premier élément ;
- un deuxième élément :

    qui contient un paragraphe.
```

```
- puis un troisième élément.
```

- un premier élément ;
- un deuxième élément :
  qui contient un paragraphe.
- puis un troisième élément.

Il est tout à fait possible d'imbriquer une liste ordonnée dans une liste non ordonnée et vice-versa.

```
1. Un premier élément :
    - avec un sous-élément ;
2. Un deuxième élément.
```

1. Un premier élément :
   - avec un sous-élément ;
2. Un deuxième élément.

Règles de typographie :
En français, les listes sont généralement introduites par une phrase se terminant par deux tirets (`:`). Les éléments de la liste ne commencent alors pas par une majuscule. Les éléments sont séparés par un point virgule (`;`), et le dernier élément se termine par un point.
Une liste comprenant :

- un premier élément ;
- un deuxième élément ;
- un troisième élément.

Lorsque la liste est numérotée, il faut en revanche mettre une majuscule à chaque élément de la liste
Une liste comprenant :

1. Un premier élément ;
2. Un deuxième élément ;
3. Un troisième élément.

Lorsqu'une liste se compose de plusieurs phrases, chaque élément commence par une majuscule et se termine par un point.
Les personnes pouvant postuler :

- Pour les personnes ayant validé une Licence 2 mention "Économie-Gestion" à Aix-Marseille Université ou dans une autre Université française : entrée de droit dans le parcours.

— Pour les personnes ayant validé un niveau équivalent à la Licence 2 en Économie ou en Économie-Gestion (Licence 2 mention Économie, DUT, CPGE, etc.) : une commission pédagogique évalue les dossiers notamment sur la base des pré-requis mentionnés plus haut ainsi que sur la base des projets de Master des étudiantes et étudiants.
— les règles de typographie françaises indiquent qu'il est d'usage de placer un point virgule à la fin des éléments de niveau 1 ou au dernier élément de niveau supérieur à 1 si celui-ci est immédiatement suivi par un élément de niveau 1 ;
— pour les éléments de niveau 2, une

En cas de listes imbriquées, la virgule est utilisée en fin de ligne, sauf pour le dernier élément de la liste imbriquée qui lui, s'achève par un point virgule ou un point s'il s'agit du dernier élément de la liste.
Une liste qui comprend :
— un élément de niveau 1
— un autre élément de niveau 1 :
— avec un sous-élément de niveau 2,
— ainsi qu'un autre ;
— un troisième élément de niveau 1.

6.2.12 Images

L'ajout d'une image se fait en insérant un point d'exclamation (`!`), suivi d'une d'un titre entre crochets, puis du chemin vers l'image, entre parenthèses (`()`). Une description de l'image peut être ajoutée entre guillemets (`""`) après le chemin, toujours dans les parenthèses (cette description est visible au survol de l'image, lorsque le pointeur de la souris reste quelques secondes au-dessus de l'image, et peut être lue à voix haute par les systèmes destinés aux personnes en situation de handicap). Enfin, pour spécifier des paramètres de taille de l'image, il est possible d'ajouter des informations entre crochets (`{}`).

```
![Un tigre](figs/tigre.JPG "Tigre en papier"){width="200px"}
```

6.2.13 Tableaux

La création d'un tableau se fait en distinguant l'en-tête du corps du tableau. Chaque cellule est séparée à l'aide d'une barre verticale (`|`, `Alt gr + 6` sur un clavier azerty sous Windows, et Alt + Shift + L‘ sur Mac Os). On ajoute des barres verticales au début et à la fin de chaque ligne du tableau. Pour délimiter l'en-tête du corps du tableau, trois tirets ou plus doivent être placés entre les barres verticales.

Figure 6.2 – Un tigre

```
| Première colonne | Deuxième colonne | Troisième colonne |
| ---------------- | ---------------- | ----------------- |
| cellule 1, ligne 1 | cellule 2, ligne 1 | cellule 3, ligne 1 |
| cellule 1, ligne 2 | cellule 2, ligne 2 | cellule 3, ligne 2 |
| cellule 1, ligne 3 | cellule 2, ligne 3 | cellule 3, ligne 3 |
```

Première colonne	Deuxième colonne	Troisième colonne
cellule 1, ligne 1	cellule 2, ligne 1	cellule 3, ligne 1
cellule 1, ligne 2	cellule 2, ligne 2	cellule 3, ligne 2
cellule 1, ligne 3	cellule 2, ligne 3	cellule 3, ligne 3

6.2.13.1 Alignement

La gestion de l'alignement des colonnes se fait à l'aide de la deuxième ligne, celle composée des tirets. Pour un alignement à gauche, on ajoute deux points (`:`) à gauche des tirets, pour un alignement à droite, on ajoute ces deux points à droite des tirets, et pour une colonne centrée, on ajoute les deux points à la fois à gauche et à droite.

```
| Colonne alignée à gauche | Colonne centrée | Colonne alignée à droite |
| :---------------- | :----------------: | -----------------: |
| cellule 1, ligne 1 | cellule 2, ligne 1 | cellule 3, ligne 1 |
| cellule 1, ligne 2 | cellule 2, ligne 2 | cellule 3, ligne 2 |
| cellule 1, ligne 3 | cellule 2, ligne 3 | cellule 3, ligne 3 |
```

Colonne alignée à gauche	Colonne centrée	Colonne alignée à droite
cellule 1, ligne 1	cellule 2, ligne 1	cellule 3, ligne 1
cellule 1, ligne 2	cellule 2, ligne 2	cellule 3, ligne 2

Colonne alignée à gauche	Colonne centrée	Colonne alignée à droite
cellule 1, ligne 3	cellule 2, ligne 3	cellule 3, ligne 3

6.2.14 Formules mathématiques en LaTeX

Il est possible d'utiliser du LaTeX pour écrure des équations mathématiques en Markdown. On distingue deux types de formules :

1. les formules en ligne : en entourant l'expression des symboles dollar (`$equation$`)
2. les formules en mode *display* : en entourant l'expression de deux symboles dollar de chaque côté (`$$equation$$`).

```
Les formules en ligne, $f(x)=x^2+3x+4, x\in\mathbb{R}$, permettent
d'insérer des formules dans un paragraphe, tandis que les formules en mode
_display_ permettent d'insérer des formules dans un bloc à part $$X\sim\mathcal{N}(0,1).$$
```

Les formules en ligne, $f(x) = x^2 + 3x + 4, x \in \mathbb{R}$, permettent d'insérer des formules dans un paragraphe, tandis que les formules en mode *display* permettent d'insérer des formules dans un bloc à part

$$X \sim \mathcal{N}(0, 1).$$

Une antisèche réalisée par Jim Hefferon, permettant de rédiger des formules en LaTeX est disponible sur le réseau complet d'archives TeX (CTAN) : http://tug.ctan.org/info/undergradmath/undergradmath.pdf.

6.2.15 Codes et blocs de code

Pour présenter du code, il existe deux manières en RMarkdown : soit des codes en ligne, directement insérés dans le texte, soit des blocs de code. Pour écrire des codes en ligne, on écrit le code entre deux accents graves (`` ` ``, `Alt gr + 7` sur un clavier azerty sous Windows, touche `` ` `` sous Mac Os). Si on désire avoir une coloration syntaxique, il faut de surcroît indiquer le langage entre crochets (`{}`), après un point (`{.nom-du-langage}`) :

```
La fonction `mean(x)`{.R} permet de calculer la moyenen de `x`.
L'instruction `SELECT * FROM base`{.SQL} permet de récupérer
l'ensemble des colonnes de la base `base` en SQL.
```

La fonction `mean(x)` permet de calculer la moyenen de `x`.
L'instruction `SELECT * FROM base` permet de récupérer l'ensemble des colonnes de la base `base` en SQL.

Pour insérer des blocs de code, on utilise trois accents grave (```) avant et après le code :

````
```
x <- mean(ventes$Sales, na.rm=TRUE)
```
````

Ce qui produira :

```
x <- mean(ventes$Sales, na.rm=TRUE)
```

À nouveau, si le langage du code contenu dans le bloc est spécifié, on obtient une coloration syntaxique. Le nom du langage est donné immédiatement après les trois accents graves.

````
```r
x <- mean(ventes$Sales, na.rm=TRUE)
```
````

Donnera :

```r
x <- mean(ventes$Sales, na.rm=TRUE)
```

6.3 Morceaux de code

Les codes et blocs de codes rédigés dans la section "Codes et blocs de code" ne sont jamais exécutés. Un des intérêts du RMarkdown réside dans la possibilité d'évaluer des codes (nous n'exécuterons ici que du R, mais il est possible d'exécuter d'autres langages dans un ficher RMarkdown), de les afficher, et d'afficher les résultats et sorties résultants de l'évaluation.

À nouveau, on peut distinguer deux types de codes : ceux en ligne et les blocs ou *chunks* de code.

6.3.1 Codes en ligne

Les codes en ligne sont délimités avec un accent grave au début et un à la fin, comme en markdown. La seule différence est que le nom du langage figure immédiatement après le premier accent grave. Le code ne sera pas retourné, seul son résultat apparaîtra dans le document final. Prenons un exemple simple.

````
La moyenne des notes est de `r mean(c(20, 18, 15, 16))`.
````

La moyenne des notes est de 17.25.

6.3.2 Blocs de code

Pour insérer des blocs de code, on utilise comme en markdown les trois accents graves avant et après le code. On fait suivre le troisième accent grave du début par le nom du langage, entre crochets ({r}) :

````
```{r}
x <- c(20, 18, 15, 16)
x
```
````

Le résultat est le suivant :

```r
x <- c(20, 18, 15, 16)
x
```

```
## [1] 20 18 15 16
```

On note que par défaut, le code est présent dans le document final ainsi que le résultat de son évaluation (tel qu'affiché dans la console).

Note : nous venons de créer l'objet `x` dans l'environnement. Nous pouvons le réutiliser dans un nouveau code, qu'il soit en ligne (`` `r ` ``) : 20, 18, 15, 16 ou dans un *chunk* (`` ```{r} ``` ``) :

````
```{r}
x
```
````

qui donne ensuite :

```r
x
```

```
## [1] 20 18 15 16
```

> **Note** : pour insérer un *chunk*, on peut utiliser le raccourci `Ctrl + Alt + I` sous Windows ou bien `cmd + alt/option + I` sous Mac Os.

6.3.2.1 Paramètres des chunks

Lorsqu'on définit un *chunk*, il est possible d'ajouter de nombreuses options qui auront un impact sur le fonctionnement du *chunk*. Par exemple, on peut demander d'afficher le code sans l'exécuter, d'afficher le résultat en masquant le code, ou bien de contrôler les

paramètres d'affichage des figures si le code en produit. Il existe plus de 50 paramétrages. La liste est disponible sur le site du créateur du *package* {`knitr`} : https://yihui.org/knitr/options/.

La syntaxe est la suivante :

````
```{r, options}
code
```
````

Une bonne pratique consiste à attribuer un identifiant unique à chaque *chunk*.

````
```{r identifiant-du-chunk, options}
x
```
````

Voici quelques options pratiques.

Syntaxe	Effet
`include = FALSE`	Ni le code ni les éventuels résultats affichés dans la console n'apparaitront dans le document final.
`echo = FALSE`	Le code n'apparaîtra pas dans le résultat final, tandis que les résultats seront visibles.
`eval = FALSE`	Le code apparaîtra dans le document final mais ne sera pas exécuté.
`message = FALSE`	Les messages générés par l'excution du code n'apparaîtront pas dans le document final.
`warning = FALSE`	Les messages d'avertissement générés par l'excution du code n'apparaîtront pas dans le document final.
`fig.cap = "titre"`	Si le code produit une image, le titre indiqué entre les guillemets sera utilisé comme titre de la figure

Syntaxe	Effet
`fig.width = "2"`, `figh.height="4`	Si le code produit une image, permet de définir sa largeur et sa hauteur, respectivement. Les valeurs sont données en pouces (*inches*).
`out.width='50%'`, `out.height='75%'`	Si le code produit ou affiche une image, permet de définir, en pourcentages, la taille de l'image.

Voici un exemple avec une image :

```
```{r plot-iris, out.width='75%',
fig.cap = "Largeur des sépals en fonction de leur longueur."}
plot(iris$Sepal.Length, iris$Sepal.Width)
```
```

Ce qui donne :

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

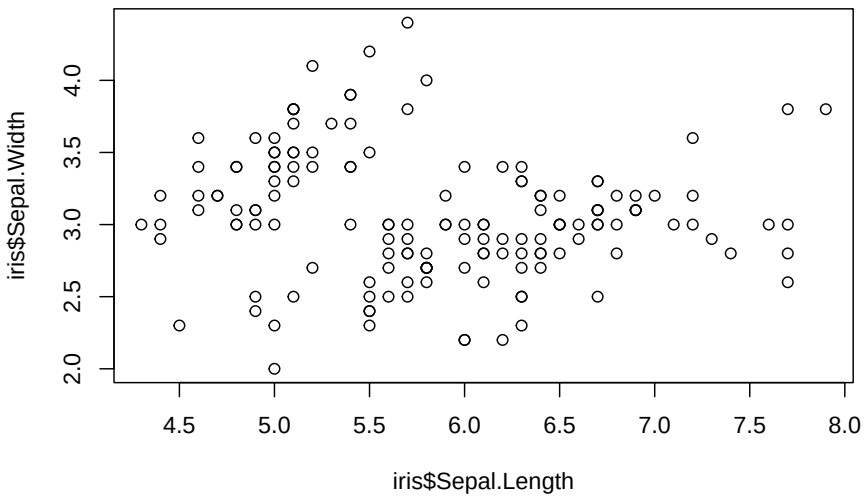


Figure 6.3 – Largeur des sépals en fonction de leur longueur.

Le paramétrage de l'**ensemble** des *chunks* est réglé par une liste (`knitr::opts_chunk`). En modifiant les valeurs de cette liste comme dans l'exemple ci-dessous, et en plaçant

cette instruction dans un *chunk* positionné juste après l'en-tête en YAML, tous les *chunks* adopteront le comportement spécifié.

Dans l'exemple ci-dessous, tous les codes des *chunks* seront visibles (`echo = TRUE`), les messages et les avertissements retournés lors de l'exécution des codes seront masqués (`message = FALSE` et `warning = FALSE`), les figures produites en R seront centrées (`fig.align = "center"`)

````
```{r chunks-settings, echo=FALSE, include=TRUE, message=FALSE, warning=FALSE}
knitr::opts_chunk$set(echo = TRUE,
                      message = FALSE,
                      warning = FALSE,
                      fig.align = "center")
```
````

Note : le paramétrage du comportement général des *chunks* peut ensuite être modifié pour certains *chunks* spécifiques. Pour ce faire, il suffit d'ajouter dans les paramètres des *chunks* à modifier le comportement désiré. Par exemple, si `echo = TRUE` est défini dans la liste des paramètres de la liste `knitr::opts_chunk`, si on souhaite masquer le code d'un *chunk* spécifique, il suffit de lui ajouter l'option `echo = FALSE` :

````
```{r, echo=FALSE}
# code du chunk
```
````

6.3.3 Mise en cache

Lorsqu'un document RMarkdown comprend des calculs qui prennent beaucoup de temps à s'effectuer lors de la compilation, il est possible de mettre en cache les résultats de tout ou partie des *chunks*. Pour ce faire, il suffit d'indiquer le paramètre `cache = TRUE` dans le ou les *chunks* que l(on souhaite mettre en cache.

Lors de la compilation du document, les *chunks* ayant été mis en cache ne seront pas exécutés par {`knitr`} s'ils ont été exécutés lors de la précédente compilation et qu'aucune modification ne leur a été apportée. Si le code à l'intérieur d'un *chunk* a été modifié entre le moment où il a été exécuté par {`knitr`} lors d'une compilation et une nouvelle compilation (ou si un paramètre du *chunk* autre que `include` a été modifié), alors il sera de nouveau réévalué et remis en cache.

Il est risqué de paramétrer l'ensemble des *chunks* de sorte qu'ils soient tous mis en cache. Si le j^{e} *chunks* est modifié, il sera réévalué par {`knitr`} lors de la compilation. Or, si l'éva-

luation dépend de variables définies avant le j^{e} *chunk* mais ayant été modifiées *après*, les résultats retournés par la nouvelle évaluation du j^{e} *chunk* risquent de ne pas être reproductibles…

Il est parfois utile de vider le cache pour éviter les erreurs mentionnées précédemment. Pour ce faire, on peut cliquer sur la flèche pointant vers le bas située à droite du bouton `Knit`, puis cliquer sur la ligne commençant par l'emoji ballet `Clear Knitr Cache...`.

Note : la mise en cache est surtout utile lordque l'on charge des résultats d'estimation de modèles longs à calculer (des modèles de machine-learning par exemple).

6.4 Insertion d'une bibliographie

Citer une source répond à des normes. Une des normes fréquemment rencontrées en économie est celle de l'association américaine de psychologie (American Psychological Association). Elle reprend l'acronyme de l'association : APA.

Des explications en français sur ces normes sont proposées sur le site des bibliothèques de l'Université de Montréal à l'adresse suivante https://bib.umontreal.ca/citer/styles-bibliographiques/apa.

On distingue deux manières de citer des références : dans le texte et en bibliographie.

Avec RMarkdown, nous n'avons qu'à nous soucier de la citation dans le texte. L'ajout des références citées en bibliographie se fait ensuite automatiquement, en respectant la norme utilisée. Pour cela, il faut :

1. créer un fichier contenant les références, en les distinguant à l'aide d'un identifiant unique ;
2. créer le lien entre ce fichier et le document RMarkdown, en le renseignant dans l'entête YAML ;
3. insérer les références aux documents, dans la partie narrative (texte) du document RMarkdown.

6.4.1 Création du fichier de bibliographie

Un fichier de bibliographie au format `.bib` (BibTeX) doit dans un premier temps être créé. Ce fichier contient les références, formatées selon le standard attendu des fichiers bib. Sur le site bibtex.com, la page suivante propose un guide sur les formats BibTeX. https://www.bibtex.com/g/bibtex-format/.

Chaque entrée dans le fichier BibTeX correspons à une référence bibliographique. Elle est composée de trois parties :

1. le type de document : book (livre), article, inbook (un chapitre de livre), inproceedings (un article de colloque), techreport (un rapport technique), etc ;
2. un identifiant unique (que l'on utilise ensuite dans le fichier RMarkdown pour faire référence à cette entrée) ;
3. une liste de paires de clés-valeurs, où les éléments de la référence sont stockés (titre, auteurs, année de publication, nom de la revue, nom de l'éditeur, etc.).

Prenons un exemple :

```
@book{xie_2018_rmarkdown,
  title={R markdown: The definitive guide},
  author={Xie, Yihui and Allaire, Joseph J and Grolemund, Garrett},
  year={2018},
  publisher={Chapman and Hall/CRC}
}
```

Il s'agit d'un livre, donc le type de document est indiqué à l'aide de `@book`. La référence unique est `xie_2018_rmarkdown`, et les paires de clés-valeurs sont renseignées dans une liste où chaque clé est donnée avant le symbole égal et chaque valeur est donnée après le symbole égal. Les paires sont séparées par une virgule.

À votre tour : créez un nouveau fichier de script R dans RStudio. Dans ce fichier, collez l'entrée bibliographique du livre de Yihui Xie et al. Enregistrez le document en le nommant `biblio.bib`.
Il vous sera demandé si vous souhaitez changer l'extension du fichier pour effectivement utiliser `.bib` plutôt que `.R` : dites oui.

Afin d'éviter de devoir rédiger à la main l'ensemble des entrées du fichier BibTeX, il existe de nombreux outils pour nous faciliter le travail. Nous allons en voir quelques-uns.

6.4.1.1 Crossref

Dans un premier temps, on peut vérifier si la référence existe sur Crossref. Ce site est très pratique pour obtenir des citations généralement propres, mais il a un inconvénient : il faut parfois être patient, le site rencontre fréquemment des erreurs 500 (*Internal Server Error*).

Essayons de chercher la référence pour l'article de Susan Athey intitulé "The State of Applied Econometrics : Causality and Policy Evaluation", publié dans "The Journal of economic perspectives" (JEP) en 2017.

Après avoir cliqué sur le bouton "Search metadata", on peut coller le titre de l'article : `The State of Applied Econometrics: Causality and Policy Evaluation`, puis lancer la

recherche en appuyant sur la touche `Enter` du clavier.

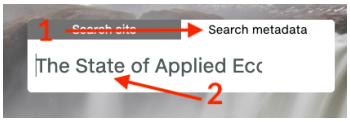


Figure 6.4 – Chercher l'article.

Parmi les références proposées, lorsqu'on a identifié celle qui correspond effectivement à celle que l'on recherche, on récupère les métadonnées en cliquant sur le lien `Actions` puis sur `Cite`.

Figure 6.5 – Identifier le bon résultat.

L'entrée bibliographique au format BibTeX est alors affichée, il ne reste plus qu'à la copier et à la coller dans notre fichier bib.

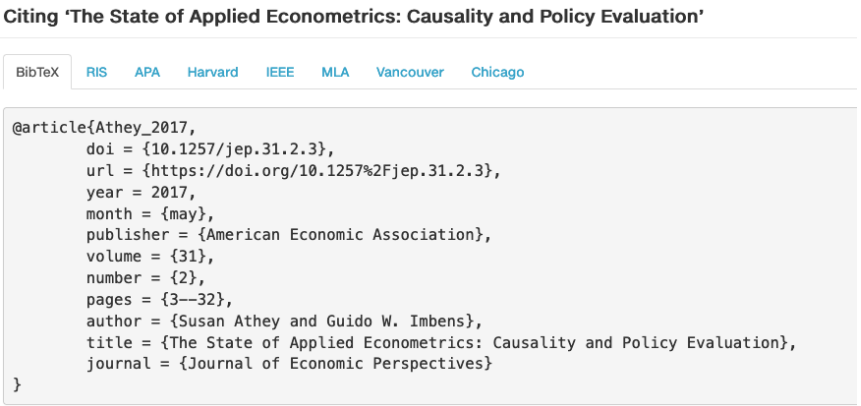


Figure 6.6 – L'entrée au format BibTeX.

À votre tour : recherchez la référence de l'article de Susan Athey publié dans le JEP en 2017 et collez l'entrée dans votre fichier BibTeX.

Votre fichier BibTeX doit à présent contenir deux entrées (n'oubliez pas de le sauvegarder) :

```
@book{xie_2018_rmarkdown,
  title={R markdown: The definitive guide},
  author={Xie, Yihui and Allaire, Joseph J and Grolemund, Garrett},
  year={2018},
  publisher={Chapman and Hall/CRC}
}
@article{Athey_2017,
    doi = {10.1257/jep.31.2.3},
    url = {https://doi.org/10.1257%2Fjep.31.2.3},
    year = 2017,
    month = {may},
    publisher = {American Economic Association},
    volume = {31},
    number = {2},
    pages = {3--32},
    author = {Susan Athey and Guido W. Imbens},
    title = {The State of Applied Econometrics:
    Causality and Policy Evaluation},
    journal = {Journal of Economic Perspectives}
}
```

6.4.1.2 Google Scholar

Parfois, on ne trouve pas l'information désirée sur Crossref. Il peut alors être une bonne idée d'aller chercher sur Google Scholar si la référence existe. Pour l'heure, les références sur Google Scholar ont l'air d'être moins bien renseignées que sur Crossref.

Essayons de retrouver la référence du papier d'Olivier Deschênes et de Michael Greenstone publié dans l'American Economic Review (AER) en 2007. Cherchons le papier par son titre :

> The economic impacts of climate change : evidence from agricultural output and random fluctuations in weather

Une fois l'article identifié dans les résultats, il suffit de cliquer sur le lien `Citer`.

Une fenêtre surgit. Les citations pré-formatées selon différents styles sont données. Pour récupérer les métadonnées au format BibTeX, il faut cliquer sur le lien `BibTeX`.

Les métadonnées s'affichent dans une page web. Il ne reste plus qu'à effectuer un copier/-coller dans le fichier BibTeX.

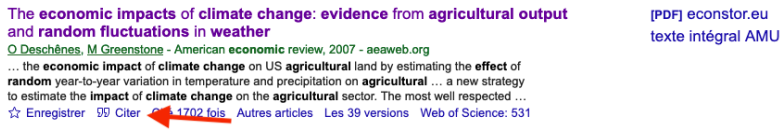


Figure 6.7 – L'article de Deschênes et Greenstone sur Google Scholar.

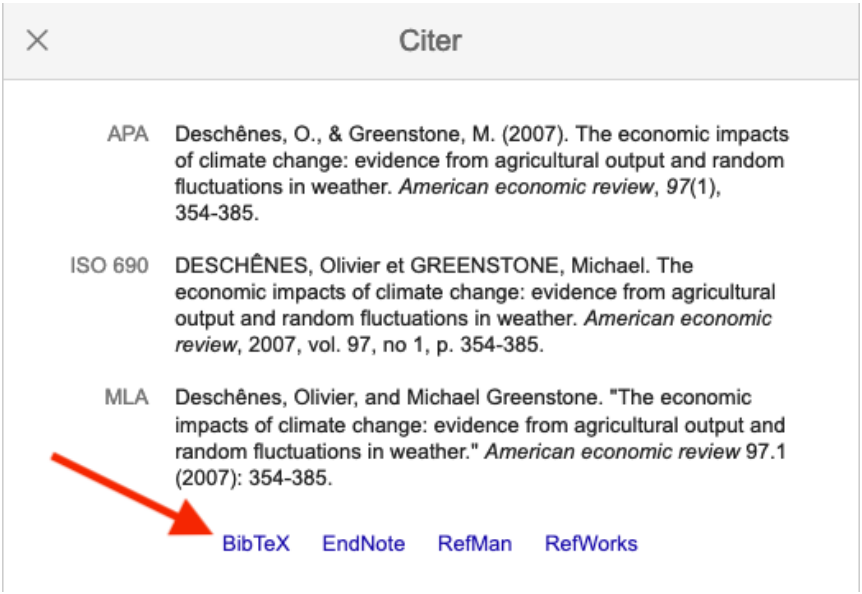


Figure 6.8 – L'article de Deschênes et Greenstone sur Google Scholar.

À votre tour : recherchez la référence de l'article d'Olivier Deschênes et de Michael Greenstone publié dans l'AER en 2007 et collez l'entrée dans votre fichier BibTeX.

Votre fichier BibTeX doit à présent contenir trois entrées (n'oubliez pas de le sauvegarder) :

```
@book{xie_2018_rmarkdown,
  title={R markdown: The definitive guide},
  author={Xie, Yihui and Allaire, Joseph J and Grolemund, Garrett},
  year={2018},
  publisher={Chapman and Hall/CRC}
}
@article{Athey_2017,
    doi = {10.1257/jep.31.2.3},
    url = {https://doi.org/10.1257%2Fjep.31.2.3},
    year = 2017,
    month = {may},
    publisher = {American Economic Association},
    volume = {31},
    number = {2},
    pages = {3--32},
    author = {Susan Athey and Guido W. Imbens},
    title = {The State of Applied Econometrics:
    Causality and Policy Evaluation},
    journal = {Journal of Economic Perspectives}
}
@article{deschenes2007economic,
  title={The economic impacts of climate change:
  evidence from agricultural output and random fluctuations in weather},
  author={Desch{\^e}nes, Olivier and Greenstone, Michael},
  journal={American economic review},
  volume={97},
  number={1},
  pages={354--385},
  year={2007}
}
```

6.4.1.3 Un générateur de citations

Si jamais la référence que l'on cherche n'est disponible ni sur le site du journal ou de l'éditeur, ni sur crossref, ni sur Google Scholar, il reste la possibilité d'utiliser un générateur

de citations comme celui proposé sur ce site : https://www.bibme.org/bibtex. Il est également évidemment possible de rentrer l'ensemble des informations à la main (mais gare aux erreurs !).

6.4.2 Lier le fichier d'entrées bibliographiques avec le document RMarkdown

Une fois le fichier BibTeX créé, il faut renseigner dans l'en-tête YAML (en-tête de métadonnées) :

1. un chemin vers le fichier de bibliographie (dans l'exemple ci-dessous, on suppose que le fichier `biblio.bib` se trouve dans le même répertoire que le fichier RMarkdown) ;
2. éventuellement, on peut préciser le style voulu (ici, nous utilisons la norme APA) ;
3. éventuellement, on peut indiquer que nous souhaitons que les citations soient cliquables et renvoient vers la bibliographie

```
---
output: html_document
bibliography: biblio.bib
biblio-style: apalike
link-citations: yes
---
```

6.4.3 Citer les documents dans le corps de texte

Nous disposons à présent d'un fichier BibTeX contenant 3 entrées, l'en-tête du document RMarkdown précise où se situe ce fichier d'entrées bibliographiques. Reste à présent à citer les entrées dans le texte.

Pour citer un document, on doit utilise son identifiant unique que l'on place entre crochets, après un symbole arobase : `[@identifiant]`.

Description	Syntaxe	Affichage dans le texte
Citation entre parenthèses	`[@xie_2018_rmarkdown]`	(Xie, Allaire, and Grolemund 2018)
Supprimer la mention de l'auteur	`[-@xie_2018_rmarkdown]`	(2018)
Ajout d'informations supplémentaires	`[voir @xie_2018_rmarkdown, pp. 10--12]`	(voir Xie, Allaire, and Grolemund 2018, 10–12)

Description	Syntaxe	Affichage dans le texte
Citation multiple	`[@xie_2018_rmarkdown;Athey_2017]`	(Xie, Allaire, and Grolemund 2018 ; Athey and Imbens 2017)
Citation sans les parenthèses pour les auteurs	`@xie_2018_rmarkdown`	Xie, Allaire, and Grolemund (2018)

La bibliographie à la fin du document sera automatiquement ajoutée. On peut placer une section `# Références` à la fin du document pour faire une rupture avec la section précédente. Cette section ne sera pas numérotée.

Chapitre 7

Statistiques descriptives

Ce chapitre vous permet de mobiliser vos connaissances acquises avec le chapitre sur les tableaux de données et le chapitre sur R Markdown pour produire rapidement des résumés statistiques à partir de données tabulaires.

7.1 Données du chapitre

Dans ce chapitre, vous allez manipuler des données qui proviennent de l'article de Bertrand and Mullainathan (2004) intitulé *Are Emily and Greg More Employable Than Lakisha and Jamal ?*, dans lequel les auteurs se sont penchés sur l'existence de discrimination sur le marché du travail aux États-Unis. Dans leur travail, les chercheurs ont envoyé des faux CVs en réponse à des annonces postées dans des journeaux à Boston et à Chicago. Ces faux CVs ont été réalisés de manière aléatoire, en faisant varier diverses caractéristiques : le genre de la personne qui postule, la qualité de rédaction, l'expérience, le niveau d'études, etc. Ce qui intéresse les auteurs est de mesurer la discrimination raciale. Plutôt que d'indiquer l'ethnicité des personnes sur les CVs, les auteurs ont choisi de donner des noms à consonnance afro-américaine ou caucasienne aux faux CVs, en attribuant de manière aléatoire les noms aux faux CVs.

Le *package* {AER} fournit les données de l'étude. Si ce *package* n'est pas installé sur votre machine, vous pouvez l'installer comme suit :

```
install.packages("AER")
```

Ensuite, vous pouvez charger le jeu de données `ResumeNames` comme suit :

```
data(ResumeNames, package = "AER")
```

Pour afficher la page d'aide du jeu de données `ResumeNames` et ainsi accéder au descriptif des variables, vous pouvez écrire dans la console R :

```
help("ResumeNames", package = "AER")
```

Nous allons nous concentrer sur les variables listées dans le Tableau 7.1 (libre à vous d'en conserver davantage par la suite pour explorer ces données) :

Table 7.1 – Caractéristiques des fichiers texte avec séparateur de champ

Variable	Description
`name`	Prénom de la personne qui postule (personne fictive)
`gender`	Genre de la personne qui postule (`female` ou `male`)
`ethnicity`	Ethnicité de la personne qui postule (`cauc` : prénom à consonnance caucasienne ou `afam` : prénom à consonnance afro-américaine)
`call`	La personne qui postule a-t-elle été rappelée ? (`no` ou `yes`)
`city`	Ville concernée par l'annonce (`chicago` ou `boston`)
`jobs`	Nombre d'emplois précédents listés dans le CV
`experience`	Nombre d'années d'expériences listées dans le CV

Créons un objet contenant uniquement les colonnes mentionnées dans le Tableau 7.1.

```
resumes <-
  ResumeNames %>%
  select(name, gender, ethnicity, call, city, jobs, experience) %>%
  as_tibble()
resumes
```

```
## # A tibble: 4,870 x 7
##    name    gender ethnicity call  city     jobs experience
##    <fct>   <fct>  <fct>     <fct> <fct>   <int>      <int>
##  1 Allison female cauc      no    chicago     2          6
##  2 Kristen female cauc      no    chicago     3          6
```

```
##  3 Lakisha female afam      no    chicago     1          6
##  4 Latonya female afam      no    chicago     4          6
##  5 Carrie  female cauc      no    chicago     3         22
##  6 Jay     male   cauc      no    chicago     2          6
##  7 Jill    female cauc      no    chicago     2          5
##  8 Kenya   female afam      no    chicago     4         21
##  9 Latonya female afam      no    chicago     3          3
## 10 Tyrone  male   afam      no    chicago     2          6
## # ... with 4,860 more rows
```

7.2 Résumés statistiques à la main

Dans un premier temps, il faut apprendre à connaître le contenu des données, fouiller dedans. Cette étape primordiale dans tout projet permet bien souvent de déceler des erreurs dans les données, ou de soulever des questions nécessitant un traitement spécifique.

7.2.1 Dimensions, résumés grossiers

La première chose à savoir sur un jeu de données est la suivante : **quelles en sont ses dimensions ?** Nous souhaitons connaître le nombre d'observations (n) et le nombre de colonnes (p). Nous l'avons vu dans le chapitre portant sur les tableaux de données, pour ce faire, la fonction `dim()` est très pratique :

```
dim(resumes)
```

```
## [1] 4870    7
```

Un simple affichage du tableau de données dans la console permet par ailleurs d'avoir accès à ces informations. L'avantage de `dim()` est que les valeurs peuvent être stockées dans un objet et réutilisées ensuite (dans le cadre de la réalisation de boucles, par exemple).

Pour avoir un aperçu grossier du tableau de données, nous pouvons utiliser la fonction `summary()` :

```
summary(resumes)
```

```
##       name          gender     ethnicity    call           city
##  Tamika : 256   male  :1124   cauc:2435   no :4478   boston :2166
##  Anne   : 242   female:3746   afam:2435   yes: 392   chicago:2704
##  Allison: 232
##  Latonya: 230
##  Emily  : 227
```

```
##  Latoya : 226
##  (Other):3457
##       jobs          experience
##  Min.   :1.000   Min.   : 1.000
##  1st Qu.:3.000   1st Qu.: 5.000
##  Median :4.000   Median : 6.000
##  Mean   :3.661   Mean   : 7.843
##  3rd Qu.:4.000   3rd Qu.: 9.000
##  Max.   :7.000   Max.   :44.000
##
```

Pour chaque variable, en fonction de son type, des informations sont proposées :

- pour des variables **numériques** : moyenne, étendue, quartiles ;
- pour des variables **catégorielles** : la fréquence des 6 premières valeurs les plus présentes.

Pour la variable `name` du tableau `resumes`, nous pouvons voir qu'il semble y avoir plus de 6 valeurs. Nous pouvons les lister avec la fonction `unique()` :

```
unique(resumes$name)
```

```
##  [1] Allison  Kristen  Lakisha  Latonya  Carrie   Jay      Jill
   Kenya
##  [9] Tyrone   Aisha    Geoffrey Matthew  Tamika   Leroy    Todd
   Greg
## [17] Keisha   Brad     Laurie   Meredith Anne     Emily    Latoya
   Ebony
## [25] Brendan  Hakim    Jamal    Neil     Tremayne Brett    Darnell
   Sarah
## [33] Jermaine Tanisha  Rasheed  Kareem
## 36 Levels: Allison Anne Carrie Emily Jill Laurie Kristen Meredith
   ... Tyrone
```

Pour dénombrer la fréquence de chaque modalité d'une variable catégorielle, on peut utiliser la fonction `table()` :

```
table(resumes$name)
```

```
##
##  Allison     Anne   Carrie    Emily     Jill   Laurie  Kristen
   Meredith
##      232      242      168      227      203      195      213
   187
```

```
##    Sarah      Brad  Brendan Geoffrey      Greg    Brett       Jay
   Matthew
##      193        63        65        59        51        59        67
         67
##     Neil     Todd    Aisha    Ebony   Keisha    Kenya  Lakisha
   Latonya
##      76        68       180       208       183       196       200
   230
##   Latoya   Tamika  Tanisha  Darnell    Hakim    Jamal Jermaine
   Kareem
##      226       256       207        42        55        61        52
         64
##    Leroy  Rasheed Tremayne   Tyrone
##       64        67        69        75
```

Rien ne nous empêche de trier les résultats avec la fonction `sort()` :

```
sort(table(resumes$name))
```

```
##
##  Darnell      Greg Jermaine    Hakim Geoffrey    Brett    Jamal
   Brad
##       42        51        52        55        59        59        61
         63
##   Kareem    Leroy  Brendan       Jay  Matthew  Rasheed     Todd
   Tremayne
##       64        64        65        67        67        67        68
         69
##   Tyrone     Neil   Carrie    Aisha   Keisha Meredith    Sarah
   Laurie
##       75        76       168       180       183       187       193
   195
##    Kenya  Lakisha      Jill  Tanisha    Ebony  Kristen   Latoya
   Emily
##      196       200       203       207       208       213       226
   227
##  Latonya  Allison      Anne   Tamika
##      230       232       242       256
```

Exercice

1. En utilisant la fonction `table()`, affichez le nombre de réponses "*no*" et "*yes*" pour le jeu de données `resumes`. Stockez le résultat dans une variable que vous nommerez `freq_reponses`.

2. Il est possible d'accéder aux éléments d'un objet de type `table` à l'aide des crochets simples (`[]`). En utilisant les crochets sur l'objet `freq_reponses`, calculez la proportion de "*yes*" dans le jeu de données.

7.2.2 Exploration des valeurs avec des tableaux

7.2.2.1 Tableaux à double entrée

La fonction `table()` est également très pratique pour faire des tableaux croisés. Pour ce faire, il convient de fournir en argument les vecteurs de valeurs (contenant des variables catégorielles de type *character* ou *factor*). Par exemple, dans l'article dont les données de ce chapitre sont issues, nous pouvons regarder combien de candidatures ont reçu ou non un retour, selon l'ethnicité :

```
call_ethnicity_tb <-
  table(resumes$call, resumes$ethnicity)
call_ethnicity_tb
```

```
##
##        cauc afam
##   no   2200 2278
##   yes   235  157
```

Pour davantage de lisibilité du tableau, les arguments de la fonction peuvent être nommés ; le tableau croisé affichera ainsi une étiquette pour les lignes et les colonnes :

```
call_ethnicity_tb <- table(call = resumes$call, ethnicity = resumes$ethnicity)
call_ethnicity_tb
```

```
##       ethnicity
## call   cauc afam
##   no   2200 2278
##   yes   235  157
```

Les marges peuvent être ajoutées avec la fonction `addmargins()` :

```
addmargins(call_ethnicity_tb)
```

```
##       ethnicity
## call   cauc afam  Sum
##   no   2200 2278 4478
##   yes   235  157  392
```

```
##   Sum 2435 2435 4870
```

Pour obtenir les proportions sans avoir à utiliser les crochets comme dans l'exercice plus haut, R propose la fonction `prop.table()`, que l'on applique au tableau croisé :

```
prop.table(table(call = resumes$call, ethnicity = resumes$ethnicity))
```

```
##       ethnicity
## call         cauc       afam
##   no   0.45174538 0.46776181
##   yes  0.04825462 0.03223819
```

Ou en utilisant l'opérateur pipe (`%>%`) pour davantage de lisibilité :

```
table(call = resumes$call, ethnicity = resumes$ethnicity) %>%
  prop.table()
```

```
##       ethnicity
## call         cauc       afam
##   no   0.45174538 0.46776181
##   yes  0.04825462 0.03223819
```

Il est possible d'appliquer une fonction sur le tableau croisé ; par exemple, on peut vouloir appliquer la fonction `round()` pour afficher moins de décimales :

```
table(call = resumes$call, ethnicity = resumes$ethnicity) %>%
  prop.table() %>%
  round(digits = 2)
```

```
##       ethnicity
## call  cauc afam
##   no  0.45 0.47
##   yes 0.05 0.03
```

Les proportions marginales peuvent être obtenues en précisant l'argument `margin` à la fonction `prop.table()`. En optant pour `margin = 1`, on obtient les fréquences relatives à chaque ligne (ici, on peut lire la fréquence marginale des ethnicités selon la réponse obtenue ou non) :

```
table(call = resumes$call, ethnicity = resumes$ethnicity) %>%
  prop.table(margin = 1) %>%
  round(2)
```

```
##      ethnicity
## call  cauc afam
##   no  0.49 0.51
##   yes 0.60 0.40
```

En optant pour `margin = 2`, on obtient les fréquences relatives à chaque colonne (ici, on peut lire la fréquence marginale des réponses selon l'ethnicité, ce qui est particulièrement intéressant dans la question posée par le papier de recherche) :

```
table(call = resumes$call, ethnicity = resumes$ethnicity) %>%
  prop.table(margin = 2) %>%
  round(digits = 2)
```

```
##      ethnicity
## call  cauc afam
##   no  0.90 0.94
##   yes 0.10 0.06
```

Exercice
Dans une expérience aléatoire, pour mesurer l'effet d'un traitement, il est important de s'assurer du côté aléatoire du traitement. Si on veut mesurer l'effet de l'ethnicité sur la probabilité d'être contacté lorsque l'on postule à une offre d'emploi, et que l'on veut aussi mesurer l'effet du genre sur cette probabilité, il est important que le genre et l'ethnicité ne soient pas des caractéristiques liées. On doit donc créer l'expérience de sorte que la proportion de CVs de femmes et d'hommes soit statistiquement identique selon que la personne soit afro-américaine ou caucasienne.

1. Affichez la proportion de femmes et d'hommes dans le jeu de données `resumes`.
2. Idem avec la proportion d'afro-américains et de caucasiens.
3. Affichez un tableau croisé montant les fréquences de femems et d'hommes selon l'ethnicité.
4. Affichez ensuite les fréquences marginales permettant de penser que l'ethnicité et le genre ne sont pas liés dans l'expérience.

Dans le cours de Statistiques Inférentielles de 2e année, vous avez appris à effectuer un test d'indépendance à l'aide d'une statistique du Khi-deux.

```
effectifs_obs <- table(call = resumes$call, ethnicity = resumes$ethnicity)
```

```
prop_table <-
  effectifs_obs %>%
  prop.table(margin = 2)
prop_table
```

```
##      ethnicity
## call        cauc       afam
##   no  0.90349076 0.93552361
##   yes 0.09650924 0.06447639
```

Avec ce tableau croisé, on lit que 9,6% des CV de personnes caucasiennes ont obtenu une réponse, tandis que seulement 6,4% des CV de personnes afro-américaines en ont obtenu une. Cette différence de 3,2% est-elle due au hasard ? Sous l'hypothèse nulle du test de Khi-deux, on s'attend à ce que les proportions soient identiques, indépendamment de l'origine ethnique. Si nous rejetons l'hypothèse nulle de ce test, nous serons amenés à penser que la différence de3,2% observée n'est pas due au hasard ; autrement dit, nous pourrons conclure, avec un risque de première espèce donné, que le taux de réponses suite à une candidature pour un emploi à Boston et Chicago est dépendant de l'ethnicité (et qu'il y a donc de la discrimination).

La statistique du test est la suivante :

$$\chi^2 = \sum_{j=1}^{q} \sum_{i=1}^{p} \frac{(n_{ij} - n_{ij}^e)^2}{n_{ij}^e},$$

avec q le nombre de sous-populations (ici, $q = 2$: afro-américains et caucasiens) et p le nombre de modalités du caractère étudié (ici, $p = 2$: rappel ou non), n_{ij} le nombre obseré d'observations dans la sous-population j disposant du caractère i et n_{ij}^e le nombre d'observations théoriques que l'on s'attend à observer dans la sous-population j avec le caractère i si l'hypothèse nulle est vraie.

Les effectifs observés sont les suivants :

```
effectifs_obs
```

```
##      ethnicity
## call  cauc afam
##   no  2200 2278
##   yes  235  157
```

Si l'etnicité est indépendante du fait de se faire rappeler suite à un dépot de CV, alors on s'attend aux valeurs suivantes :

```
# caucasien & pas de rappel
(2200+235) * (2200+2278) / sum(effectifs_obs)
```

```
## [1] 2239
```

```
# caucasien et rappel
(2200+235) * (235+157) / sum(effectifs_obs)
```

```
## [1] 196
```

```
# afro-américain & pas de rappel
(2278 + 157) * (2200+2278) / sum(effectifs_obs)
```

```
## [1] 2239
```

```
# afro-américain & rappel
(2278 + 157) * (235+157) / sum(effectifs_obs)
```

```
## [1] 196
```

On peut obtenir ces effectifs théoriques sans entrer à la main une seule valeur :

```
# Effectifs selon le caractère (rappelé ou non)
somme_ligne <- rowSums(effectifs_obs)
somme_ligne
```

```
##   no  yes
## 4478  392
```

```
# Effectifs selon les sous-populations (afam ou cauc)
somme_colonne <- colSums(effectifs_obs)
somme_colonne
```

```
## cauc afam
## 2435 2435
```

```r
# Nombre total d'observations
n <- sum(effectifs_obs)
n
```

```
## [1] 4870
```

Reste alors à calculer le produit extérieur (on multiplie chaque élément du vecteur `somme_ligne` par chaque élément du vecteur `somme_colonne`), puis à diviser par l'effectif total :

```r
effectifs_theo <- outer(somme_ligne, somme_colonne, "*")/n
effectifs_theo
```

```
##      cauc afam
## no   2239 2239
## yes   196  196
```

On peut de fait calculer la statistique de test :

```r
statstique <- sum((effectifs_obs - effectifs_theo)^2 / effectifs_theo)
statstique
```

```
## [1] 16.87905
```

Cette statistique est à comparer avec le quantile d'ordre α, $\chi^{2,\alpha}_{(p-1)(q-1)}$ que l'on peut lire dans une table du Khi-deux à $(p-1)(q-1)$ degrés de liberté… ou que l'on peut obtenir avec R ! La fonction `qchisq()` permet d'obtenir le quantile $\chi^{2,\alpha}_{(p-1)(q-1)}$ tel que la probabilité d'obtenir une valeur supérieure à ce quantile selon une Khi-deux à $(p-1)(q-1)$ degrés de liberté soit égale à $1-\alpha$.

```r
alpha = 5/100
p <- 2 ; q <- 2
qchisq(p = 1-alpha, df = (p-1)*(q-1))
```

```
## [1] 3.841459
```

La statistique observée étant largement supérieure au quantile obtenu, on rejette l'hypothèse nulle du test (avec un risque de rejetter l'hypothèse nulle à tort de 5%) d'indépendance de l'ethnicité et de la réponse à une candidature à Boston ou Chicago. Il semble bien qu'il y ait une discrimination à l'accès à l'emploi.

On peut également calculer la probabilité d'observer une valeur au moins aussi grande que celle que nous observons pour notre statistique (16.88) : c'est-à-dire, chercher à calculer la p-value associée au test. La fonction `pchisq()` nous donne la probailité d'observer une valeur inférieure ou égale à un quantile donné, pour un degré de liberté spécifique. Aussi, ce que nous recherchons est 1 moins cette probabilité :

```
1-pchisq(q = statstique, df = (p-1)*(q-1))
```

```
## [1] 3.983887e-05
```

Cette p-value est très faible. Grossièrement parlant, il y a une probabilité de $3,98 \times 10^{-5}$ d'observer une valeur de la statistique du test qui soit au moins aussi grande que 16.87, si l'hypothèse nulle du test est vraie.

Rassurez-vous, pour effectuer un test du Khi-deux, il n'est pas nécessaire de passer par toutes ces étapes avec R. Il existe évidemment une fonction qui vous permette d'effectuer le test en une seule ligne, à partir du moment où vous avez obtenu votre tableau croisé : `chisq.test()`. On retrouve bien les mêmes valeurs.

```
res <- chisq.test(effectifs_obs, correct = FALSE)
res
```

```
##
##  Pearson's Chi-squared test
##
## data:  effectifs_obs
## X-squared = 16.879, df = 1, p-value = 3.984e-05
```

```
res$statistic
```

```
## X-squared
##  16.87905
```

```
res$parameter
```

```
## df
##  1
```

```
res$p.value
```

```
## [1] 3.983887e-05
```

Note : Par défaut, l'argument `correct` de la fonction `chisq.test()` vaut `TRUE`. Dans ce cas, la correction de continuité de Yates est appliquée. La statistique de test est alors légèrment modifiée :

$$\chi^2 = \sum_{j=1}^{q} \sum_{i=1}^{p} \frac{\left(|n_{ij} - n_{ij}^e| - 0,5\right)^2}{n_{ij}^e}.$$

7.2.2.2 Statistiques par sous-groupes

Pour obtenir le même type d'informations que ce qui est offert par la fonction `table()`, nous pouvons effectuer dans un premier temps un regroupement du tableau de données avec la fonction `group_by()` que nous avons déjà utilisée dans le chapitre portant sur les tableaux de données. Ensuite, en appliquant la fonction `count()` au résultat, nous pouvons obtenir les fréquence pour chacun des sous-groupes.

```
resumes %>%
  group_by(call, ethnicity) %>%
  count()
```

```
## # A tibble: 4 x 3
## # Groups:   call, ethnicity [4]
##   call  ethnicity     n
##   <fct> <fct>     <int>
## 1 no    cauc       2200
## 2 no    afam       2278
## 3 yes   cauc        235
## 4 yes   afam        157
```

L'argument `name` de la fonction `count()` permet de spécifier le nom de la colonne dans laquelle les effectifs seront reportés :

```
resumes %>%
  group_by(call, ethnicity) %>%
  count(name = "Nombre d'observations")
```

```
## # A tibble: 4 x 3
## # Groups:   call, ethnicity [4]
##   call  ethnicity `Nombre d'observations`
##   <fct> <fct>                       <int>
```

```
## 1 no     cauc                          2200
## 2 no     afam                          2278
## 3 yes    cauc                           235
## 4 yes    afam                           157
```

On peut vérifier, à l'aide de la fonction `filter()` que le décompte correspond bien au nombre d'observations pour lesquelles la valeur dans la colonne `call` est `"no"` et pour lesquelles celles dans la colonne `ethnicity` vaut `"cauc"` :

```
resumes %>%
  filter(call == "no" & ethnicity == "cauc") %>%
  nrow()
```

```
## [1] 2200
```

Pour obtenir les fréquences marginales, il suffit d'effectuer un nouveau regroupement après l'appel à la fonction `count()` :

```
resumes %>%
  group_by(call, ethnicity) %>%
  count(name = "Nombre d'observations") %>%
  group_by(ethnicity) %>%
  mutate(prop = `Nombre d'observations` / sum(`Nombre d'observations`),
         prop = round(prop, 2))
```

```
## # A tibble: 4 x 4
## # Groups:   ethnicity [2]
##   call  ethnicity `Nombre d'observations`  prop
##   <fct> <fct>                        <int> <dbl>
## 1 no    cauc                          2200  0.9
## 2 no    afam                          2278  0.94
## 3 yes   cauc                           235  0.1
## 4 yes   afam                           157  0.06
```

Pour obtenir des résumés statistiques de variables numériques, on peut utiliser le combo `group_by()` et `summarise()` abordé dans le chapitre portant sur les tableaux de données :

```
resumes %>%
  group_by(gender, ethnicity) %>%
  summarise(mean_experience = mean(experience),
            sd_experience = sd(experience),
            q1_experience = quantile(experience, probs = .25))
```

```
## # A tibble: 4 x 5
## # Groups:   gender [2]
##   gender ethnicity mean_experience sd_experience q1_experience
##   <fct>  <fct>               <dbl>         <dbl>         <dbl>
## 1 male   cauc                 7.68          5.34             5
## 2 male   afam                 7.37          5.01             5
## 3 female cauc                 7.91          4.99             5
## 4 female afam                 7.96          5.00             5
```

Exercice

1. Calculez le nombre d'observations pour chaque valeur différente de la colonne `name` du tableau `resumes`.
2. Calculez les proportions de chaque valeur différente de la colonne `name` du tableau `resumes`.
3. Calculez, pour chaque valeur différente de la colonne `name`, la proportion de réponses positives (`call` valant `"yes"`{R}) et de réponses négatives (`call` valant `"no"`{R}).
4. Calculez le nombre d'emplois passés (colonne `jobs`) selon le nom et le genre (colonnes `name` et `gender`).

7.3 Résumés avec gtsummary : vers la communication des résultats

Les différentes méthodes abordées jusqu'à présent pour réaliser des résumés statistiques sont très pratiques pour réaliser une fouille grossière des données. Dans cette partie, nous allons explorer quelques possibilités offertes par un *package* nommé {gtsummary}, qui permet de produire des tableaux formatés pour être communiqués. Il existe de bien nombreux *packages* qui permettent de réaliser des tableaux prêts pour la communication, mais il me semble que {gtsummary} offre des fonctionnalités particulièrement utiles pour des économètres.

Dans un premier temps, si le *package* n'est pas installé :

```
install.packages(gtsummary)
```

Ensuite, chargons le *package* :

```
library(gtsummary)
```

La fonction au cœur du package se nomme `tbl_summary()`. Elle produit un tableau qui s'affiche dans l'onglet "Viewer". Le tableau qui est produit s'adapte en fonction du contenu des colonnes : une détection automatique du type des variables est effectuée. Lorsque les variables sont **discrètes**, la colonne descriptive indique le nombre d'observation et la proportion que chaque modalité représente. Lorsque les variables sont **continues**, la description est la médiane et l'écart interquartile.

```
resumes %>%
  tbl_summary()
```

Characteristic	**N = 4,870**
name	
Allison	232 (4.8%)
Anne	242 (5.0%)
Carrie	168 (3.4%)
Emily	227 (4.7%)
Jill	203 (4.2%)
Laurie	195 (4.0%)
Kristen	213 (4.4%)
Meredith	187 (3.8%)
Sarah	193 (4.0%)
Brad	63 (1.3%)
Brendan	65 (1.3%)
Geoffrey	59 (1.2%)
Greg	51 (1.0%)
Brett	59 (1.2%)
Jay	67 (1.4%)
Matthew	67 (1.4%)
Neil	76 (1.6%)
Todd	68 (1.4%)
Aisha	180 (3.7%)
Ebony	208 (4.3%)
Keisha	183 (3.8%)
Kenya	196 (4.0%)
Lakisha	200 (4.1%)

Latonya	230 (4.7%)
Latoya	226 (4.6%)
Tamika	256 (5.3%)
Tanisha	207 (4.3%)
Darnell	42 (0.9%)
Hakim	55 (1.1%)
Jamal	61 (1.3%)
Jermaine	52 (1.1%)
Kareem	64 (1.3%)
Leroy	64 (1.3%)
Rasheed	67 (1.4%)
Tremayne	69 (1.4%)
Tyrone	75 (1.5%)
gender	
male	1,124 (23%)
female	3,746 (77%)
ethnicity	
cauc	2,435 (50%)
afam	2,435 (50%)
call	392 (8.0%)
city	
boston	2,166 (44%)
chicago	2,704 (56%)
jobs	
1	110 (2.3%)
2	704 (14%)
3	1,429 (29%)
4	1,611 (33%)
5	533 (11%)
6	464 (9.5%)
7	19 (0.4%)
experience	6 (5, 9)

[1] n (%) ; Median (IQR)

Parmi les arguments de la fonction `tbl_summary()`, figure `by=` qui permet de réaliser des sous-groupes dans les données avant de reporter les statistiques descriptives. Les différents groupes sont alors créés en fonction des valeurs distinctes rencontrées dans le colonne dont

le nom est indiqué à l'argument `by=`. Par exemple, pour obtenir des statistiques descriptives du jeu de données `resumes` selon l'origine ethnique (colonne `ethnicity`), on pourra écrire :

```
resumes %>%
  tbl_summary(by = ethnicity)
```

Characteristic	**cauc**, N = 2,435	**afam**, N = 2,435
name		
Allison	232 (9.5%)	0 (0%)
Anne	242 (9.9%)	0 (0%)
Carrie	168 (6.9%)	0 (0%)
Emily	227 (9.3%)	0 (0%)
Jill	203 (8.3%)	0 (0%)
Laurie	195 (8.0%)	0 (0%)
Kristen	213 (8.7%)	0 (0%)
Meredith	187 (7.7%)	0 (0%)
Sarah	193 (7.9%)	0 (0%)
Brad	63 (2.6%)	0 (0%)
Brendan	65 (2.7%)	0 (0%)
Geoffrey	59 (2.4%)	0 (0%)
Greg	51 (2.1%)	0 (0%)
Brett	59 (2.4%)	0 (0%)
Jay	67 (2.8%)	0 (0%)
Matthew	67 (2.8%)	0 (0%)
Neil	76 (3.1%)	0 (0%)
Todd	68 (2.8%)	0 (0%)
Aisha	0 (0%)	180 (7.4%)
Ebony	0 (0%)	208 (8.5%)
Keisha	0 (0%)	183 (7.5%)
Kenya	0 (0%)	196 (8.0%)
Lakisha	0 (0%)	200 (8.2%)
Latonya	0 (0%)	230 (9.4%)
Latoya	0 (0%)	226 (9.3%)
Tamika	0 (0%)	256 (11%)
Tanisha	0 (0%)	207 (8.5%)
Darnell	0 (0%)	42 (1.7%)
Hakim	0 (0%)	55 (2.3%)

Jamal	0 (0%)	61 (2.5%)
Jermaine	0 (0%)	52 (2.1%)
Kareem	0 (0%)	64 (2.6%)
Leroy	0 (0%)	64 (2.6%)
Rasheed	0 (0%)	67 (2.8%)
Tremayne	0 (0%)	69 (2.8%)
Tyrone	0 (0%)	75 (3.1%)
gender		
male	575 (24%)	549 (23%)
female	1,860 (76%)	1,886 (77%)
call	235 (9.7%)	157 (6.4%)
city		
boston	1,083 (44%)	1,083 (44%)
chicago	1,352 (56%)	1,352 (56%)
jobs		
1	54 (2.2%)	56 (2.3%)
2	347 (14%)	357 (15%)
3	726 (30%)	703 (29%)
4	800 (33%)	811 (33%)
5	258 (11%)	275 (11%)
6	243 (10.0%)	221 (9.1%)
7	7 (0.3%)	12 (0.5%)
experience	6 (5, 9)	6 (5, 9)

[1] n (%) ; Median (IQR)

Pour la suite de cette section, nous allons éviter d'afficher les informations pour la colonne `name`, pour gagner un peu en lisibilité. Pour ce faire, il suffit d'exclure du tableau cette colonne avant d'appliquer la fonction `tbl_summary` :

```
resumes %>%
  select(-name) %>%
  tbl_summary(by = ethnicity)
```

Characteristic	**cauc**, N = 2,435	**afam**, N = 2,435
gender		
male	575 (24%)	549 (23%)
female	1,860 (76%)	1,886 (77%)

call	235 (9.7%)	157 (6.4%)
city		
boston	1,083 (44%)	1,083 (44%)
chicago	1,352 (56%)	1,352 (56%)
jobs		
1	54 (2.2%)	56 (2.3%)
2	347 (14%)	357 (15%)
3	726 (30%)	703 (29%)
4	800 (33%)	811 (33%)
5	258 (11%)	275 (11%)
6	243 (10.0%)	221 (9.1%)
7	7 (0.3%)	12 (0.5%)
experience	6 (5, 9)	6 (5, 9)

[1] n (%) ; Median (IQR)

Pour réaliser un **test d'indépendance** entre chaque variables et la variable de regroupement, on peut appliquer la fonction `add_p()` sur le tableau gtsummary. Une colonne est ajoutée à droite avec les p-values.

```
resumes %>%
  select(-name) %>%
  tbl_summary(by = ethnicity) %>%
  add_p()
```

Characteristic	**cauc**, N = 2,435	**afam**, N = 2,435	**p-value**
gender			0.4
male	575 (24%)	549 (23%)	
female	1,860 (76%)	1,886 (77%)	
call	235 (9.7%)	157 (6.4%)	<0.001
city			>0.9
boston	1,083 (44%)	1,083 (44%)	
chicago	1,352 (56%)	1,352 (56%)	
jobs			0.7
1	54 (2.2%)	56 (2.3%)	
2	347 (14%)	357 (15%)	
3	726 (30%)	703 (29%)	
4	800 (33%)	811 (33%)	

5	258 (11%)	275 (11%)	
6	243 (10.0%)	221 (9.1%)	
7	7 (0.3%)	12 (0.5%)	
experience	6 (5, 9)	6 (5, 9)	0.9

[1] n (%) ; Median (IQR)
[2] Pearson's Chi-squared test ; Wilcoxon rank sum test

Avoir des statistiques pour chacun des sous groupes est très utile, mais il est parfois également intéressant d'afficher les statistiques pour l'échantillon total. Pour ce faire, on peut utiliser la fonction `add_overall()`, qui ajoutera une colonne à gauche. En fournissant à l'argument `col_label=` une chaîne de caractère, le nom de la colonne ainsi ajoutée peut être modifié (par défaut, la valeur `"Overall"` est utilisée) :

```
resumes %>%
  select(-name) %>%
  tbl_summary(by = ethnicity) %>%
  add_p() %>%
  add_overall(col_label = "Ech. total")
```

Characteristic	Ech. total	**cauc**, N = 2,435	**afam**, N = 2,435	**p-value**
gender				0.4
male	1,124 (23%)	575 (24%)	549 (23%)	
female	3,746 (77%)	1,860 (76%)	1,886 (77%)	
call	392 (8.0%)	235 (9.7%)	157 (6.4%)	<0.001
city				>0.9
boston	2,166 (44%)	1,083 (44%)	1,083 (44%)	
chicago	2,704 (56%)	1,352 (56%)	1,352 (56%)	
jobs				0.7
1	110 (2.3%)	54 (2.2%)	56 (2.3%)	
2	704 (14%)	347 (14%)	357 (15%)	
3	1,429 (29%)	726 (30%)	703 (29%)	
4	1,611 (33%)	800 (33%)	811 (33%)	
5	533 (11%)	258 (11%)	275 (11%)	
6	464 (9.5%)	243 (10.0%)	221 (9.1%)	
7	19 (0.4%)	7 (0.3%)	12 (0.5%)	
experience	6 (5, 9)	6 (5, 9)	6 (5, 9)	0.9

[1] n (%) ; Median (IQR)

[2] Pearson's Chi-squared test ; Wilcoxon rank sum test

Sur la première colonne, on peut lire en cartouche : **Characteristic**. Pour changer cela, on peut faire appel à la fonction `modify_header()`, en indiquant la valeur souhaitée à l'argument `label=`. On peut fournir une valeur rédigée en markdown. Dans l'exemple ci-dessous, pour que la valeur **Variable** soit affichée, on écrit ainsi :

```
resumes %>%
  select(-name) %>%
  tbl_summary(by = ethnicity) %>%
  add_p() %>%
  add_overall(col_label = "Ech. total") %>%
  modify_header(label = "**Variable**")
```

Variable	Ech. total	**cauc**, N = 2,435	**afam**, N = 2,435	**p-value**
gender				0.4
male	1,124 (23%)	575 (24%)	549 (23%)	
female	3,746 (77%)	1,860 (76%)	1,886 (77%)	
call	392 (8.0%)	235 (9.7%)	157 (6.4%)	<0.001
city				>0.9
boston	2,166 (44%)	1,083 (44%)	1,083 (44%)	
chicago	2,704 (56%)	1,352 (56%)	1,352 (56%)	
jobs				0.7
1	110 (2.3%)	54 (2.2%)	56 (2.3%)	
2	704 (14%)	347 (14%)	357 (15%)	
3	1,429 (29%)	726 (30%)	703 (29%)	
4	1,611 (33%)	800 (33%)	811 (33%)	
5	533 (11%)	258 (11%)	275 (11%)	
6	464 (9.5%)	243 (10.0%)	221 (9.1%)	
7	19 (0.4%)	7 (0.3%)	12 (0.5%)	
experience	6 (5, 9)	6 (5, 9)	6 (5, 9)	0.9

[1] n (%) ; Median (IQR)

[2] Pearson's Chi-squared test ; Wilcoxon rank sum test

Pour ajouter une étiquette permettant de regrouper visuellement les sous-groupes, le *package* {gtsummary} propose la fonction `modify_spanning_header()`. Dans notre exemple, nous avons deux sous-groupes formés selon les modalités de la variable `ethnicity`. Aussi,

nous pourrons indiquer que nous souhaitons que les colonnes `stat_1` et `stat_2` (colonne contenant les statistiques descriptives pour le premier et le second sous-groupes) disposent d'une en-tête :

```
resumes %>%
  select(-name) %>%
  tbl_summary(by = ethnicity) %>%
  add_p() %>%
  add_overall(col_label = "Ech. total") %>%
  modify_header(label = "**Variable**") %>%
  modify_spanning_header(c("stat_1", "stat_2") ~ "**Consonnance du nom**")
```

		Consonnance du nom		
Variable	Ech. total	**cauc**, N = 2,435	**afam**, N = 2,435	**p-value**
gender				0.4
male	1,124 (23%)	575 (24%)	549 (23%)	
female	3,746 (77%)	1,860 (76%)	1,886 (77%)	
call	392 (8.0%)	235 (9.7%)	157 (6.4%)	<0.001
city				>0.9
boston	2,166 (44%)	1,083 (44%)	1,083 (44%)	
chicago	2,704 (56%)	1,352 (56%)	1,352 (56%)	
jobs				0.7
1	110 (2.3%)	54 (2.2%)	56 (2.3%)	
2	704 (14%)	347 (14%)	357 (15%)	
3	1,429 (29%)	726 (30%)	703 (29%)	
4	1,611 (33%)	800 (33%)	811 (33%)	
5	533 (11%)	258 (11%)	275 (11%)	
6	464 (9.5%)	243 (10.0%)	221 (9.1%)	
7	19 (0.4%)	7 (0.3%)	12 (0.5%)	
experience	6 (5, 9)	6 (5, 9)	6 (5, 9)	0.9

[1] n (%) ; Median (IQR)

[2] Pearson's Chi-squared test ; Wilcoxon rank sum test

Lorsque les colonnes du tableau de données fourni à la fonction `tbl_summary()` contient des étiquettes, ces dernières sont automatiquement utilisées dans le tableau retourné à la place des noms de variables. Pour ajouter des étiquettes à des colonnes, nous pouvons utiliser le *package* {labelled}, qu'il faut au préalable installer.

```
install.packages("labelled")
```

Une fois le *package* installé, il est nécessaire de le charger :

```
library(labelled)
```

La fonction `set_variable_labels()` de {labelled} permet d'ajouter les étiquettes aux variables. Il suffit de donner le nom des variables et, après un symbole égal (`=`), leur étiquette correspondante.

```
resumes <-
  resumes %>%
  labelled::set_variable_labels(
    name = "Prénom",
    gender = "Genre",
    ethnicity = "Ethnicité",
    call = "Entretien",
    city = "Ville",
    jobs = "Nombre d'emplois listés",
    experience = "Nombre d'années d'expérience"
  )
```

Si on évalue à nouveau notre dernier code permettant d'afficher un tableau de statistiques descriptives pour notre jeu de données, on note que les noms de variables (`gender`, `call`, `city`, etc.) ne s'affichent plus et qu'à la place, l'étiquette est utilisée :

```
resumes %>%
  select(-name) %>%
  tbl_summary(by = ethnicity) %>%
  add_p() %>%
  add_overall(col_label = "Ech. total") %>%
  modify_header(label = "**Variable**") %>%
  modify_spanning_header(c("stat_1", "stat_2") ~ "**Consonnance du nom**")
```

Note
Il est possible de fournir à l'argument `label=` de la fonction `tbl_summary()` les étiquettes à utiliser pour chaque variable, plutôt que de passer par la définition des étiquettes via l'ajout d'un attribut aux colonnes avec la fonction

		Consonnance du nom		
Variable	Ech. total	**cauc**, N = 2,435	**afam**, N = 2,435	**p-value**
Genre				0.4
male	1,124 (23%)	575 (24%)	549 (23%)	
female	3,746 (77%)	1,860 (76%)	1,886 (77%)	
Entretien	392 (8.0%)	235 (9.7%)	157 (6.4%)	<0.001
Ville				>0.9
boston	2,166 (44%)	1,083 (44%)	1,083 (44%)	
chicago	2,704 (56%)	1,352 (56%)	1,352 (56%)	
Nombre d'emplois listés				0.7
1	110 (2.3%)	54 (2.2%)	56 (2.3%)	
2	704 (14%)	347 (14%)	357 (15%)	
3	1,429 (29%)	726 (30%)	703 (29%)	
4	1,611 (33%)	800 (33%)	811 (33%)	
5	533 (11%)	258 (11%)	275 (11%)	
6	464 (9.5%)	243 (10.0%)	221 (9.1%)	
7	19 (0.4%)	7 (0.3%)	12 (0.5%)	
Nombre d'années d'expérience	6 (5, 9)	6 (5, 9)	6 (5, 9)	0.9

[1] n (%) ; Median (IQR)

[2] Pearson's Chi-squared test ; Wilcoxon rank sum test

`set_variable_labels()`.

Le nom des différents sous-groupes constitués à l'aide de la variable dont le nom est fourni à l'argument `by=` de la fonction `tbl_summary()` sont définis selon les valeurs présentes dans le tableau. Pour recoder les valeurs d'une variable factorielle, nous pouvons utiliser la fonction `recode()` de {dplyr}. Attention, la syntaxe est assez peu heureuse, dans la mesure où elle est à contre courant de ce que l'on peut trouver avec la fonction `rename()` (fonction permettant de renommer les colonnes d'un tableau de données) : on écrit l'ancien nomn, puis après un symbole égal (`=`) le nouveau nom, dans une chaîne de caractères.

```
resumes <-
  resumes %>%
  mutate(
    gender = recode(gender, female = "femme", male = "homme"),
    ethnicity = recode(ethnicity,
                       cauc = "Caucasien", afam = "Afro-americain"),
    call = recode(call, no = "Non", yes = "Oui")
  )

resumes
```

```
## # A tibble: 4,870 x 7
##    name    gender ethnicity      call  city     jobs experience
##    <fct>   <fct>  <fct>          <fct> <fct>   <int>      <int>
##  1 Allison femme  Caucasien      Non   chicago     2          6
##  2 Kristen femme  Caucasien      Non   chicago     3          6
##  3 Lakisha femme  Afro-americain Non   chicago     1          6
##  4 Latonya femme  Afro-americain Non   chicago     4          6
##  5 Carrie  femme  Caucasien      Non   chicago     3         22
##  6 Jay     homme  Caucasien      Non   chicago     2          6
##  7 Jill    femme  Caucasien      Non   chicago     2          5
##  8 Kenya   femme  Afro-americain Non   chicago     4         21
##  9 Latonya femme  Afro-americain Non   chicago     3          3
## 10 Tyrone  homme  Afro-americain Non   chicago     2          6
## # ... with 4,860 more rows
```

Une fois les valeurs recodées à l'intérieur du tableau, celui qui est produit par la fonction `tbl_summary()` est également impacté :

```
resumes %>%
  select(-name) %>%
  tbl_summary(by = ethnicity) %>%
```

Variable	Ech. total	Consonnance du nom		p-value
		Caucasien, N = 2,435	**Afro-americain**, N = 2,435	
Genre				0.4
homme	1,124 (23%)	575 (24%)	549 (23%)	
femme	3,746 (77%)	1,860 (76%)	1,886 (77%)	
Entretien				<0.001
Non	4,478 (92%)	2,200 (90%)	2,278 (94%)	
Oui	392 (8.0%)	235 (9.7%)	157 (6.4%)	
Ville				>0.9
boston	2,166 (44%)	1,083 (44%)	1,083 (44%)	
chicago	2,704 (56%)	1,352 (56%)	1,352 (56%)	
Nombre d'emplois listés				0.7
1	110 (2.3%)	54 (2.2%)	56 (2.3%)	
2	704 (14%)	347 (14%)	357 (15%)	
3	1,429 (29%)	726 (30%)	703 (29%)	
4	1,611 (33%)	800 (33%)	811 (33%)	
5	533 (11%)	258 (11%)	275 (11%)	
6	464 (9.5%)	243 (10.0%)	221 (9.1%)	
7	19 (0.4%)	7 (0.3%)	12 (0.5%)	
Nombre d'années d'expérience	6 (5, 9)	6 (5, 9)	6 (5, 9)	0.9

[1] n (%) ; Median (IQR)

[2] Pearson's Chi-squared test ; Wilcoxon rank sum test

```
  add_p() %>%
  add_overall(col_label = "Ech. total") %>%
  modify_header(label = "**Variable**") %>%
  modify_spanning_header(c("stat_1", "stat_2") ~ "**Consonnance du nom**")
```

Pour décider qulles variables inclure dans le tableau de statistiques descriptives, on peut utiliser la fonction `select()` de {dplyr} sur le tableau de l'on fournit à la fonction `tbl_summary()`, ou bien on peut préciser quelles colonnes inclure à l'aide de l'argument `include=` de `tbl_summary()`.

```
resumes %>%
  tbl_summary(
    include = c("gender", "ethnicity", "call", "jobs", "experience"),
    by = ethnicity) %>%
  add_p() %>%
  add_overall(col_label = "Ech. total") %>%
  modify_header(label = "**Variable**") %>%
  modify_spanning_header(c("stat_1", "stat_2") ~ "**Consonnance du nom**")
```

Variable	Ech. total	Consonnance du nom		p-value
		Caucasien, N = 2,435	**Afro-americain**, N = 2,435	
Genre				0.4
homme	1,124 (23%)	575 (24%)	549 (23%)	
femme	3,746 (77%)	1,860 (76%)	1,886 (77%)	
Entretien				<0.001
Non	4,478 (92%)	2,200 (90%)	2,278 (94%)	
Oui	392 (8.0%)	235 (9.7%)	157 (6.4%)	
Nombre d'emplois listés				0.7
1	110 (2.3%)	54 (2.2%)	56 (2.3%)	
2	704 (14%)	347 (14%)	357 (15%)	
3	1,429 (29%)	726 (30%)	703 (29%)	
4	1,611 (33%)	800 (33%)	811 (33%)	
5	533 (11%)	258 (11%)	275 (11%)	
6	464 (9.5%)	243 (10.0%)	221 (9.1%)	
7	19 (0.4%)	7 (0.3%)	12 (0.5%)	
Nombre d'années d'expérience	6 (5, 9)	6 (5, 9)	6 (5, 9)	0.9

[1] n (%) ; Median (IQR)

[2] Pearson's Chi-squared test ; Wilcoxon rank sum test

L'argument `statistic=` de la fonction `tbl_summary()` permet de changer les statistiques reportées. On fournit une liste dans laquelle on indique des formules spécifiant les types de statistiques descriptives à afficher pour les variables. Nous avons vu plus haut que les valeurs par défaut qui sont utilisées sont la médiane, suivi, entre parenthèses de l'écart interquartile pour les variables continues, et le nombre d'observations suivi de la proportion entre parenthèses, soit :

```
list(all_continuous() ~ "{median} ({p25}, {p75})",
     all_categorical() ~ "{n} ({p}%)")
```

Si on désire afficher la moyenne (*mean*) suivie de l'écart-type (*standard deviation*) pour les variables continues, et le nombre d'observations et la proportion pour les variables discrètes, on écrira :

```
resumes %>%
  tbl_summary(
    include = c("gender", "ethnicity", "call", "jobs", "experience"),
    by = ethnicity,
    statistic = list(
      all_continuous() ~ "{mean} ({sd})",
      all_categorical() ~ "{n} ({p}%)"
      )
```

Variable	Ech. total	Consonnance du nom		p-value
		Caucasien, N = 2,435	**Afro-americain**, N = 2,435	
Genre				0.4
homme	1,124 (23%)	575 (24%)	549 (23%)	
femme	3,746 (77%)	1,860 (76%)	1,886 (77%)	
Entretien				<0.001
Non	4,478 (92%)	2,200 (90%)	2,278 (94%)	
Oui	392 (8.0%)	235 (9.7%)	157 (6.4%)	
Nombre d'emplois listés				0.7
1	110 (2.3%)	54 (2.2%)	56 (2.3%)	
2	704 (14%)	347 (14%)	357 (15%)	
3	1,429 (29%)	726 (30%)	703 (29%)	
4	1,611 (33%)	800 (33%)	811 (33%)	
5	533 (11%)	258 (11%)	275 (11%)	
6	464 (9.5%)	243 (10.0%)	221 (9.1%)	
7	19 (0.4%)	7 (0.3%)	12 (0.5%)	
Nombre d'années d'expérience	8 (5)	8 (5)	8 (5)	0.9

[1] n (%) ; Mean (SD)

[2] Pearson's Chi-squared test ; Wilcoxon rank sum test

```
  ) %>%
 add_p() %>%
 add_overall(col_label = "Ech. total") %>%
 modify_header(label = "**Variable**") %>%
 modify_spanning_header(c("stat_1", "stat_2") ~ "**Consonnance du nom**")
```

Pour afficher des résumés statistiques sur plusieurs lignes pour les variables continues, il faut le préciser, à l'aide de l'argument `type=` de la fonction `tbl_summary()`. En écrivant `type = all_continuous() ~ "continuous2"`, toutes les variables continues disposeront de deux lignes de résumés statistiques. Reste alors à indiquer ce qu'indiquera chaque ligne. Pour cela, ils nous suffit de fournir un vecteur de longueur 2 à l'élément de la liste fournie au pramètre `statistic=` qui se charge de définir les statistiques à retourner pour les variables continues. Dans l'exemple qui suit, nous reporterons la moyenne et l'écart-type sur la première ligne, et la médiane accompagnée de l'écart interquartile sur la seconde.

```
resumes %>%
  tbl_summary(
    include = c("gender", "ethnicity", "call", "jobs", "experience"),
    by = ethnicity,
    type = all_continuous() ~ "continuous2",
    statistic = list(
```

		Consonnance du nom		
Variable	Ech. total	**Caucasien**, N = 2,435	**Afro-americain**, N = 2,435	**p-value**
Genre				0.4
homme	1,124 (23%)	575 (24%)	549 (23%)	
femme	3,746 (77%)	1,860 (76%)	1,886 (77%)	
Entretien				<0.001
Non	4,478 (92%)	2,200 (90%)	2,278 (94%)	
Oui	392 (8.0%)	235 (9.7%)	157 (6.4%)	
Nombre d'emplois listés				0.7
1	110 (2.3%)	54 (2.2%)	56 (2.3%)	
2	704 (14%)	347 (14%)	357 (15%)	
3	1,429 (29%)	726 (30%)	703 (29%)	
4	1,611 (33%)	800 (33%)	811 (33%)	
5	533 (11%)	258 (11%)	275 (11%)	
6	464 (9.5%)	243 (10.0%)	221 (9.1%)	
7	19 (0.4%)	7 (0.3%)	12 (0.5%)	
Nombre d'années d'expérience				0.9
Mean (SD)	8 (5)	8 (5)	8 (5)	
Median (IQR)	6 (5, 9)	6 (5, 9)	6 (5, 9)	

[1] n (%)

[2] Pearson's Chi-squared test ; Wilcoxon rank sum test

```
      all_continuous() ~ c("{mean} ({sd})", "{median} ({p25}, {p75})"),
      all_categorical() ~ "{n} ({p}%)"
      )
    ) %>%
  add_p() %>%
  add_overall(col_label = "Ech. total") %>%
  modify_header(label = "**Variable**") %>%
  modify_spanning_header(c("stat_1", "stat_2") ~ "**Consonnance du nom**")
```

Avec la fonction `add_stat_label()`, nous pouvons préciser, en créant une liste contenant des formules, les indications à faire figurer dans le tableau pour les étiquettes des statistiques qui s'afficheront. Dans l'exemple qui suit, nous demandons à ce que sur la première ligne figure la valeur `Moyenne (Ecart-type)` et sur la deuxième `Médiane (IQR)` lorsque la variables est continue, et `n (%)` lorsqu'elle est discrète.

```
resumes %>%
  tbl_summary(
    include = c("gender", "ethnicity", "call", "jobs", "experience"),
    by = ethnicity,
```

		Consonnance du nom		
Variable	Ech. total	**Caucasien**, N = 2,435	**Afro-americain**, N = 2,435	**p-value**
Genre, n (%)				0.4
homme	1,124 (23%)	575 (24%)	549 (23%)	
femme	3,746 (77%)	1,860 (76%)	1,886 (77%)	
Entretien, n (%)				<0.001
Non	4,478 (92%)	2,200 (90%)	2,278 (94%)	
Oui	392 (8.0%)	235 (9.7%)	157 (6.4%)	
Nombre d'emplois listés, n (%)				0.7
1	110 (2.3%)	54 (2.2%)	56 (2.3%)	
2	704 (14%)	347 (14%)	357 (15%)	
3	1,429 (29%)	726 (30%)	703 (29%)	
4	1,611 (33%)	800 (33%)	811 (33%)	
5	533 (11%)	258 (11%)	275 (11%)	
6	464 (9.5%)	243 (10.0%)	221 (9.1%)	
7	19 (0.4%)	7 (0.3%)	12 (0.5%)	
Nombre d'années d'expérience				0.9
Moyenne (Ecart-type)	8 (5)	8 (5)	8 (5)	
Médiane (IQR)	6 (5, 9)	6 (5, 9)	6 (5, 9)	

[1] Pearson's Chi-squared test ; Wilcoxon rank sum test

```
    type = all_continuous() ~ "continuous2",
    statistic = list(
      all_continuous() ~ c("{mean} ({sd})", "{median} ({p25}, {p75})"),
      all_categorical() ~ "{n} ({p}%)"
      )
  ) %>%
 add_p() %>%
 add_overall(col_label = "Ech. total") %>%
 modify_header(label = "**Variable**") %>%
 modify_spanning_header(c("stat_1", "stat_2") ~ "**Consonnance du nom**") %>%
 add_stat_label(
   label = list(
     all_continuous() ~ c("Moyenne (Ecart-type)", "Médiane (IQR)"),
     all_categorical() ~ "n (%)"
   )
 )
```

Dans l'ensemble des tableaux que nous avons réalisés avec la fonction `tbl_summary()`, la variable `jobs` qui indique le nombre de précédents emplois occupés a été considérée comme étant discrète. Si nous souhaitons que la fonction considère cette colonne comme étant numérique, il faut l'indiquer à travers l'argument `type`, en donnant comme élément de liste

		Consonnance du nom		
Variable	Ech. total	**Caucasien**, N = 2,435	**Afro-americain**, N = 2,435	**p-value**
Genre, n (%)				0.4
homme	1,124 (23%)	575 (24%)	549 (23%)	
femme	3,746 (77%)	1,860 (76%)	1,886 (77%)	
Entretien, n (%)				<0.001
Non	4,478 (92%)	2,200 (90%)	2,278 (94%)	
Oui	392 (8.0%)	235 (9.7%)	157 (6.4%)	
Nombre d'emplois listés				>0.9
Moyenne (Ecart-type)	4 (1)	4 (1)	4 (1)	
Médiane (IQR)	4 (3, 4)	4 (3, 4)	4 (3, 4)	
Nombre d'années d'expérience				0.9
Moyenne (Ecart-type)	8 (5)	8 (5)	8 (5)	
Médiane (IQR)	6 (5, 9)	6 (5, 9)	6 (5, 9)	

[1] Pearson's Chi-squared test ; Wilcoxon rank sum test

une formule spécifique pour cette colonne :

```
resumes %>%
  tbl_summary(
    include = c("gender", "ethnicity", "call", "jobs", "experience"),
    by = ethnicity,
    type = list(all_continuous() ~ "continuous2",
                jobs ~ "continuous2"),
    statistic = list(
      all_continuous() ~ c("{mean} ({sd})", "{median} ({p25}, {p75})"),
      all_categorical() ~ "{n} ({p}%)"
      )
    ) %>%
  add_p() %>%
  add_overall(col_label = "Ech. total") %>%
  modify_header(label = "**Variable**") %>%
  modify_spanning_header(c("stat_1", "stat_2") ~ "**Consonnance du nom**") %>%
  add_stat_label(
    label = list(
      all_continuous() ~ c("Moyenne (Ecart-type)", "Médiane (IQR)"),
      all_categorical() ~ "n (%)"
    )
  )
```

Le contrôle du nombre de chiffres après la virgule s'effectue via l'argument `digits`. Encore une fois, une liste contenant des formules est donnée. Ici, demandons d'afficher 3 décimales

		Consonnance du nom		
Variable	Ech. total	**Caucasien**, N = 2,435	**Afro-americain**, N = 2,435	**p-value**
Genre, n (%)				0.4
homme	1,124 (23%)	575 (24%)	549 (23%)	
femme	3,746 (77%)	1,860 (76%)	1,886 (77%)	
Entretien, n (%)				<0.001
Non	4,478 (92%)	2,200 (90%)	2,278 (94%)	
Oui	392 (8.0%)	235 (9.7%)	157 (6.4%)	
Nombre d'emplois listés				>0.9
Moyenne (Ecart-type)	3.661 (1.219)	3.664 (1.219)	3.658 (1.219)	
Médiane (IQR)	4.000 (3.000, 4.000)	4.000 (3.000, 4.000)	4.000 (3.000, 4.000)	
Nombre d'années d'expérience				0.9
Moyenne (Ecart-type)	7.843 (5.045)	7.856 (5.079)	7.830 (5.011)	
Médiane (IQR)	6.000 (5.000, 9.000)	6.000 (5.000, 9.000)	6.000 (5.000, 9.000)	

[1] Pearson's Chi-squared test ; Wilcoxon rank sum test

après la virgule pour chaque statistique des variables continues :

```
resumes %>%
  tbl_summary(
    include = c("gender", "ethnicity", "call", "jobs", "experience"),
    by = ethnicity,
    type = list(all_continuous() ~ "continuous2",
                jobs ~ "continuous2"),
    statistic = list(
      all_continuous() ~ c("{mean} ({sd})", "{median} ({p25}, {p75})"),
      all_categorical() ~ "{n} ({p}%)"
      ),
    digits = list(all_continuous() ~ 3)
    ) %>%
  add_p() %>%
  add_overall(col_label = "Ech. total") %>%
  modify_header(label = "**Variable**") %>%
  modify_spanning_header(c("stat_1", "stat_2") ~ "**Consonnance du nom**") %>%
  add_stat_label(
    label = list(
      all_continuous() ~ c("Moyenne (Ecart-type)", "Médiane (IQR)"),
      all_categorical() ~ "n (%)"
    )
  )
```

Lorsque de multiples statistiques sont retournées pour une variable, on fournit un vecteur de valeurs à la formule si on désire que le nombre de décimales soit différent d'une statistique

Variable	Ech. total	Consonnance du nom		p-value
		Caucasien, N = 2,435	**Afro-americain**, N = 2,435	
Genre, n (%)				0.4
homme	1,124 (23%)	575 (24%)	549 (23%)	
femme	3,746 (77%)	1,860 (76%)	1,886 (77%)	
Entretien, n (%)				<0.001
Non	4,478 (92%)	2,200 (90%)	2,278 (94%)	
Oui	392 (8.0%)	235 (9.7%)	157 (6.4%)	
Nombre d'emplois listés				>0.9
Moyenne (Ecart-type)	3.7 (1.219)	3.7 (1.219)	3.7 (1.219)	
Médiane (IQR)	4.000 (3.0, 4.0)	4.000 (3.0, 4.0)	4.000 (3.0, 4.0)	
Nombre d'années d'expérience				0.9
Moyenne (Ecart-type)	7.8 (5.045)	7.9 (5.079)	7.8 (5.011)	
Médiane (IQR)	6.000 (5.0, 9.0)	6.000 (5.0, 9.0)	6.000 (5.0, 9.0)	

[1] Pearson's Chi-squared test ; Wilcoxon rank sum test

à l'autre. La longueur de ce vecteur doit correspondre au nombre de statistiques demandées via l'argument `statistic=`. Ici, demandons d'afficher 1 décimale pour la moyenne, 3 pour l'écart-type et la médiane, puis 1 pour les premiers et troisièmes quartiles.

```
resumes %>%
  tbl_summary(
    include = c("gender", "ethnicity", "call", "jobs", "experience"),
    by = ethnicity,
    type = list(all_continuous() ~ "continuous2",
                jobs ~ "continuous2"),
    statistic = list(
      all_continuous() ~ c("{mean} ({sd})", "{median} ({p25}, {p75})"),
      all_categorical() ~ "{n} ({p}%)"
      ),
    digits = list(all_continuous() ~ c(1,3, 3,1,1))
    ) %>%
  add_p() %>%
  add_overall(col_label = "Ech. total") %>%
  modify_header(label = "**Variable**") %>%
  modify_spanning_header(c("stat_1", "stat_2") ~ "**Consonnance du nom**") %>%
  add_stat_label(
    label = list(
      all_continuous() ~ c("Moyenne (Ecart-type)", "Médiane (IQR)"),
      all_categorical() ~ "n (%)"
    )
  )
```

Il est également possible de définir le nombre de décimales spécifiquement pour une ou plusieurs variables, en ajoutant une ou plusieurs formules à la liste fournie à l'argument `digits=` de la fonction `tbl_summary()`.

```
resumes %>%
  tbl_summary(
    include = c("gender", "ethnicity", "call", "jobs", "experience"),
    by = ethnicity,
    type = list(all_continuous() ~ "continuous2",
                jobs ~ "continuous2"),
    statistic = list(
      all_continuous() ~ c("{mean} ({sd})", "{median} ({p25}, {p75})"),
      all_categorical() ~ "{n} ({p}%)"
      ),
    digits = list(all_continuous() ~ c(1,3, 3,1,1),
                  jobs ~ c(0, 0, 0, 0, 0))
    ) %>%
  add_p() %>%
  add_overall(col_label = "Ech. total") %>%
  modify_header(label = "**Variable**") %>%
  modify_spanning_header(c("stat_1", "stat_2") ~ "**Consonnance du nom**") %>%
  add_stat_label(
    label = list(
      all_continuous() ~ c("Moyenne (Ecart-type)", "Médiane (IQR)"),
      all_categorical() ~ "n (%)"
    )
  )
```

Les tableaux produits utilisent par défaut des normes anglo-saxonnes. Pour utiliser des normes françaises, on peut se servir de la fonction `theme_gtsummary_language()` et indiquer la valeur `"fr"` à l'argument `language=` :

```
theme_gtsummary_language(language = "fr")
```

```
resumes %>%
  tbl_summary(
    include = c("gender", "ethnicity", "call", "jobs", "experience"),
    by = ethnicity,
    type = all_continuous() ~ "continuous2",
    statistic = list(
```

Variable	Ech. total	Consonnance du nom		p-value
		Caucasien, N = 2,435	**Afro-americain**, N = 2,435	
Genre, n (%)				0.4
homme	1,124 (23%)	575 (24%)	549 (23%)	
femme	3,746 (77%)	1,860 (76%)	1,886 (77%)	
Entretien, n (%)				<0.001
Non	4,478 (92%)	2,200 (90%)	2,278 (94%)	
Oui	392 (8.0%)	235 (9.7%)	157 (6.4%)	
Nombre d'emplois listés				>0.9
Moyenne (Ecart-type)	4 (1)	4 (1)	4 (1)	
Médiane (IQR)	4 (3, 4)	4 (3, 4)	4 (3, 4)	
Nombre d'années d'expérience				0.9
Moyenne (Ecart-type)	7.8 (5.045)	7.9 (5.079)	7.8 (5.011)	
Médiane (IQR)	6.000 (5.0, 9.0)	6.000 (5.0, 9.0)	6.000 (5.0, 9.0)	

[1] Pearson's Chi-squared test ; Wilcoxon rank sum test

```
    all_continuous() ~ c("{mean} ({sd})", "{median} ({p25}, {p75})"),
    all_categorical() ~ "{n} ({p}%)"),
  digits = list(
    all_continuous() ~ 2,
    all_categorical() ~ 0
  )
) %>%
add_p() %>%
add_overall(col_label = "Ech. total") %>%
modify_header(label ~ "**Variable**") %>%
modify_spanning_header(c("stat_1", "stat_2") ~ "**Consonnance du nom**") %>%
add_stat_label(
  label = list(
    all_continuous() ~ c("Moyenne (Ecart-type)", "Médiane (IQR)"),
    all_categorical() ~ "n (%)"
  )
)
```

		Consonnance du nom		
Variable	Ech. total	**Caucasien**, N = 2,435	**Afro-americain**, N = 2,435	**p-valeur**
Genre, n (%)				0.4
homme	1,124 (23%)	575 (24%)	549 (23%)	
femme	3,746 (77%)	1,860 (76%)	1,886 (77%)	
Entretien, n (%)				<0.001
Non	4,478 (92%)	2,200 (90%)	2,278 (94%)	
Oui	392 (8%)	235 (10%)	157 (6%)	
Nombre d'emplois listés, n (%)				0.7
1	110 (2%)	54 (2%)	56 (2%)	
2	704 (14%)	347 (14%)	357 (15%)	
3	1,429 (29%)	726 (30%)	703 (29%)	
4	1,611 (33%)	800 (33%)	811 (33%)	
5	533 (11%)	258 (11%)	275 (11%)	
6	464 (10%)	243 (10%)	221 (9%)	
7	19 (0%)	7 (0%)	12 (0%)	
Nombre d'années d'expérience				0.9
Moyenne (Ecart-type)	7.84 (5.04)	7.86 (5.08)	7.83 (5.01)	
Médiane (IQR)	6.00 (5.00, 9.00)	6.00 (5.00, 9.00)	6.00 (5.00, 9.00)	

[1] test du khi-deux d'indépendance ; test de Wilcoxon-Mann-Whitney

Chapitre 8

Régressions linéaires

Pour aborder sereinement ce chapitre, il est utile de savoir manipuler des vecteurs, des matrices et des tableaux de données.
Des compléments de lecture sont disponibles dans le chapitre des notes de cours concernant la régression linéaire avec R.

8.1 Contexte et rappels

Dans ce chapitre, nous allons utiliser R pour estimer la relation entre une variable à expliquer et une ou plusieurs autres variables. Nous effectuerons quelques rappels d'économétrie, puis effectuerons les calculs à la main dans un premier temps. Ensuite, nous utiliserons des routines permettant d'estimer des modèles de régression linéaire et nous attacherons à expliquer comment lire et extraire les résultats obtenus.

Pour illustrer notre propos, nous tâcherons de relier la mesure de la taille de l'empan à la taille d'un humain. L'empan correspond à la distance entre l'extrêmité du pouce et celle de l'auriculaire, comme illustré sur la Figure 8.1.

En notant y_i la taille de l'individu i, et x son empan, il s'agit de regarder si la taille dépend linéairement de l'empan. On pose alors, pour tous les individus $i = 1,$
$ldots, n$:

$$y_i = \beta_0 x_i + \beta_1 + \varepsilon_i,$$

avec $\mathbb{E}(\varepsilon_i) = 0, \quad \forall i \in 1, \dots, n$, $\mathbb{E}(\varepsilon_i \varepsilon_j) = 0, \forall i \neq j$, et $\mathbb{V}(\varepsilon_i) = \sigma_\varepsilon^2, \quad \forall i = 1, \dots, n.$

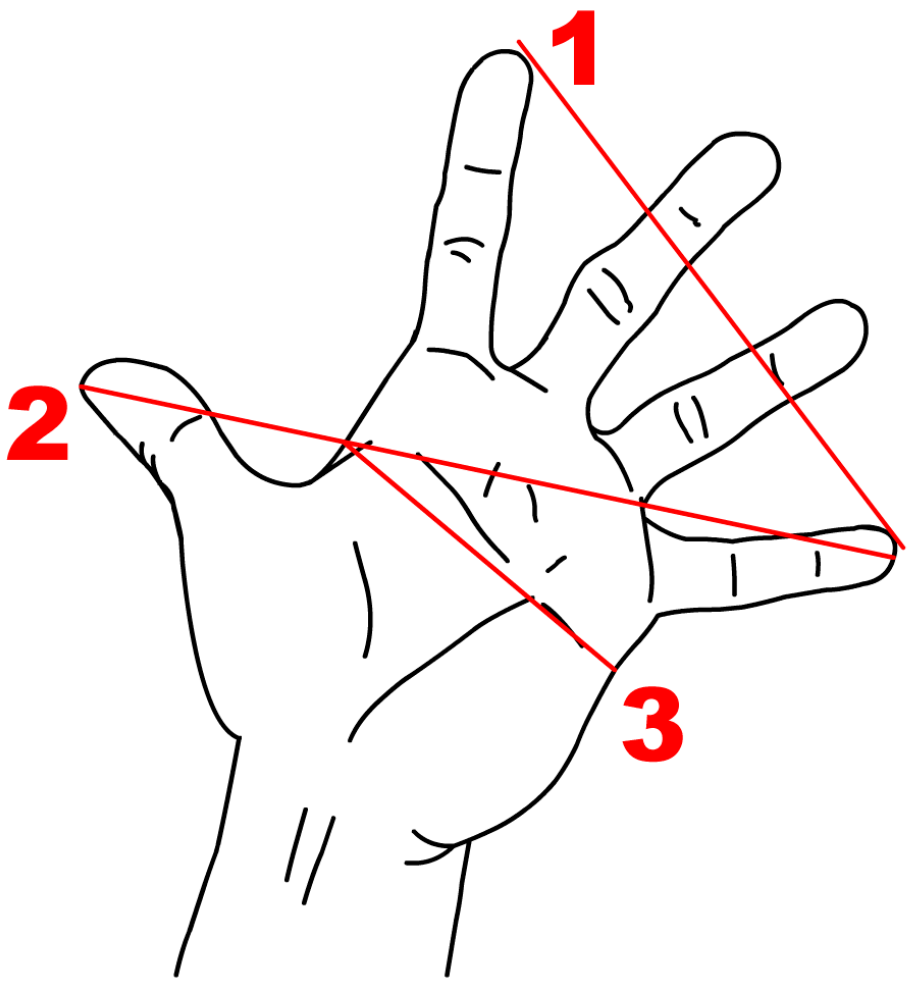


Figure 8.1 – Schéma montrant que l'empan correspond à la distance entre l'extrêmité du pouce et celle de l'auriculaire.

En termes matriciels, on peut noter :

$$y = X\beta + \varepsilon,$$

avec $\varepsilon \sim \mathcal{N}(0, \Sigma^2)$, et

$$\text{où } y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, X = \begin{bmatrix} 1 & x_{1,1} \\ 1 & x_{1,2} \\ 1 & \vdots \\ 1 & x_{1,n} \end{bmatrix}, \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} \text{ et } \varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}.$$

La méthode des moindres carrés ordinaires (MCO) propose une estimation $(\hat{\alpha}, \hat{\beta})$ telle que la somme des carrés des résidus soit minimisée :

$$\arg\min_{\beta} \mid\mid y - X\beta \mid\mid^2 = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - x_i\beta)^2.$$

Les résidus correspondent à l'écart entre la valeur observée de la taille y_i et celle prédite par le modèle $\hat{y}_i$: $e_i = y_i - \hat{y}_i$.

Le modèle estimé s'écrit :

$$\hat{y_i} = \hat{\beta}_0 + \hat{\beta}_1 x_i,$$

soit, en termes matriciels :

$$\hat{y} = X\hat{\beta},$$

$$\text{où } \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix}, X = \begin{bmatrix} 1 & x_{1,1} \\ 1 & x_{1,2} \\ 1 & \vdots \\ 1 & x_{1,n} \end{bmatrix}, \text{ et } \hat{\beta} = \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \end{bmatrix}.$$

La condition du premier ordre donne [1] :

$$\begin{aligned} & X^t X\hat{\beta} - 2X^t X\hat{\beta} - 2X^t y = 0 \\ \Leftrightarrow \quad & X^t X\hat{\beta} = X^t y \\ \Leftrightarrow \quad & \hat{\beta} = (X^t X)^{-1} X^t y. \end{aligned}$$

8.2 Chargement du jeu de données

1. On utilise les propriétés suivantes : $\frac{\partial x^t A}{\partial x} = A^t$, $\frac{\partial Ax}{\partial x} = A$ et $\frac{\partial a\varepsilon}{\partial x} = a\frac{\partial u}{\partial x}$, avec $u = u(x)$.

Créez un projet RStudio pour ce chapitre (cf. le chapitre d'initiation à R). Pensez bien à créer le répertoire `Data` dans le répertoire contenant le fichier de projet `.Rproj`.

Les données ont été mesurées sur des étudiantes et étudiants de l'Université de Rennes 1. Elles sont disponibles dans un fichier CSV à l'adresse suivante : https://egallic.fr/Enseignement/L3_EcoFi/Exercices/empan.csv. Téléchargeons ces données et enregistrons le fichier dans le répertoire `Data` :

```
download.file("https://egallic.fr/Enseignement/L3_EcoFi/Exercices/empan.csv",
              destfile = "Data/empan.csv")
```

Puis, chargeons-le dans R :

```
library(tidyverse)
df_empan <- read_csv("Data/empan.csv")
df_empan
```

```
## # A tibble: 40 x 6
##    empan_main_forte empan_main_faible   age genre taille forte
##               <dbl>             <dbl> <dbl> <chr>  <dbl> <chr>
##  1             19                19      19 H        172 D
##  2             20.5              20.5    27 H        179 G
##  3             21                21      26 H        179 G
##  4             19                19      27 F        160 D
##  5             21                22      25 H        186 D
##  6             19                18.5    25 H        178 D
##  7             19                19      20 H        182 B
##  8             21                22      21 H        179 D
##  9             23                22      19 H        186 D
## 10             20                20      20 F        163 D
## # ... with 30 more rows
```

Les colonnes sont les suivantes (les caractéristiques des individus) :

- `empan_main_forte` : empan main forte (en cm)
- `empan_main_faible` : empan main faible (en cm)
- `age` : age (en année)
- `genre` : genre (H, F, A)
- `taille` : taille (en cm)
- `forte` : main forte (“G” pour gauche, “D” pour droite, “B” pour ambidextre)

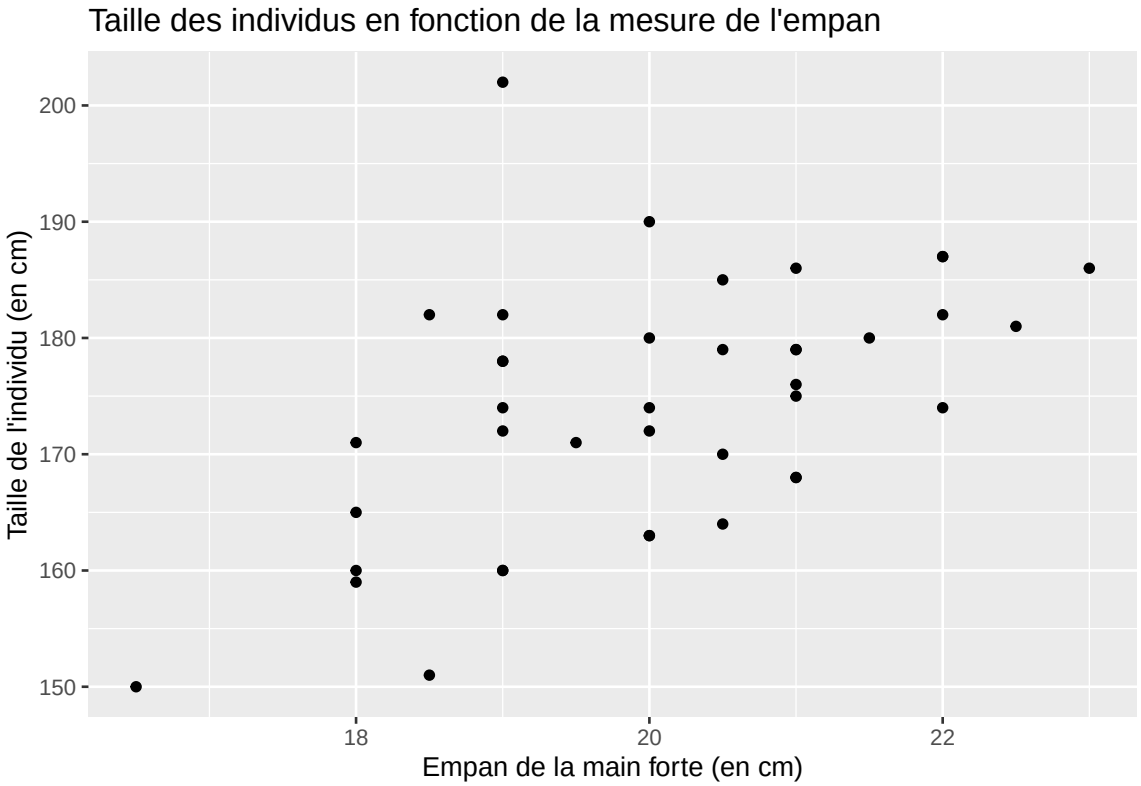
1. Affichez quelques statistiques descriptives pour vous familiariser avec les données (moyenne, écart-type, quantiles, proportions, etc.). Vous pouvez, par exemple utiliser la fonction `tbl_summary()` de {gtsummary} abordée au chapitre de statistiques descriptives.
2. Calculez la corrélation entre la taille de l'empan de la main forte et la taille des individus.

8.3 Régression pas à pas

Dans cette partie, nous allons effectuer quelques opérations "à la main". Nous allons créer avec R tous les objets nécessaires pour pouvoir estimer par la méthode des moindres carrés les coefficient de la régression. Ensuite, nous regarderons comment réaliser des tests d'hypothèse de nullité des coefficients. La section suivante permettra de retrouver les mêmes résultats à l'aide de routines.

Nous verrons dans un autre chapitre comment réaliser des graphiques avec le *package* {ggplot2}. Pour réaliser des nuages de points facilement, nous pouvons utiliser la fonction `qplot()` de ce *package*. Le tableau de données contenant les observations à afficher est indiqué à l'argument `data=`. Les arguments `x=` et `y=` reçoivent le nom des colonnes du tableau à utiliser pour l'axe des abscisses et des ordonnées, respectivement. L'argument `geom=` définit le type de géométrie à représenter. Dans l'exemple ci-dessous, nous souhaitons réaliser un nuage de points. Nous indiquerons donc `geom = point`. Une fois le nuage créé, nous ajoutons une couche au graphique, pour nommer les axes et le graphique, avec la fonction `labs()`.

```
library(ggplot2)
qplot(x = empan_main_forte, y = taille,
      data = df_empan, geom = "point") +
  labs(x = "Empan de la main forte (en cm)",
       y = "Taille de l'individu (en cm)",
       title = "Taille des individus en fonction de la mesure de l'empan")
```



8.3.1 Estimation des coefficients de la régression par moindres carrés ordinaires

Comme indiqué précédemment, nous souhaitons estimer la relation suivante :

$$y = X\beta + \varepsilon,$$

où y est un vecteur contenant les tailles des individus, X une matrice contenant deux colonne : une pour la constante et une deuxième pour les valeurs de l'empan de la main forte. Nous allons estimer les valeurs des deux coefficients du vecteur β (celui pour la constante, et celui pour l'empan de la main forte).

3. Dans un objet que vous appellerez `n`, stockez le nombre d'observations dans le tableau de données `df`.
4. Créez un vecteur que vous nommerez `constante`. Cet objet doit contenir la valeur `1` répétée `n` fois.
5. Créez un vecteur que vous nommerez `y`. Cet objet doit contenir les valeurs la colonne `taille` du tableau `df_empan`. Cet objet sera notre variable de réponse,

celle que nous souhaitons pouvoir prédire à l'aide du modèle de régression linéaire. Les premières valeurs de `y` seront comme suit :

```
## [1] 172 179 179 160 186 178
```

6. Créez une matrice que vous nommerez `X`, qui contiendra deux colonnes : une première contenant les valeurs de `constante`, et une deuxième contenant les valeurs de la colonne `empan_main_forte` de `df`. Les premières valeurs de `X` seront comme suit :

```
##      [,1] [,2]
## [1,]    1 19.0
## [2,]    1 20.5
## [3,]    1 21.0
## [4,]    1 19.0
## [5,]    1 21.0
## [6,]    1 19.0
```

Avec la méthode des moindres carrés, les coefficients de la régression sont estimé en effectuant le calcul suivant :

$$\hat{\beta} = (X^t X)^{-1} X^t y.$$

7. Calculez la transposée de `X`, c'est-à-dire X^t.
8. Calculez, à l'aide d'un produit matriciel, la matrice $X^t X$
9. Calculez l'inverse de la matrice de la fonction précédente, c'est-à-dire $\left(X^t X\right)^{-1}$
10. Calculez le vecteur colonne $X^t y$
11. Calculez le vecteur des coefficients de régression par la méthode des moindres carrés ordinaires, c'est-à-dire $\hat{\beta} = \left(X^t X\right)^{-1} X^t y$. Stockez le résultat dans un objet que vous nommerez `hat_beta`.
12. Avec les crochets, extrayez chaque coefficient de `hat_beta` (le coefficient associé à la constane – *i.e.*, la pente ici – et le coefficient associé à la taille de l'empan). Vous devez obtenir les valeurs suivantes :

```
## Constante
```

```
## [1] 96.27554
```

```
## Coeff empan
```

```
## [1] 3.885117
```

Note : lors de la séance portant sur les matrices, vous avez déjà appris à effectuer toutes ces opérations.

8.3.2 Prédiction des valeurs

Une fois que nous avons obtenu les coefficients de la régression, nous pouvons les réutiliser pour calculer les valeurs prédites par le modèle, à savoir $\hat{y}$. Il suffit pour cela d'appliquer la formule :

$$\hat{y} = X\hat{\beta}.$$

13. Calculez les valeurs prédites, c'est-à-dire $\hat{y} = X\hat{\beta}$. Stockez les valeurs dans un objet que vous nommerez `y_pred`. Les premières valeurs sont les suivantes :

```
##          [,1]
## [1,] 170.0928
## [2,] 175.9204
## [3,] 177.8630
## [4,] 170.0928
## [5,] 177.8630
## [6,] 170.0928
```

8.3.3 Les résidus de la régression

Les résidus correspondent à la différence entre les valeurs observées et les valeurs prédites, soit $e = y - X\hat{\beta}$.

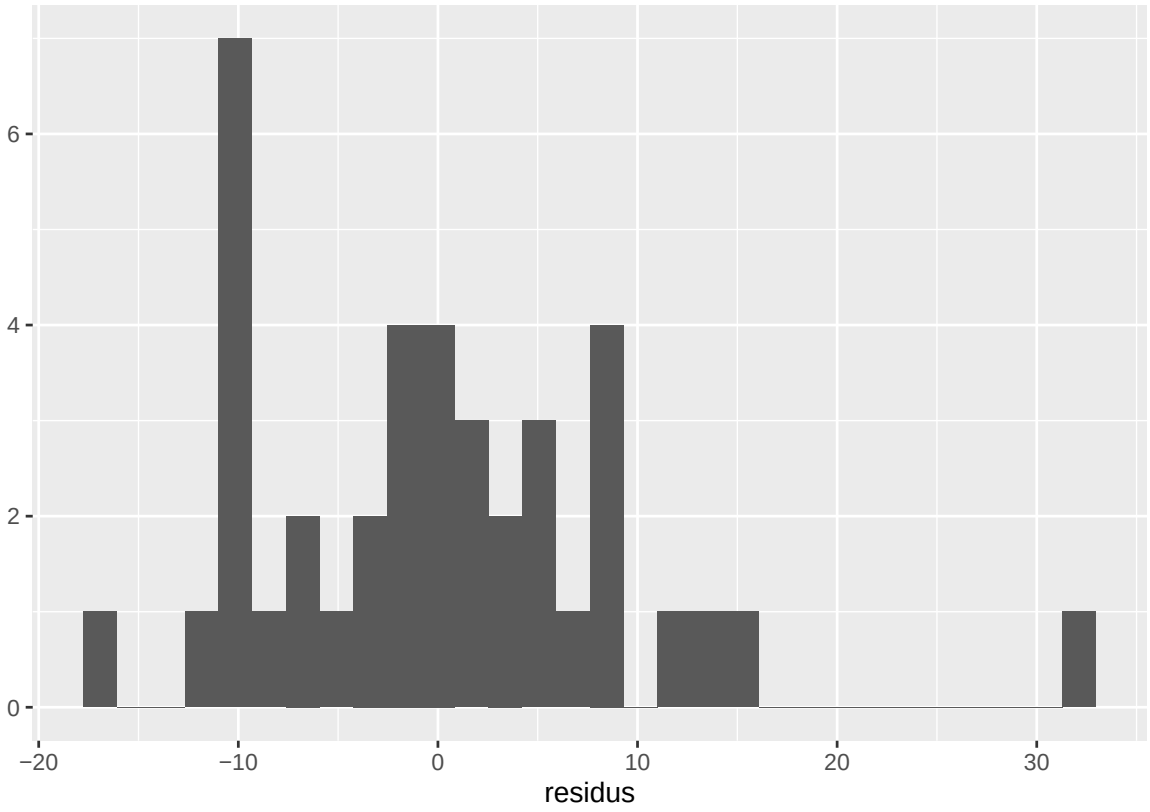
14. Calculez les résidus $e = y - X\hat{\beta}$. Stockez le résultat dans un objet que vous nommerez `residus`. Les premières valeurs sont les suivantes :

```
##            [,1]
## [1,]   1.907245
## [2,]   3.079570
## [3,]   1.137011
## [4,] -10.092755
## [5,]   8.137011
```

```
## [6,]   7.907245
```

Nous pouvons regarder la distribution des résidus, pour voir s'ils ont une allure normale (ici, nous avons peu d'observations…) :

```
qplot(residus, geom="histogram")
```



8.3.4 Un indicateur de la qualité d'ajustement : le coefficient de détermination

Pour avoir une idée de la qualité d'ajustement du modèle, il est coutume de calculer le coefficient de détermination, le R^2. Nous pouvons au préalable calculer la somme des carrés des écarts expliqués ainsi que la somme des carrés totale.

La variance peut se décomposer en deux termes : une part expliquée par le modèle et une part inexpliquée :

$$\underbrace{(y_i - \bar{y})}_{\text{Ecart total}} = \underbrace{(\hat{y}_i - \bar{y})}_{\text{Écart expliqué}} + \underbrace{(y_i - \hat{y}_i)}_{\text{Écart résiduel}} ,$$

avec $\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ la moyenne de$y$.

On peut montrer que :

$$\underbrace{\sum_{i=1}^{n}(y_i - \bar{y})^2}_{\text{SCT}} = \underbrace{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}_{\text{SCE}} + \underbrace{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}_{\text{SCR}},$$

avec :

— SCT : la somme des carrés totale
— SCE : la somme des carrés des écarts expliqués par le modèle
— SCR : la somme des carrés des résidus

15. Calculez la somme des carrés des résidus, c'est-à-dire SCR $= \sum_{i=1}^{n} \left(y_i - \hat{y_i}\right)^2$. Stockez le résultat dans un objet que vous nommerez `somme_carre_residus`. Vous obtiendrez :

```
## [1] 3421.886
```

16. Calculez SCT et SCE et stockez les valeurs dans des objets que vous nommerez `somme_carres_totale` et `somme_carres_expliques`, respectivement. Vous obtiendrez :

```
## Somme des carrés totale :
```

```
## [1] 4666.775
```

```
## Somme des carrés expliqués :
```

```
## [1] 1244.889
```

17. Calculez le coefficient de détermination : $R^2 = \frac{\text{SCE}}{\text{SCT}}$. Vous obtiendrez :

```
r_2
```

8.3.5 Tests de nullité des coefficients de la régression linéaire

À présent, nous souhaitons pouvoir effectuer des tests de significativité des coefficients de la régression. Le test se présente sous la forme suivante :

$$\begin{cases} H_0 : \beta_i = 0 \\ H_1 : \beta_i \neq 0 \end{cases} , i = 1, 2$$

Ce test s'appuie sur la statistique de test suivante :

$$T = \frac{\hat{\beta}_i - \beta_{i,H_0}}{\hat{\sigma}_{\hat{\beta}_i}} \sim \mathcal{S}t(n - m - 1),$$

avec β_{i,H_0} la valeur de β_i sous l'hypothèse nulle, $\hat{\sigma}_{\hat{\beta}_i}$ l'estimation de l'écart-type de l'estimation du paramètre β_i.

Pour effectuer ce test bilatéral, on peut lire dans la table de la loi de Student deux fractiles tels que :

$$\mathbb{P}\left(-t_{1-\alpha/2} < \frac{\hat{\beta}_i - \alpha_{i,H_0}}{\hat{\sigma}_{\hat{\beta}_i}} < t_{1-\alpha/2}\right) = 1 - \alpha.$$

avec α le risque de première espèce (le risque de rejeter H_0 alors qu'elle est vraie).

À partir des observations, il est possible de calculer :

$$t_{i,\text{obs.}} = \frac{\hat{\beta}_i}{\hat{\sigma}_{\hat{\beta}_i}}.$$

La règle de décision est la suivante :

- si $t_{i,\text{obs.}} \in [-t_{1-\alpha/2}, t_{1-\alpha/2}]$, nous somme dans la région de non rejet, on ne rejette donc pas H_0 au seuil α (avec $\alpha = 5\%$, par exemple), et on considère alors que β_i n'est pas statistiquement différent de zéro ;
- si en revanche $t_{i,\text{obs.}} \notin [-t_{1-\alpha/2}, t_{1-\alpha/2}]$, nous sommes dans la région critique et cette fois on rejette l'hypothèse nulle en faveur de l'hypothèse alternative. On considère alors qu'avec un risque de première espèce α, on a $\alpha_i \neq 0$.

18. Calculez la variance des erreurs $\hat{\sigma}^2_\varepsilon = \frac{\text{SCR}}{n-m-1}$. Stockez le résultat dans un objet que vous nommerez `hat_sigma_2_u`. Vous devez obtenir :

```
## [1] 90.04964
```

19. Calculez la matrice de variance-covariance des estimateurs $\mathbb{V}\left(\hat{\beta}\right) = \hat{\sigma}_{\varepsilon}^2 \left(X^t X\right)^{-1}$. Stockez le résultat dans un objet que vous nommerez `var_cov`. Vous obtiendrez la matrice suivante :

```
##            [,1]       [,2]
## [1,] 440.08045 -21.864130
## [2,] -21.86413   1.091842
```

20. Extrayez les éléments de la diagonale de la matrice de variance-covariance (les variances, donc), puis calculez leur racine carrée de manière à obtenir les erreurs-types $\hat{\sigma}_{\hat{\beta}}$. Stockez le résultat dans un vecteur que vous nommerez `hat_sigma_betas`. Vous obtiendrez les valeurs suivantes :

```
## [1] 20.978094  1.044912
```

21. Pour chaque coefficient, calculez la statistique observée du test de nullité des coefficients $t_{i,\text{obs.}} = \frac{\hat{\beta}_i}{\hat{\sigma}_{\hat{\beta}_i}}$. Stockez le résultat dans un vecteur que vous nommerez `t_obs`. Vous obtiendrez les valeurs suivantes :

```
##          [,1]
## [1,] 4.589337
## [2,] 3.718127
```

22. À l'aide de la fonction `qt()`, stockez dans un objet que vous nommerez `t_tab` le quantile d'ordre 95% d'une Student à `n-m-1` degrés de libertés. Vous obtiendrez la valeur suivante :

```
## [1] 2.024394
```

23. Comparez les valeurs de `t_obs` avec celles de `t_tab`. Concluez pour chaque coefficient.
24. À l'aide de la fonction `pt()`, calculez la p-value associée au test de nullité de chaque coefficient (rappel : nous effectuons un test bilatéral ici, donc nous cherchons $2 \times \mathbb{P}(|T| > t_{obs})$). Vous obtiendrea les valeurs suivantes :

```
p_val
```

```
##                [,1]
## [1,] 4.745605e-05
## [2,] 6.453478e-04
```

8.4 Régression avec lm

La fonction permettant de réaliser une régression linéaire avec R se nomme `lm()`. Il est nécessaire de fournir une formule à l'arguemnt `formula=`. L'argument `data=` indique le tableau de données dans lequel les variables mentionnées dans la formule se trouvent.

Pour écrire la formule, on indique le nom de la variable à expliquer, puis, après avoir ajouté un tilde (`~`), on écrit le nom des variables explicatives en les séparant par un symbole plus (`+`). Par défaut, la constante est ajoutée au modèle.

```
reg <- lm(taille ~ empan_main_forte, data = df_empan)
reg
```

```
##
## Call:
## lm(formula = taille ~ empan_main_forte, data = df_empan)
##
## Coefficients:
##       (Intercept)  empan_main_forte
##            96.276             3.885
```

Note
Pour estimer le modèle sans la constante, on ajoute dans le membre à droite du tilde : `- 1` :

```
reg_2 <- lm(taille ~ -1 +empan_main_forte, data = df_empan)
reg_2
```

```
##
## Call:
## lm(formula = taille ~ -1 + empan_main_forte, data = df_empan)
##
## Coefficients:
## empan_main_forte
```

```
##             8.668
```

Pour ajouter des variables explicatives, il suffit d'écrire leur nom en séparant chaque variables par le symbole plus (`+`) :

```
reg_3 <- lm(taille ~ empan_main_forte + genre, data = df_empan)
reg_3
```

```
##
## Call:
## lm(formula = taille ~ empan_main_forte + genre, data = df_empan)
##
## Coefficients:
##      (Intercept)  empan_main_forte            genreH
##          141.897             1.058            16.275
```

L'objet retourné par la fonction `lm()` est de classe `lm`. Il s'agit d'une liste contenant plusieurs éléments nommés, dont :

- `coefficients` : un vecteur nommé de coefficients ($\hat{\beta}$) ; les noms correspondent aux noms des variables explicatives
- `residuals` : les résidus
- `fitted.values` : les valeurs prédites par le modèle

On peut donc accéder à ces éléments avec les crochets, ou à leur contenu avec les doubles crochets (ou le dollar) :

```
reg[["coefficients"]]
```

```
##      (Intercept) empan_main_forte
##        96.275538         3.885117
```

```
reg$coefficients
```

```
##      (Intercept) empan_main_forte
##        96.275538         3.885117
```

On retrouve bien les mêmes valeurs que celles que nous avons calculées "à la main". On note que le coefficient associé à la constante se nome `(Intercept)` (soit la pente de la droite de régression).

8.4.1 Résumé de l'estimation

La fonction `summary()` permet d'obtenir quelques résumés concernant l'estimation :

```
resume_reg <- summary(reg)
resume_reg
```

```
##
## Call:
## lm(formula = taille ~ empan_main_forte, data = df_empan)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -17.150  -7.343   0.108   5.252  31.907
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)         96.276     20.978   4.589 4.75e-05 ***
## empan_main_forte     3.885      1.045   3.718 0.000645 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.489 on 38 degrees of freedom
## Multiple R-squared:  0.2668, Adjusted R-squared:  0.2475
## F-statistic: 13.82 on 1 and 38 DF,  p-value: 0.0006453
```

Les valeurs qui s'affichent sont accessibles (avec les crochets doubles ou le symbole dollar) :

```
resume_reg$coefficients
```

```
##                   Estimate Std. Error  t value     Pr(>|t|)
## (Intercept)      96.275538  20.978094 4.589337 4.745605e-05
## empan_main_forte  3.885117   1.044912 3.718127 6.453478e-04
```

Regardons d'un peu plus près les différents éléments de la sortie :

- `Call` : la formule du modèle.
- `Residuals` : des statistiques descriptives des résidus.
- `Coefficients` : un tableau à deux entrées où les lignes correspondent aux coefficients associés aux variables explicatives, et les colonnes, dans l'ordre, à l'estimation du coefficient, l'écart-type estimé, la valeur du test de Student de nullité statistique du coefficient et enfin la p-value associé à ce test, suivie d'un symbole pour lire rapidement la significativité.
- `Signif. codes` : les significations des symboles de niveau de significativité.

— `Residual standard error` : estimation de l'écart-type de l'aléa et degré de liberté.
— `Multiple R-squared` : coefficient de détermination.
— `Adjusted R-squared` : coefficient de détermination ajusté.
— `F-statistic` : valeur de la statistique de Fisher du test de significativité globale, ainsi que les degrés de liberté et la p-value associée au test.

8.4.2 Prédictions des valeurs

Les valeurs prédites par le modèle sont dans l'élément `fitted.values` de l'objet retourné par `lm()`.

```
reg$fitted.values
```

```
##        1        2        3        4        5        6        7
          8
## 170.0928 175.9204 177.8630 170.0928 177.8630 170.0928 170.0928
   177.8630
##        9       10       11       12       13       14       15
         16
## 185.6332 173.9779 170.0928 173.9779 173.9779 173.9779 166.2076
   181.7481
##       17       18       19       20       21       22       23
         24
## 183.6907 175.9204 168.1502 181.7481 170.0928 177.8630 173.9779
   177.8630
##       25       26       27       28       29       30       31
         32
## 173.9779 170.0928 177.8630 170.0928 168.1502 177.8630 181.7481
   172.0353
##       33       34       35       36       37       38       39
         40
## 181.7481 179.8055 175.9204 166.2076 166.2076 160.3800 175.9204
   166.2076
```

La fonction `predict()` peut aussi être appliquée à l'objet de régression (il s'agit d'un alias de la fonction `predict.lm()` ; pour afficher la fiche d'aide il faut écrire `?predict.lm()`) :

```
predict(reg)
```

```
##        1        2        3        4        5        6        7
          8
## 170.0928 175.9204 177.8630 170.0928 177.8630 170.0928 170.0928
   177.8630
```

```
##         9        10        11        12        13        14        15
         16
## 185.6332 173.9779 170.0928 173.9779 173.9779 173.9779 166.2076
   181.7481
##        17        18        19        20        21        22        23
         24
## 183.6907 175.9204 168.1502 181.7481 170.0928 177.8630 173.9779
   177.8630
##        25        26        27        28        29        30        31
         32
## 173.9779 170.0928 177.8630 170.0928 168.1502 177.8630 181.7481
   172.0353
##        33        34        35        36        37        38        39
         40
## 181.7481 179.8055 175.9204 166.2076 166.2076 160.3800 175.9204
   166.2076
```

Cette fonction permet par ailleurs d'effectuer des prédictions pour de nouvelles valeurs, à condition de fournir un tableau de données contenant les colonnes utilisées comme variables explicatrices. Pour prédire la taille d'une personne, selon notre premier modèle, si la mesure de l'empan est de 15 cm :

```
new_data <- tibble(empan_main_forte = 15)
predict(reg, newdata = new_data)
```

```
##        1
## 154.5523
```

Avec la régression de la taille sur l'empan de la main forte et le genre, si la mesure de l'empan est de 15 cm et que l'individu est un homme :

```
predict(reg_3, newdata = tibble(empan_main_forte = 15, genre = "H"))
```

```
##        1
## 174.0465
```

> 25. À l'aide d'un mètre-ruban, mesurez la taille de votre empan. En utilsant le modèle estimé de votre choix, prédisez votre taille. Comparez avec la réalité.

Pour munir une prédiction d'un intervalle de confiance, on peut ajouter des arguments à la fonction `predict()`. Avec l'argument `interval=`, on choisit le type d'intervalle

("confidence" ou "prediction"), avec `level=` on choisir le niveau de confiance, et avec `se.fit=` on précise si l'on souhaite ou non que l'écart-type estimé soit retourné.

```
pred <-
  predict(reg, newdata = new_data,
          interval = "prediction", level = 0.95, se.fit = TRUE)
pred
```

```
## $fit
##        fit       lwr       upr
## 1 154.5523 132.3881 176.7165
##
## $se.fit
## [1] 5.460854
##
## $df
## [1] 38
##
## $residual.scale
## [1] 9.489449
```

Dans l'élément `fit`, les valeurs sont les suivantes : `fit` pour la valeur prédite, `lwr` et `upr` les bornes inférieure et supérieure de l'intervalle de confiance de la prédiction (`lwr` pour *lower* et `upr` pour *upper*).

8.4.3 Tests de nullité des coefficients de la régression linéaire

L'élément appelé `coefficients` obtenu en application de la fonction `summary()` sur l'objet issu de la régression contient un tableau donnant pour chaque coefficient (en ligne), la valeur estimée (`Estimate`), l'écart-type de l'estimateur (`Std. Error`), la valeur observé de la statistique de Student du test de nullité du coefficient (`Std. Error`) et la p-value associée au test (`Pr(>|t|)`).

```
resume_reg$coefficients
```

```
##                   Estimate Std. Error  t value     Pr(>|t|)
## (Intercept)      96.275538  20.978094 4.589337 4.745605e-05
## empan_main_forte  3.885117   1.044912 3.718127 6.453478e-04
```

8.4.4 Mise en forme des résultats avec modelsummary

Pour créer de jolis tableaux de statistiques descriptives, nous allons utiliser la fonction `modelsummary()` du *package* du même nom. Un tutoriel est disponible sur le site du créateur

du *package*, Vincent Arel-Bundock. Nous allons ici explorer uniquement les rudiments de la fonction `modelsummary()`.

Note
Nous avons vu dans le chapitre sur les résumés statistiques que nous pouvions utiliser une fonction du *package* {gtsummary}. Pour des tableaux présentant les tableaux de régression, bien que {gtsummary} propose une fonction nommée `tbl_regression()`, nous préférons utiliser ici le *package* {modelsummary}. Pour celles et ceux qui voudraient apprendre à utiliser `tbl_regression()`, une bonne vignette facile d'accès est disponible sur le site de Daniel D. Sjoberg, le créateur du *package*.

Dans un premier temps, nous allons charger le *package* {modelsummary} :

```
library(modelsummary)
```

L'argument `models=` de la fonction `modelsummary()` reçoit un objet issu d'une régression (ou une liste d'objets issus de régressions si l'on souhaite afficher plusieurs estimateurs provenant de plusieurs modèles). En fournissant les objets issus d'une régression dans une liste nommée, les noms que l'on donne sont ensuite utilisés en en-tête.

```
modelsummary(models = list("MCO 1" = reg,
                           "MCO 2" = reg_2,
                           "MCO 3" = reg_3))
```

	MCO 1	MCO 2	MCO 3
(Intercept)	96.276		141.897
	(20.978)		(17.069)
empan_main_forte	3.885	8.668	1.058
	(1.045)	(0.092)	(0.896)
genreH			16.275
			(2.748)
Num.Obs.	40	40	40
R2	0.267	0.996	0.624
R2 Adj.	0.247	0.996	0.603
AIC	297.5	313.1	272.8
BIC	302.5	316.5	279.6
Log.Lik.	−145.739	−154.559	−132.403
F	13.824		30.649
RMSE	9.49	11.68	6.89

Le tableau ainsi créé retourne les coefficients estimés et leur écart-type pour chaque modèle, suivis de du nombre d'observations et de quelques statistiques de qualité d'ajustement des modèles (R^2, R^2 ajusté, AIC, BIC, log vraisemblance, F de Fisher, racine de l'erreur quadratique moyenne).

Lorsque nous affichons le résumé d'une régression avec la fonction `summary()`, le **nombre de décimales des coefficients** peut s'avérer plus grand que nécessaire pour la communication des résultats. L'argument `fmt=` permet de déterminer comment formater les valeurs numériques du tableau. Si l'on souhaite par exemple ne faire figurer que 2 décimales, on peut écrire :

```
modelsummary(models = list("MCO 1" = reg,
                           "MCO 2" = reg_2,
                           "MCO 3" = reg_3),
             fmt = 2)
```

	MCO 1	MCO 2	MCO 3
(Intercept)	96.28		141.90
	(20.98)		(17.07)
empan_main_forte	3.89	8.67	1.06
	(1.04)	(0.09)	(0.90)
genreH			16.28
			(2.75)
Num.Obs.	40	40	40
R2	0.267	0.996	0.624
R2 Adj.	0.247	0.996	0.603
AIC	297.5	313.1	272.8
BIC	302.5	316.5	279.6
Log.Lik.	−145.739	−154.559	−132.403
F	13.824		30.649
RMSE	9.49	11.68	6.89

Nous pouvons aussi utiliser une fonction pour formater les nombres, ce qui peut s'avérer utile pour produire un tableau avec des normes françaises (virgule utilisée comme séparateur décimal, espace comme sépartateur de milliers).

```
fmt_fr <- function(x){
  formatC(x, digits = 2, big.mark = " ", decimal.mark = ",", format = "f")
}
```

```
modelsummary(models = list("MCO 1" = reg,
                           "MCO 2" = reg_2,
                           "MCO 3" = reg_3),
             fmt = fmt_fr)
```

	MCO 1	MCO 2	MCO 3
(Intercept)	96.28		141.90
	(20.98)		(17.07)
empan_main_forte	3.89	8.67	1.06
	(1.04)	(0.09)	(0.90)
genreH			16.28
			(2.75)
Num.Obs.	40	40	40
R2	0.267	0.996	0.624
R2 Adj.	0.247	0.996	0.603
AIC	297.5	313.1	272.8
BIC	302.5	316.5	279.6
Log.Lik.	−145.739	−154.559	−132.403
F	13.824		30.649
RMSE	9.49	11.68	6.89

Note
On remarque que les statistiques en bas du tableau ne sont pas affectées par l'argument `fmt=`. Nous verrons plus bas comment changer le format de ces statistiques.

Pour faire figurer une indication visuelle permettant de savoir rapidement le **seuil de significativité du test de nullité des coefficients**, la fonction `modelsummary()` accepte un argument `star=`. Les symboles retenus sont les suivants : `+` pour un seuil de 10%, `*` pour un seuil de 5%, `**` pour un seuil de 1%, `***` pour un seuil de 0,1%. La légende est ajoutée en note de bas de tableau.

```
modelsummary(models = list("MCO 1" = reg,
                           "MCO 2" = reg_2,
                           "MCO 3" = reg_3),
             stars = TRUE)
```

	MCO 1	MCO 2	MCO 3
(Intercept)	96.276***		141.897***
	(20.978)		(17.069)
empan_main_forte	3.885***	8.668***	1.058
	(1.045)	(0.092)	(0.896)
genreH			16.275***
			(2.748)
Num.Obs.	40	40	40
R2	0.267	0.996	0.624
R2 Adj.	0.247	0.996	0.603
AIC	297.5	313.1	272.8
BIC	302.5	316.5	279.6
Log.Lik.	−145.739	−154.559	−132.403
F	13.824		30.649
RMSE	9.49	11.68	6.89

+ p < 0.1, * p < 0.05, ** p < 0.01, *** p < 0.001

Pour utiliser des seuils que l'on définit nous-mêmes, il suffit de fournir à l'argument `stars=` un vecteur nommé : le nom correspond au symbole utilisé, la valeur au seuil retenu.

```
modelsummary(models = list("MCO 1" = reg,
                           "MCO 2" = reg_2,
                           "MCO 3" = reg_3),
             stars = c("*" = .1, "**" = .05, "***" = 0.01))
```

	MCO 1	MCO 2	MCO 3
(Intercept)	96.276***		141.897***
	(20.978)		(17.069)
empan_main_forte	3.885***	8.668***	1.058
	(1.045)	(0.092)	(0.896)
genreH			16.275***
			(2.748)
Num.Obs.	40	40	40
R2	0.267	0.996	0.624
R2 Adj.	0.247	0.996	0.603
AIC	297.5	313.1	272.8
BIC	302.5	316.5	279.6
Log.Lik.	−145.739	−154.559	−132.403
F	13.824		30.649
RMSE	9.49	11.68	6.89

* p < 0.1, ** p < 0.05, *** p < 0.01

L'argument `coef_rename=` permet de **renommer les coefficients dans le tableau**, afin de produire un résultat plus lisible.

```
modelsummary(models = list("MCO 1" = reg,
                           "MCO 2" = reg_2,
                           "MCO 3" = reg_3),
             stars = c("*" = .1, "**" = .05, "***" = 0.01),
             coef_rename = c("(Intercept)" = "Constante",
                             "empan_main_forte" = "Empan main forte",
                             "genreH" = "Genre - Homme"))
```

	MCO 1	MCO 2	MCO 3
Constante	96.276***		141.897***
	(20.978)		(17.069)
Empan main forte	3.885***	8.668***	1.058
	(1.045)	(0.092)	(0.896)
Genre - Homme			16.275***
			(2.748)
Num.Obs.	40	40	40
R2	0.267	0.996	0.624
R2 Adj.	0.247	0.996	0.603
AIC	297.5	313.1	272.8
BIC	302.5	316.5	279.6
Log.Lik.	−145.739	−154.559	−132.403
F	13.824		30.649
RMSE	9.49	11.68	6.89

* p < 0.1, ** p < 0.05, *** p < 0.01

L'argument `gof_map=` contrôle l'**affichage des statistiques de qualité d'ajustement** (*goodness of fit*) et d'autres informations relatives aux modèles.

```
modelsummary(models = list("MCO 1" = reg,
                           "MCO 2" = reg_2,
                           "MCO 3" = reg_3),
             stars = c("*" = .1, "**" = .05, "***" = 0.01),
             coef_rename = c("(Intercept)" = "Constante",
                             "empan_main_forte" = "Empan main forte",
                             "genreH" = "Genre - Homme"),
             gof_map = c("nobs", "r.squared", "adj.r.squared", "F"))
```

	MCO 1	MCO 2	MCO 3
Constante	96.276***		141.897***
	(20.978)		(17.069)
Empan main forte	3.885***	8.668***	1.058
	(1.045)	(0.092)	(0.896)
Genre - Homme			16.275***
			(2.748)
Num.Obs.	40	40	40
R2	0.267	0.996	0.624
R2 Adj.	0.247	0.996	0.603
F	13.824		30.649

* p < 0.1, ** p < 0.05, *** p < 0.01

Les noms des statistiques à afficher correspondent aux noms des colonnes du tableau retourné par l'évaluation de la fonction `get_gof()` :

```
get_gof(reg)
```

```
## # A tibble: 1 x 14
##   r.squared adj.r.squared sigma statistic  p.value    df logLik
   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <dbl>  <dbl> <dbl
   > <dbl>
## 1     0.267         0.247  9.49      13.8 0.000645     1  -146.
   297.  303.
## # ... with 5 more variables: deviance <dbl>, df.residual <int>, nobs
     <int>,
## #   F <dbl>, rmse <dbl>
```

Pour **renommer** et contrôler le **format** des stastistiques affichées dans le bas du tableau, on peut fournir à l'argument `gof_map=` une liste de liste, chacune contenant trois éléments :

1. `raw` : le nom de la statiqtique telle que retournée par la fonction `get_gof()`‘
2. `clean` : le nom à afficher dans le tableau
3. `fmt` : le format à utiliser (on peut à nouveau utiliser une fonction définie par nous-mêmes) :

```
modelsummary(
  models = list("MCO 1" = reg,
```

```
                "MCO 2" = reg_2,
                "MCO 3" = reg_3),
  fmt = fmt_fr,
  stars = c("*" = .1, "**" = .05, "***" = 0.01),
  coef_rename = c("(Intercept)" = "Constante",
                  "empan_main_forte" = "Empan main forte",
                  "genreH" = "Genre - Homme"),
  gof_map = list(
    list("raw" = "nobs", "clean" = "N", "fmt" = fmt_fr),
    list("raw" = "r.squared", "clean" = "R2", "fmt" = fmt_fr),
    list("raw" = "adj.r.squared", "clean" = "R2 ajusté", "fmt" = fmt_fr),
    list("raw" = "F", "clean" = "F", "fmt" = fmt_fr)
  )
)
```

	MCO 1	MCO 2	MCO 3
Constante	96.28***		141.90***
	(20.98)		(17.07)
Empan main forte	3.89***	8.67***	1.06
	(1.04)	(0.09)	(0.90)
Genre - Homme			16.28***
			(2.75)
N	40.00	40.00	40.00
R2	0.27	1.00	0.62
R2 ajusté	0.25	1.00	0.60
F	13.82		30.65

* p < 0.1, ** p < 0.05, *** p < 0.01

Pour **exporter le tableau** dans un fichier, la fonction `modelsummary()` propose l'argument `output=`. En fonction de l'extension du fichier dans lequel nous souhaitons exporter le tableau, la fonction '`modelsummary()`' s'adapte. Les extensions que l'on peut utiliser :

- `.docx` : pour un export en format word ;
- `.html` : pour un export en format html ;
- `.tex` : pour un export en format LaTeX ;
- `.md` : pour un export en format markdown ;
- `.txt` : pour un export en format texte brut ;
- `.png` : pour un export en format image PNG.

Voici un exemple pour créer un fichier word (`.docx`) intitulé `regressions.docx`, dans le répetoire `Output` contenu dans le répertoire courant :

```
modelsummary(
  models = list("MCO 1" = reg,
                "MCO 2" = reg_2,
                "MCO 3" = reg_3),
  fmt = fmt_fr,
  stars = c("*" = .1, "**" = .05, "***" = 0.01),
  coef_rename = c("(Intercept)" = "Constante",
                  "empan_main_forte" = "Empan main forte",
                  "genreH" = "Genre - Homme"),
  gof_map = list(
    list("raw" = "nobs", "clean" = "N", "fmt" = fmt_fr),
    list("raw" = "r.squared", "clean" = "R2", "fmt" = fmt_fr),
    list("raw" = "adj.r.squared", "clean" = "R2 ajusté", "fmt" = fmt_fr),
    list("raw" = "F", "clean" = "F", "fmt" = fmt_fr)
  ),
  output = "Output/regression.docx"
)
```

Note
Si un message d'erreur vous indique qu'il faut installer le *package* {flextable} :

```
install.packages("flextable")
```

Enfin, si vous souhaitez afficher le code LaTeX, HTML ou Markdown dans la console (sans créer de fichier), il suffit de donner à l'argument `output=` la chaine de caractère `"latex"`, `"HTML"`, ou `"markdown"`, respectivement.

```
modelsummary(
  models = list("MCO 1" = reg,
                "MCO 2" = reg_2,
                "MCO 3" = reg_3),
  fmt = fmt_fr,
  stars = c("*" = .1, "**" = .05, "***" = 0.01),
  coef_rename = c("(Intercept)" = "Constante",
                  "empan_main_forte" = "Empan main forte",
                  "genreH" = "Genre - Homme"),
```

```
  gof_map = list(
    list("raw" = "nobs", "clean" = "N", "fmt" = fmt_fr),
    list("raw" = "r.squared", "clean" = "R2", "fmt" = fmt_fr),
    list("raw" = "adj.r.squared", "clean" = "R2 ajusté", "fmt" = fmt_fr),
    list("raw" = "F", "clean" = "F", "fmt" = fmt_fr)
  ),
  output = "markdown"
)
```

Chapitre 9

Instructions conditionnelles et boucles

Ce chapitre s'appuie sur les notes de cours concernant les boucles.

Lorsqu'une ou plusieurs instructions doivent être répétées jusqu'à ce qu'une condition soit atteinte, plutôt que d'évaluer manuellement chaque itération du processus de répétition, nous pouvons utiliser des boucles. Dans ce chapitre, nous allons dans un premier temps aborder la notion d'instructions conditionnelles, puis nous verrons deux types de boucles.

9.1 Instructions conditionnelles

Dans le chapitre 2, nous avons rencontré les données de type logique (`TRUE`, `FALSE` et `NA`). Lorsque nous avons réalisé des indexations d'objets par condition, nous avons fait appel à des valeurs logiques retournées par l'évaluation **d'instructions conditionnelles**. Dans ce chapitre, nous allons également nous appuyer sur des valeurs logiques retournées par l'évaluation d'instructions conditionnelles, pour savoir si un morceau de code doit être évalué ou non. La compréhension du fonctionnement des intructions conditionnelles nous permettra par la suite de pouvoir utiliser des boucles, pour savoir si le processur itératif doit continuer ou s'arrêter.

9.1.1 Instruction `if`

Avec une instruction `if`, une action sera prise seulement si une condition particulière est vérifiée. La syntaxe est simple :

```
if(instruction_conditionnelle) action
```

avec `instruction_conditionnelle` une instruction conditionnelle retournant une valeur logique, et `action` une instruction qui sera évaluée uniquement si le logique retourné par l'instruction conditionnelle vaut `TRUE`.

Voici un exemple simple. Admettons que nous disposons d'un objet `x` contenant un entier naturel. Par xemple, supposons que `x` contient l'âge d'une personne, et que nous soyons amenés à divulguer une information à cette personne que si celle-ci a 13 ans ou plus. L'information peut être par exemple l'affichage du message suivant : "La fée des dents n'existe pas".

Admettons que la personne nous ait indiqué avoir 22 ans.

```
x <- 22
```

Pour vérifier si la personne a 13 ans ou plus, on peut écrire l'instruction conditionnelle suivante, qui nous retourne `TRUE` ici :

```
x >= 13
```

```
## [1] TRUE
```

Écrivons à présent notre instruction conditionnelle. Si la personne a 13 ans ou plus, nous afficherons un message dans la console, à l'aide de la fonction `print()` :

```
if(x >= 13) print("La fée des dents n'existe pas")
```

```
## [1] "La fée des dents n'existe pas"
```

Admettons à présent que la personne à qui nous faisons face n'est âgée que de 5 ans.

```
x <- 5
```

Cette fois-ci, lorsqu'on évalue notre instruction conditionnelle, la valeur logique retournée est `FALSE` :

```
x >= 13
```

```
## [1] FALSE
```

Aussi, l'évaluation de notre instruction `if` ne donnera pas lieu à l'évaluation de l'action :

```
if(x >= 13) print("La fée des dents n'existe pas")
```

Si l'action à réaliser est composée de plusieurs instructions, la syntaxe de l'instruction `if` est légèrement modifiée : l'action est simplement placée à l'intérieur d'accolades `{}`.

```
if(instruction_conditionnelle) {
  action_1
  action_2
  ...
  action_n
}
```

Retournons à notre exemple pour illustrer cela. Admettons que nous souhaitons afficher la carré de l'âge de la personne en plus du message.

Retournons à notre individu de 22 ans :

```
x <- 22
```

L'instruction `if` :

```
if(x >= 13) {
  age_carre <- x^2
  print(paste("Votre age au carré vaut :", age_carre))
  print("La fée des dents n'existe pas")
}
```

```
## [1] "Votre age au carré vaut : 484"
## [1] "La fée des dents n'existe pas"
```

Note
L'instruction conditionnelle a été évaluée et la valeur logique `TRUE` a été retournée :

```
x >= 13
```

```
## [1] TRUE
```

Comme l'instruction logique vaut `TRUE`, les instructions à l'intérieur des accoldes ont été exécutées. Dans un premier temps, la variable `age_carre` a été créée. On peut

noter que cette variable existe encore après l'évaluation de l'instruction `if` :

```
age_carre
```

```
## [1] 484
```

Ensuite, les deux instructions suivantes permettant d'afficher des messages dans la console ont été évaluées.

Si notre individu est plus jeune :

```
x <- 5
```

Le test `x >=13` retourne `FALSE` et les actions à l'intérieur des accolades ne sont pas réalisées.

```
if(x >= 13) {
  age_carre <- x^2
  print(paste("Votre age au carré vaut :", age_carre))
  print("La fée des dents n'existe pas")
}
```

Note
L'instruction conditionnelle `x >=13` ayant retourné la valeur logique `FALSE`, la valeur de la variable `age_carre` calculée précédemment n'a pas été modifiée :

```
age_carre
```

```
## [1] 484
```

1. La fonction `readline()` permet de demander à un utilisateur de rentrer une valeur dans la console. En affectant la valeur lue dans la console dans une variable, l'information donnée par l'utilisateur peut être stockée. Évaluez l'instruction suivante, et écrivez votre âge dans la console, puis validez en appuyant sur la touche `Entrée` :

   ```
   age <- readline(prompt="Votre âge : ")
   ```

2. Rédigez une instruction conditionnelle qui retourne la racine carrée de l'âge

renseigné par l'utilisateur (stockée dans l'objet `age`) uniquement si la valeur stockée dans `age` est comprise entre 20 et 25.

3. Essayez d'évaluer votre instruction en renseignant différentes valeurs pour âge : 18 ans, 22 ans, 27 ans.

9.1.2 Instruction `if-else`

Dans notre exemple, si l'individu est âgé de 13 ans ou plus, nous lui retournons une information. Dans le cas contraire, aucune action n'est menée. Si nous souhaitons réaliser une action dans les deux cas, que notre individu soit âgé de moins de 13 ans ou qu'il ait 13 ans ou plus, nous pouvons utiliser une instruction `if-else`. La syntaxe est la suivante :

```
if(instruction_conditionnelle) {
  # Si instruction_conditionnelle retourne TRUE
  action_1
  action_2
  ...
  action_n
}else {
  # Sinon
  action_a
  action_b
  ...
  action_x
}
```

Pour notre individu, s'il est âgé de 13 ans ou plus, nous lui afficherons son âge au carré et le message d'information. Si en revanche l'individu est trop jeune, nous lui indiquerons un autre message. Admettons que l'utilisateur soit âgé de 22 ans :

```
age <- 22
```

Notre instructions `if-else` sera :

```
if(age >= 13){
  age_carre <- age^2
  print(paste("Votre age au carré vaut :", age_carre))
  print("La fée des dents n'existe pas")
}else{
```

```
  print("La fée des dents passera-t-elle cette nuit ?")
}
```

```
## [1] "Votre age au carré vaut : 484"
## [1] "La fée des dents n'existe pas"
```

Seules les instructions dans le bloc de code correspondant aux actions à conduire si l'âge est supérieur ou égal à 13 sont évaluées. Si à présent notre individu est âgé de 5 ans :

```
age <- 5
```

L'instruction logique `age >= 13` retournera `FALSE`. Les actions à mener si l'instruction logique retourne `TRUE` ne seront pas évaluées. En revanche, l'instruction à conduire dans le cas où l'instruction logique retourne `FALSE` sera bien évaluée :

```
if(age >= 13){
  age_carre <- age^2
  print(paste("Votre age au carré vaut :", age_carre))
  print("La fée des dents n'existe pas")
}else{
  print("La fée des dents passera-t-elle cette nuit ?")
}
```

```
## [1] "La fée des dents passera-t-elle cette nuit ?"
```

Il est possible d'imbriquer des conditions `if` ou `if-else` à l'intérieure de conditions `if` et `if-else`. Par

```
age <- 22

if(age >= 13){

  if(age >= 18){
    print("Vous êtes majeur(e)")
  }

  age_carre <- age^2
  print(paste("Votre age au carré vaut :", age_carre))
  print("La fée des dents n'existe pas")
}else{
```

```
  if(age < 7){
    print("Vous n'avez pas atteint l'âge de raison")
  }else{
    print("Vous avez atteint l'âge de raison")
  }

  print("La fée des dents passera-t-elle cette nuit ?")
}
```

```
## [1] "Vous êtes majeur(e)"
## [1] "Votre age au carré vaut : 484"
## [1] "La fée des dents n'existe pas"
```

Dans le code précédent :

- Si la valeur de l'objet `age` est supérieure ou égale à 13 :
  - Si la valeur de l'objet `age` est supérieure ou égale à 18 : le message "Vous êtes majeur(e)" s'affiche
  - Si la valeur de l'objet `age` est n'est pas supérieure ou égale à 18, le message "Vous êtes majeur(e)" ne s'affiche pas
  - Que la valeur de l'objet `age` soit supérieure ou égal à 18 ou non, du moment qu'elle est supérieure ou égale à 13, le carré de l'âge est calculé et les deux messages indiquant la valeur de l'âge au carré et la terrible vérité au sujet de l'existance de la fée des dents s'affichent.
- Si la valeur de l'objet `age` n'est pas supérieure ou égale à 13 :
  - Si de surcroît la valeur de l'objet `age` est inférieure à 7 : le message "Vous n'avez pas atteint l'âge de raison" s'affiche
  - Si en revanche la valeur de l'objet `age` n'est pas inférieure à 7 (mais bien inférieure à 13) : le message "Vous avez atteint l'âge de raison" s'affiche.
  - Que la valeur de l'objet `age` soit inférieure ou non à 7 (du moment qu'elle est inférieure à 13), le message "La fée des dents passera-t-elle cette nuit ?" s'affiche.

1. Demandez à l'utilisateur de renseigner sa moyenne dans les deux Éléments Constitutifs de l'Unité d'Enseignement d'Economie et Finance internationales :

```
finance_inter <-
  readline(prompt="Votre moyenne en Finance internationale : ")
eco_inter <-
  readline(prompt="Votre moyenne en Economie internationale : ")
```

2. Si la moyenne à l'Unité d'Enseignement est supérieure ou égale à 10, affichez dans la console le message suivant : "Vous validez l'UE". Si en revanche la moyenne est inférieure à 10, affichez dans la console le message "Vous ne validez pas l'UE".
3. Reprenez le code de la question précédente, et ajoutez la fonctionnalité suivantes :
   — Si la personne ne valide pas l'Unité d'Enseignement (moyenne aux deux épreuves inférieure à 10), indiquez-lui quelles sont les matières à repasser (uniquement celles pour lesquelles la note est strictement inférieure à 10).

Il est possible de vérifier successivement des instructions conditionnelles et d'effectuer une action dès lors qu'une des instructions est évaluée à `TRUE`. Pour notre exemple de message à afficher en fonction de l'âge de l'utilisateur, on peut dans un premier temps regarder si l'individu a moins de 13 ans et lui adresser un message adapté ; puis, s'il n'a pas moins de 13 ans, s'il a moins de 18 ans et lui adresser un autre message ; et enfin, s'il n'a pas moins de 18, lui retourner un message encore différent. La syntaxe de ce type d'expression sera la suivante :

```
if(instruction_conditionnelle_1){
  actions_1
}else if(instruction_conditionnelle_2){
  actions_2
}else{
  actions_3
}
```

Si notre individu a 7 ans :

```
age <- 7

if(age < 13){
  print(paste("L'âge que vous avez indiqué :", age, "ans"))
  print("L'âge minimum pour s'inscrire est de 13 ans.")
```

```
}else if(age < 18){
  print(paste("L'âge que vous avez indiqué :", age, "ans"))
  print("Contenus sensibles masqués.")
}else{
  print(paste("L'âge que vous avez indiqué :", age, "ans"))
  print("Contenus sensibles non masqués.")
}
```

```
## [1] "L'âge que vous avez indiqué : 7 ans"
## [1] "L'âge minimum pour s'inscrire est de 13 ans."
```

La valeur de l'objet `age` étant 7, l'instruction `age < 13` retourne `TRUE` et seules les instructions à l'intérieur du premier bloc défini par les accolades sont exécutées.

Si notre individu a 22 ans :

```
age <- 22

if(age <= 13){
  print(paste("L'âge que vous avez indiqué :", age, "ans"))
  print("L'âge minimum pour s'inscrire est de 13 ans.")
}else if(age < 18){
  print(paste("L'âge que vous avez indiqué :", age, "ans"))
  print("Contenus sensibles masqués.")
}else{
  print(paste("L'âge que vous avez indiqué :", age, "ans"))
  print("Contenus sensibles non masqués.")
}
```

```
## [1] "L'âge que vous avez indiqué : 22 ans"
## [1] "Contenus sensibles non masqués."
```

La valeur de l'objet `age` étant 7, l'instruction `age < 13` retourne `FALSE`. Les instructions à l'intérieur du premier bloc défini par les accolades ne sont pas exécutées. La deuxième instruction conditionnelle, `age < 18`, est évaluée et retourne `FALSE`. Les instructions à l'intérieur du deuxième bloc défini par les accolades ne sont pas exécutées. Il n'y a pas d'autres instruction conditionnelle avant le mot clé `else`, donc le code à l'intérieur du bloc défini par les accolades associées à `else` est évalué.

9.2 Boucles

Il existe deux sortes de boucles dans R. Celles pour lesquelles les itérations continuent tant qu'une condition n'est pas invalidée (`while()`), et celles pour lesquelles le nombre d'itérations est défini au moment de lancer la boucle (`for()`).

9.2.1 Boucles `while()`

Les codes à l'intérieur d'une boucle `while` seront évaluées de manière itérative. Avant chaque itération, une instruction conditionnelle est évaluée. Si cette instruction retourne le logique `TRUE`, le code est évalué. Puis débute une nouvelle itération. Si en revanche l'instruction conditionnelle retourne le logique `FALSE`, le code n'est pas évalué et le processus itératif cesse. Le schéma de la Figure 9.1 illustre le fonctionnement d'une boucle `while()`.

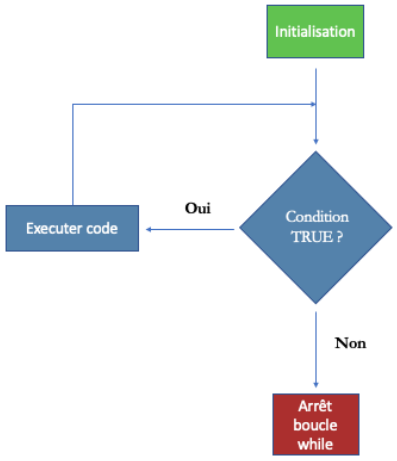


Figure 9.1 – Fonctionnement d'une boucle while.

La syntaxte est la suivante :

```
while(condition) instruction
```

Ou encore, si plusieurs instructions doivent être évaluées à chaque itération :

```
while(condition){
  instruction_1
  instruction_2
  ...
  instruction_n
```

```
}
```

Prenons un exemple. Soit un entier `x` contenant une valeur qui peut varier. Admettons que cette valeur vaille 50 :

```
x <- 50
```

Créons une boucle `while` dans laquelle chaque itération vient diviser la valeur de `x` par 3, puis remplace l'ancienne de valeur de `x` par le résultat de la division :

```
while(x / 3 > 1){
  print(x / 3)
  x <- x/3
}
```

```
## [1] 16.66667
## [1] 5.555556
## [1] 1.851852
```

Regardons pas-à-pas les étapes du processus itératif précédent. Lors de l'initialisation, la valeur de `x` vaut 50.
L'instruction conditionnelle `x / 3 > 1` est évaluée et retourne le logique `TRUE`. En effet, $50/3 \sim 16.66667$ est bien strictement supérieur à 1.

- La *première itération* commence.
- Le bloc de code de la boucle, défini par les accolades, peut alors être évalué.
- La valeur `x/3` est affichée avec la fonction `print()` : la valeur `16.66667` s'affiche dans la console.
- Puis, la valeur de `x` est remplacée par le tiers de sa valeur (`x <- x/3`).
- La nouvelle valeur de `x` est donc `16.66667`.
- La première itération s'achève.

Ensuite, l'instruction conditionnelle `x / 3 > 1` est évaluée et retourne le logique `TRUE`. En effet, $16.66667/3 \sim 5.555556$ est bien strictement supérieur à 1.

- La *deuxième itération* commence.
- Le bloc de code de la boucle, défini par les accolades, peut alors être évalué.
- La valeur `x/3` est affichée avec la fonction `print()` : la valeur `5.555556` s'affiche dans la console.
- Puis, la valeur de `x` est remplacée par le tiers de sa valeur (`x <- x/3`).
- La nouvelle valeur de `x` est donc `5.555556`.
- La deuxième itération s'achève.

L'instruction conditionnelle `x / 3 > 1` est évaluée et retourne le logique `TRUE`. En effet, $5.555556/3 \sim 1.851852$ est bien strictement supérieur à 1.

- La *troisième itération* commence.
- Le bloc de code de la boucle, défini par les accolades, peut alors être évalué.
- La valeur `x/3` est affichée avec la fonction `print()` : la valeur `1.851852` s'affiche dans la console.
- Puis, la valeur de `x` est remplacée par le tiers de sa valeur (`x <- x/3`).
- La nouvelle valeur de `x` est donc `1.851852`.
- La troisième itération s'achève.

L'instruction conditionnelle `x / 3 > 1` est évaluée et retourne le logique `FALSE`. En effet, $1.851852/3 \sim 0.617284$ et n'est donc pas strictement supérieur à 1. Le processus itératif s'arrête. La valeur contenue dans `x` est donc `1.851852`.

Il n'existe pas qu'une seule manière de rédiger des boucles. Il est parfois utile, avec une boucle `while`, de créer une variable (que l'on peut appeler `continuer` par exemple) contenant une valeur logique `TRUE` et d'utiliser cette variable comme instruction conditionnelle pour continuer les itérations. Le code à l'intérieur de la boucle se chargera de venir changer la valeur de cette variable au moment opportun. Par exemple, pour obtenir le même résultat dans `x` que dans l'exemple précédent, on pourra écrire :

```
x <- 50
continuer <- (x / 3 > 1)

while(continuer){
  print(x / 3)

  if(x / 3 < 1){
    continuer <- FALSE
  }else{
    x <- x/3
  }
}
```

```
## [1] 16.66667
## [1] 5.555556
## [1] 1.851852
## [1] 0.617284
```

```
x
```

```
## [1] 1.851852
```

Pour éviter de créer la variable `continuer`, nous pouvons utiliser le mot clé `break` à l'intérieur de la boucle pour stopper la boucle au moment opportun.

```
x <- 50
while(TRUE){
  print(x / 3)

  if(x / 3 < 1){
    break
  }else{
    x <- x/3
  }
}
```

```
## [1] 16.66667
## [1] 5.555556
## [1] 1.851852
## [1] 0.617284
```

```
x
```

```
## [1] 1.851852
```

Exercice
À l'aide de la fonction `while()`, créez une boucle qui permet de calculer la factorielle d'un nombre.

Exercice bonus
À l'aide d'une boucle `while`, créez un code dans lequel vous devrez tenter de faire deviner un nombre magique tiré aléatoirement entre 1 et 100.
Dans une première étape, un nombre magique est tiré aléatoirement :

```
nombre_magique <- sample(x=1:100, size = 1)
```

Votre code doit demander à l'utilisateur d'entrer une réponse :

```
tentative <-
  readline(prompt="Nombre entier compris entre 1 et 100 : ")
```

Vous devez ensuite indiquer, à l'aide d'instructions conditionnelles si le nombre donné par l'utilisateur est inférieur, supérieur, ou égal au nombre magique.

9.2.2 Boucles `for()`

Lorsqu'on connaît *a priori* le nombre d'itérations que l'on souhaite faire, il est coutume de faire appel à une boucle `for()`. On définit un itérateur qui prendra successivement des valeurs dans un ensemble donné jusqu'à ce que toutes les valeurs de l'ensemble auront été utilisées comme itérateur. La syntaxe est la suivante :

```
for(iterateur in ensemble){
  instruction_1
  instruction_2
  ...
  instruction_n
}
```

Le schéma de la Figure 9.2 illustre le fonctionnement d'une boucle `for()`.

Illustrons par ailleurs le fonctionnement à l'aide d'un exemple simple. Considérons un itérateur que l'on nommera `i` et l'ensemble des entiers naturels de 1 à 10. À chaque itération, `i` prendra une des valeurs de l'ensemble :

```
for(i in 1:10){
  carre <- i^2
  print(paste("Le carré de", i, "est :", carre))
}
```

```
## [1] "Le carré de 1 est : 1"
## [1] "Le carré de 2 est : 4"
## [1] "Le carré de 3 est : 9"
## [1] "Le carré de 4 est : 16"
## [1] "Le carré de 5 est : 25"
## [1] "Le carré de 6 est : 36"
```

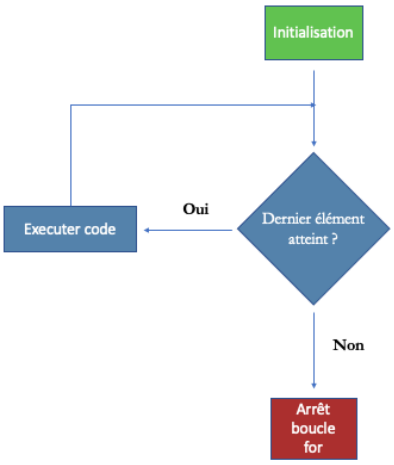


Figure 9.2 – Fonctionnement d'une boucle for

```
## [1] "Le carré de 7 est : 49"
## [1] "Le carré de 8 est : 64"
## [1] "Le carré de 9 est : 81"
## [1] "Le carré de 10 est : 100"
```

Regardons en détail ce qu'il s'est passé lors de l'éxécution de ce bout de code.

- À la *première itération*, l'objet `i` est créé et la valeur `1` lui est affectée (si un objet `i` existait déjà, sa valeur eut été remplacée par la valeur `1`).
- L'objet `carre` est créé et le carré de `i`, c'est à dire $1^2 = 1$ lui est affecté.
- La chaîne de caractère indiquant la valeur de `i` et de son carré est ensuite affichée dans la console.
- La première itération prend fin.
- L'ensemble des valeurs des entiers de 1 à 10 n'a pas encore été utilisé lors du processus itératif ; ce dernier se poursuit donc.
- La *deuxième itération* débute, et la valeur de l'objet `i` devient la valeur suivante de l'ensemble `1:10`, c'est-à-dire 2.
- La valeur du carré de `i`, c'est-à-dire $2^2 = 4$ est affectée à l'objet `carré`.
- La chaîne de caractère indiquant la valeur de `i` et de son carré est ensuite affichée dans la console.
- La deuxième itération prend fin.
- L'ensemble des valeurs des entiers de 1 à 10 n'a pas encore été utilisé lors du processus itératif ; ce dernier se poursuit donc.
- ...

— La *dixième itération* débute, et la valeur de l'objet `i` devient la valeur suivante de l'ensemble `1:10`, c'est-à-dire 10.
— La valeur du carré de `i`, c'est-à-dire $10^2 = 100$ est affectée à l'objet `carré`.
— La chaîne de caractère indiquant la valeur de `i` et de son carré est ensuite affichée dans la console.
— La dixième itération prend fin.
— L'ensemble des valeurs des entiers de 1 à 10 a été utilisé lors du processus itératif ; ce dernier s'achève.

Note
À l'issue des itérations, l'objet `i` contient la dernière valeur de l'ensemble.

Note
Nous pouvons utiliser une boucle `while()` pour obtenir le même résultat. Le code est un tout petit moins lisible.

```
i <- 0
while(i < 10){
  i <- i+1
  carre <- i^2
  print(paste("Le carré de", i, "est :", carre))
}
```

```
## [1] "Le carré de 1 est : 1"
## [1] "Le carré de 2 est : 4"
## [1] "Le carré de 3 est : 9"
## [1] "Le carré de 4 est : 16"
## [1] "Le carré de 5 est : 25"
## [1] "Le carré de 6 est : 36"
## [1] "Le carré de 7 est : 49"
## [1] "Le carré de 8 est : 64"
## [1] "Le carré de 9 est : 81"
## [1] "Le carré de 10 est : 100"
```

L'ensemble de valeurs n'est pas nécessairement composé d'entiers. Il peut s'agir de n'importe quel élément contenu dans une structure sur laquelle il est possible d'itérer :

```
for(i in c("A", "B", "C")){
  print(paste("La valeur courante de i vaut :", i))
```

```
}
```

```
## [1] "La valeur courante de i vaut : A"
## [1] "La valeur courante de i vaut : B"
## [1] "La valeur courante de i vaut : C"
```

```
for(i in list(toto=c(1,2), titi=c("A", "B"))){
  print(i)
}
```

```
## [1] 1 2
## [1] "A" "B"
```

Exercice
Si vous avez quelques interrogations quant à l'utilité des boucles dans le cadre de vos études en économie, cet exercice devrait vous permettre d'apprécier le gain de temps potentiel que celles-ci peuvent vous faire gagner.
Il s'agit d'importer une multitude de fichiers CSV (en l'occurence, 35) présentant une structure similaire. Ces fichiers contiennent des observations de commptage de vélos à des points spécifiques de la ville de Paris, en 2018. Chaque fichier correspond à une localisation donnée, la localisation est utilisée comme nom de fichier.
Les 35 fichiers CSV sont contenus dans une archive Zip disponible à l'adresse suivante : https://egallic.fr/Enseignement/R/Exercices/Boucles/velo.zip. (*Note* : Les données brutes sont disponible sur le site de l'open data de la ville de Paris.)

1. Téléchargez l'archive contenant les 35 fichiers CSV, puis extrayez son contenu dans le dossier `Data` de votre répertoire de projet.
2. Créez un objet que vous nommerez `df_velo` qui contiendra la valeur `NULL`.
3. Importez le fichier `Voie_Georges_Pompidou.csv` dans R, dans un objet que vous nommerez `df_tmp`.
4. Utilisez la fonction `bind_rows()` pour empiler les données du tableau `df_tmp` sur `df_velo`. Le résultat doit venir écraser l'objet `df_velo` pour que son contenu devienne le résultat de l'application de la fonction `bind_rows()`. Le tableau `df_velo` contiendra alors 17 473 observations.
5. Importez le fichier `106_Avenue_Denfert_Rochereau_NE-SO.csv` dans R, dans un objet que vous nommerez `df_tmp`.
6. Utilisez la fonction `bind_rows()` pour empiler les données du tableau `df_tmp` sur `df_velo`. Le résultat doit venir écraser l'objet `df_velo` pour que son contenu devienne le résultat de l'application de la fonction `bind_rows()`. Votre

tableau `df_velo` contiendra alors les 17 473 observations du premier fichier et les 8 737 du deuxième, soit 26 210 observations.

7. Importez le fichier `28_Boulevard_Diderot.csv` dans R, dans un objet que vous nommerez `df_tmp`.
8. Utilisez la fonction `bind_rows()` pour empiler les données du tableau `df_tmp` sur `df_velo`. Le résultat doit venir écraser l'objet `df_velo` pour que son contenu devienne le résultat de l'application de la fonction `bind_rows()`. Votre tableau `df_velo` contiendra alors les 17 473 observations du premier fichier, les 8 737 du deuxième, et les 17 473 du troisième, soit 43 684 observations.
9. Prenez le temps d'imginer à quel point il serait pénible d'empiler les 35 fichiers.

À présent, tentons de réfléchir à l'utilisation d'une boucle pour effectuer la tâche. Les premières questions devraient vous laisser apercevoir que vous avez réalisé une tâche répétitive qui consiste à chaque itération à charger un fichier et à ajouter son contenu à un tableau existant. Vous avez donc besoin dans un premier temps de récupérer dans un vecteur le lien de chacun des fichiers. Pour ce faire, vous allez utiliser la fonction `list.files()`, qui permet de lister le contenu d'un répertoire.

```
N <- list.files("Data/velo/", full.names = TRUE)
N
```

```
##  [1] "Data/velo//10_avenue_de_la_Grande_Arme´ e_SE-NO.csv"
##  [2] "Data/velo//10_Bd_Auguste_Blanqui_NE-SO.csv"
##  [3] "Data/velo//100_Rue_La_Fayette_O-E.csv"
##  [4] "Data/velo//102_Bd_magenta_SE-NO.csv"
##  [5] "Data/velo//105_Rue_La_Fayette_E-O.csv"
##  [6] "Data/velo//106_Avenue_Denfert_Rochereau_NE-SO.csv"
##  [7] "Data/velo//135_Avenue_Daumesnil_SE-NO.csv"
##  [8] "Data/velo//152_Boulevard_du_Montparnasse.csv"
##  [9] "Data/velo//16_Avenue_de_la_Porte_des_Ternes_E-O.csv"
## [10] "Data/velo//18_quai_de_l'hotel_de_ville.csv"
## [11] "Data/velo//21_Boulevard_Auguste_Blanqui_SO-NE.csv"
## [12] "Data/velo//243_Boulevard_Saint_Germain_NO-SE.csv"
## [13] "Data/velo//26_Boulevard_de_Me´ nilmontant_SE-NO.csv"
## [14] "Data/velo//27_Boulevard_Diderot_E-O.csv"
## [15] "Data/velo//28_Boulevard_Diderot.csv"
## [16] "Data/velo//35_Boulevard_de_Menilmontant_NO-SE.csv"
## [17] "Data/velo//39_Quai_Francois_Mauriac.csv"
## [18] "Data/velo//6_rue_Julia_Bartet.csv"
## [19] "Data/velo//67_boulevard_Voltaire_SE-NO.csv"
## [20] "Data/velo//7_Avenue_de_la_Grande_Arme´ e_NO-SE.csv"
## [21] "Data/velo//72_Bd_Richard_Lenoir_S-N.csv"
## [22] "Data/velo//72_boulevard_Voltaire_NO-SE.csv"
```

```
## [23] "Data/velo//77_Bd_Richard_Lenoir_N-S.csv"
## [24] "Data/velo//89_boulevard_de_Magenta_NO-SE.csv"
## [25] "Data/velo//97_Avenue_Denfert_Rochereau_SO-NE.csv"
## [26] "Data/velo//Face_104_Rue_d'Aubervilliers.csv"
## [27] "Data/velo//Face_au_16_Avenue_de_la_Porte_des_Ternes_O-E.
   csv"
## [28] "Data/velo//Face_au_25_quai_de_l'Oise.csv"
## [29] "Data/velo//Face_au_48_quai_de_la_marne.csv"
## [30] "Data/velo//face_au_70_Quai_de_Bercy.csv"
## [31] "Data/velo//Pont_du_Garigliano_NO-SE.csv"
## [32] "Data/velo//Pont_du_Garigliano_SE-NO.csv"
## [33] "Data/velo//Pont_National_NE-SO.csv"
## [34] "Data/velo//Pont_National_SO-NE.csv"
## [35] "Data/velo//Voie_Georges_Pompidou.csv"
```

L'objet `N` consistera votre ensemble sur lequel itérer.

10. À l'aide d'une boucle `for()`, chargez l'ensemble des 35 fichiers CSV du répertoire `velo` et placez leur contenu dans un tableau de données que vous nommerez `df_velo`.

Exercice bonus
Utilisez une boucle `for()` pour reproduire la suite de Fibonacci jusqu'à son dixième terme (la séquence F_n est définie par la relation de récurrence suivante : $F_n = F_n - 1 + F_n - 2$; les valeurs initiales sont : $F_0 = 0$ et $F_1 = 1$).

Des compléments sur les fonctions de type `lapply()` ou `map()` vous permettront de réaliser des opérations répétées. Nous n'aurons malheureusement pas le temps de les aborder dans le cadre de ces travaux dirigés. En revanche, vous êtes invité(e) à consulter la section consacrée à la vectorisation sur les notes de cours.

Chapitre 10

References

Athey, Susan, and Guido W. Imbens. 2017. "The State of Applied Econometrics : Causality and Policy Evaluation." *Journal of Economic Perspectives* 31 (2) : 3–32. https://doi.org/10.1257/jep.31.2.3.

Bertrand, Marianne, and Sendhil Mullainathan. 2004. "Are Emily and Greg More Employable Than Lakisha and Jamal ? A Field Experiment on Labor Market Discrimination." *American Economic Review* 94 (4) : 991–1013. https://doi.org/10.1257/0002828042002561.

Xie, Yihui, Joseph J Allaire, and Garrett Grolemund. 2018. *R Markdown : The Definitive Guide*. Chapman ; Hall/CRC.