guide-R

Guide pour l'analyse de données d'enquêtes avec R

Joseph Larmarange

2022-09-10T00:00:00+02:00

Table des matières

Préface

> **Site en construction**
>
> Le présent site est en cours de construction et sera complété dans les prochains mois.
> En attendant, nous vous conseillons de consulter le site analyse-R.

Ce guide porte sur l'analyse de données d'enquêtes avec le logiciel R, un logiciel libre de statitistiques et de traitement de données. Les exemples présentés ici relèvent principalement du champs des sciences sociales quantitatives et des sciences de santé. Ils peuvent néanmoins s'appliquer à d'autre champs disciplinaires. Cependant, comme tout ouvrage, ce guide ne peut être exhaustif.

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

De même, il ne s'agit pas d'une introduction ou d'un guide pour les utilisatrices et utilisateurs débutant · es. Si vous découvrez **R**, nous vous conseillons la lecture de l'*Introduction à R et au tidyverse* de Julien Barnier (https://juba.github.io/tidyverse/). Vous pouvez également lire les chapitres introductifs d'*analyse-R : Introduction à l'analyse d'enquêtes avec R et RStudio* (https://larmarange.github.io/analyse-R/).

Néanmoins, quelques rappels sur les bases du langage sont fournis dans la section *Rappels*.

R disposent de nombreuses extensions ou packages (plus de 16000) et il existe souvent plusieurs manières de procéder pour arriver au même résultat. En particulier, en matière de manipulation de données, on oppose[1] souvent *base R* qui repose sur les fonctions disponibles en standard dans **R**, la majorité étant fournies dans les packages `{base}`, `{utils}` ou encore `{stats}`, qui sont toujours chargés par défaut, et le `{tidyverse}` qui est une collection de packages comprenant, entre autres, `{dplyr}`, `{tibble}`, `{tidyr}`, `{forcats}` ou encore `{ggplot2}`. Il y a un débat ouvert, parfois passionné, sur le fait de privilégier l'une ou l'autre approche, et les avantages et inconvénients de chacune dépendent de nombreux facteurs, comme la lisibilité du code ou bien les performances en temps de calcul. Dans ce guide, nous avons adopté un point de vue pragmatique et utiliserons, le plus souvent mais pas exclusivement, les fonctions du `{tidyverse}`, de même que nous avons privilégié d'autres packages, comme `{gtsummary}` ou `{questionr}` par exemple pour la statistique descriptive. Cela ne signifie pas, pour chaque point abordé, qu'il s'agit de l'unique manière de procéder. Dans certains cas, il s'agit simplement de préférences personnelles.

[1] Une comparaison des deux syntaxes est illustrée par une vignette dédiée de dplyr.

Bien qu'il en reprenne de nombreux contenus, ce guide ne se substitue pas au site analyse-R. Il s'agit plutôt d'une version complémentaire qui a vocation à être plus structurée et parfois plus sélective dans les contenus présentés.

En complément, on pourra également se référer aux webin-R, une série de vidéos avec partage d'écran, librement accessibles sur Youtube : https://www.youtube.com/c/webinR.

Cette version du guide a utilisé *R version 4.1.3 (2022-03-10)*. Ce document est généré avec quarto et le code source est disponible sur GitHub. Pour toute suggestion ou correction, vous pouvez ouvrir un ticket GitHub. Pour d'autres questions, vous pouvez utiliser les forums de discussion disponibles en bas de chaque page sur la version web du guide. Ce document est régulièrement mis à jour. La dernière version est consultable sur https://larmarange.github.io/guide-R/.

Remerciements

Ce document a bénéficié de différents apports provenant notamment de l'*Introduction à R* et de l'*Introduction à R et au tidyverse* de Julien Barnier et d'analyse-R : introduction à l'analyse d'enquêtes avec R et RStudio.

Merci donc à Julien Barnier, Julien Biaudet, François Briatte, Milan Bouchet-Valat, Ewen Gallic, Frédérique Giraud, Joël Gombin, Mayeul Kauffmann, Christophe Lalanne & Nicolas Robette.

Licence

partie I

Rappels

Même lorsque l'on a déjà une bonne expérience de **R**, il est souvent utile de réviser les bases du language.

Une bonne compréhension des bases du langage, bien qu'un peu ardue de prime abord, permet de comprendre le sens des commandes qu'on utilise et de pleinement exploiter la puissance que **R** offre en matière de manipulation de données.

1 Vecteurs, indexation et assignation

Les vecteurs sont l'objet de base de **R** et correspondent à une liste de valeurs. Leurs propriétés fondamentales sont :

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

1.1 Types et classes

Dans **R**, il existe plusieurs types fondamentaux de vecteurs et, en particulier, :

- les nombres réels (c'est-à-dire les nombres décimaux[2]), par exemple `5.23` ;
- les nombres entiers, que l'on saisi en ajoutant le suffixe `L`[3], par exemple `4L` ;
- les chaînes de caractères (qui correspondent à du texte), que l'on saisit avec des guillemets doubles (") ou simples (’), par exemple `"abc"` ;
- les valeurs logiques ou valeurs booléennes, à savoir vrai ou faux, que l'on représente avec les mots `TRUE` et `FALSE` (en majuscules[4]).

En plus de ces types de base, il existe de nombreux autres types de vecteurs utilisés pour représenter toutes sortes de données, comme les facteurs (voir Chapitre 2) ou les dates (voir Chapitre 3).

[2] Pour rappel, **R** étant anglophone, le caractère utilisé pour indiqué les chiffres après la virgule est le point (`.`).

[3] **R** utilise 32 bits pour représenter des nombres entiers, ce qui correspond en informatique à des entiers longs ou *long integers* en anglais, d'où la lettre `L` utilisée pour indiquer un nombre entier.

[4] On peut également utiliser les raccourcis `T` et `F`. Cependant, pour une meilleure lisibilité du code, il est préférable d'utiliser les versions longues `TRUE` et `FALSE`.

La fonction `class()` renvoie la nature d'un vecteur tandis que la fonction `typeof()` indique la manière dont un vecteur est stocké de manière interne par **R**.

Table 1.1: Le type et la classe des principaux types de vecteurs

x	class(x)	typeof(x)
3L	integer	integer
5.3	numeric	double
TRUE	logical	logical
"abc"	character	character
factor("a")	factor	integer
as.Date("2020-01-01")	Date	double

> **Astuce**
>
> Pour un vecteur numérique, le type est `"double"` car **R** utilise une double précision pour stocker informatiquement les nombres réels.
> En interne, les facteurs sont représentés par un nombre entier auquel est attaché une étiquette, c'est pourquoi `typeof()` renvoie `"integer"`.
> Quand aux dates, elles sont stockées en interne sous la forme d'un nombre réel représentant le nombre de jours depuis le 1er janvier 1970, d'où le fait que `typeof()` renvoie `"double"`.

1.2 Création d'un vecteur

Pour créer un vecteur, on utilisera la fonction `c()` en lui passant la liste des valeurs à combiner[5].

[5] La lettre `c` est un raccourci du mot anglais *combine*, puisque cette fonction permet de combiner des valeurs individuelles dans un vecteur unique.

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

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

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

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

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

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

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

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

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

```
class(x)
```

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

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

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

```
rep(2, 10)
```

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

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

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

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

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

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

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

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

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

```
seq(10, 0)
```

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

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

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

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

```
 [1] 1.23 1.56 1.89 2.22 2.55 2.88 3.21 3.54 3.87 4.20 4.53 4.86 5.19 5.52
```

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

```
1:5
```

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

```
24:32
```

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

```
55:43
```

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

1.3 Longueur d'un vecteur

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

```
length(taille)
```

```
[1] 4
```

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

```
[1] 2
```

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

```
length(c())
```

```
[1] 0
```

1.4 Combiner des vecteurs

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

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

```
[1] 4
```

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

```
[1] 6
```

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

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

```
length(z)
```

```
[1] 10
```

1.5 Valeurs manquantes

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

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

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

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

```
length(taille)
```

```
[1] 6
```

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

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

```
c(NULL, NULL, NULL)
```

```
NULL
```

```
length(c(NULL, NULL, NULL))
```

```
[1] 0
```

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

```
c(NA, NA, NA)
```

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

```
length(c(NA, NA, NA))
```

```
[1] 3
```

1.6 Indexation par position

L'indexation est l'une des fonctionnalités les plus puissantes mais aussi les plus difficiles à maîtriser de **R**. Il s'agit d'opérations permettant de sélectionner des sous-ensembles de valeurs en fonction de différents critères. Il existe trois types d'indexation : (i) l'indexation par position, (ii) l'indexation par nom et (iii) l'indexation par condition. Le principe est toujours le même : on indique entre crochets[6] (`[]`) ce qu'on souhaite garder ou non.

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

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

Reprenons notre vecteur `taille` :

```
taille
```

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

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

```
taille[1]
```

```
[1] 1.88
```

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

```
taille[1:3]
```

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

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

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

Si on veut le dernier élément :

```
taille[length(taille)]
```

```
[1] NA
```

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

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

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

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

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

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

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

```
taille[23:25]
```

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

1.7 Des vecteurs nommés

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

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

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

```
sexe
```

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

La liste des noms s'obtient avec `names()`.

```
names(sexe)
```

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

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

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

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

Pour supprimer tous les noms, il y a la fonction `unname()` :

```
anonyme <- unname(sexe)
anonyme
```

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

1.8 Indexation par nom

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

```
sexe["Anna"]
```

```
Anna
 "f"
```

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

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

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

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

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

1.9 Indexation par condition

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

```
sexe
```

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

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

```
Michael    John
    "h"     "h"
```

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

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

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

```
sexe[urbain]
```

```
Michael    Alex    Mary
    "h"      NA     "f"
```

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

Opérateur de comparaison	Signification
`==`	égal à
`!=`	différent de
`>`	strictement supérieur à
`<`	strictement inférieur à
`>=`	supérieur ou égal à
`<=`	inférieur ou égal à

Voyons tout de suite un exemple :

```
poids >= 80
```

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

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

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

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

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

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

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

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

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

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

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

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

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

```
taille
```

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

```
taille > 1.8
```

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

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

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

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

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

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

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

```
taille
```

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

```
poids
```

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

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

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

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

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

```
[1] 80 87
```

1.10 Assignation par indexation

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

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

Prenons donc un exemple simple :

```
v <- 1:5
v
```

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

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

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

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

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

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

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

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

```
length(sexe)
```

```
[1] 6
```

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

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

```
length(sexe)
```

```
[1] 7
```

1.11 En résumé

- Un vecteur est un objet unidimensionnel contenant une liste de valeurs qui sont toutes du même type (entières, numériques, textuelles ou logiques).
- La fonction `class()` permet de connaître le type du vecteur et la fonction `length()` sa longueur, c'est-à-dire son nombre d'éléments.
- La fonction `c()` sert à créer et à combiner des vecteurs.
- Les valeurs manquantes sont représentées avec `NA`.
- Un vecteur peut être nommé, c'est-à-dire qu'un nom textuel a été associé à chaque élément. Cela peut se faire lors de sa création ou avec la fonction `names()`.

- L'indexation consiste à extraire certains éléments d'un vecteur. Pour cela, on indique ce qu'on souhaite extraire entre crochets (`[]`) juste après le nom du vecteur. Le type d'indexation dépend du type d'information transmise.
- S'il s'agit de nombres entiers, c'est l'indexation par position : les nombres représentent la position dans le vecteur des éléments qu'on souhaite extraire. Un nombre négatif s'interprète comme tous les éléments sauf celui-là.
- Si on indique des chaînes de caractères, c'est l'indexation par nom : on indique le nom des éléments qu'on souhaite extraire. Cette forme d'indexation ne fonctionne que si le vecteur est nommé.
- Si on transmet des valeurs logiques, le plus souvent sous la forme d'une condition, c'est l'indexation par condition : `TRUE` indique les éléments à extraire et `FALSE` les éléments à exclure. Il faut être vigilant aux valeurs manquantes (`NA`) dans ce cas précis.
- Enfin, il est possible de ne modifier que certains éléments d'un vecteur en ayant recours à la fois à l'indexation (`[]`) et à l'assignation (`<-`).

1.12 webin-R

On pourra également se référer au webin-R #02 (*les bases du langage R*) sur YouTube.

https://youtu.be/Eh8piunoqQc

2 Facteurs

partie II

Manipulation de données

3 Dates avec `lubridate`