

Dates 1: Reconnaître et Savoir Formatter des Dates

GRAPH Network & WHO, supported by the Global Fund to fight HIV, TB & Malaria

October 2023

This document is a draft of a lesson made by the GRAPH Network, a non-profit headquartered at the University of Geneva Global Health Institute, in collaboration with the World Health Organization, under a Global Fund 2023 grant to create e-learning modules to build in-country data capacity for epidemiological and impact analysis for National HIV, TB and malaria programs

Introduction
Learning Objectives
Packages
Datasets
PID Malawi
Séjours hospitaliers
Introduction aux Dates en R
Conversion de chaînes de caractères en dates
Avec les fonctions natives de R (Base R)
Avec lubridate
Gérer des dates mixtes avec <code>lubridate:::parse_date_time()</code>
Modifier l'Affichage des Dates
EN RÉSUMÉ
Answer Key

```
## [1] "fr_FR.UTF-8"
```

Introduction

Comprendre comment manipuler les dates est une compétence cruciale lorsque l'on travaille avec des données de santé. Les calendriers de vaccination, la surveillance des maladies, et les changements dans les indicateurs de santé à l'échelle de la population nécessitent tous de travailler avec des dates. Dans cette leçon, nous allons apprendre comment R stocke et affiche les dates, ainsi que comment les manipuler, les analyser et les formater efficacement. Commençons !

Learning Objectives

- Vous comprenez comment les dates sont stockées et manipulées dans R
- Vous comprenez comment convertir des chaînes de caractères en dates
- Vous savez gérer les colonnes de dates de formats mixtes
- Vous êtes capable de changer l'affichage des dates

Packages

Veuillez charger les packages nécessaires pour cette leçon avec le code ci-dessous :

```
if(!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse,
                lubridate)
```

Datasets

PID Malawi

Le premier jeu de données que nous utiliserons contient des données liées aux pulvérisation intradomiciliaire d'insecticide (PID) dans le cadre des efforts de lutte contre le paludisme entre 2014 et 2019 à Illovo, au Malawi. Notez que le jeu de données est au format long, chaque ligne représentant une période pendant laquelle les PID ont eu lieu dans un village. Étant donné que le même village est pulvérisé à plusieurs reprises à différents moments, les noms de village se répètent. Les jeux de données au format long sont souvent utilisés lorsque l'on traite des données de séries chronologiques avec des mesures répétées, car ils sont plus faciles à manipuler pour les analyses et les visualisations.

```
pid <- read_csv(here("data/Illovo_PID.csv"))
```

```
pid
```

```
## # A tibble: 5 × 9
##   village      cible_PID  reelle_PID couverture_p
##   <chr>        <dbl>     <dbl>       <dbl>
## 1 Mess           87        64         73.6
## 2 Nkombedzi     183       169        92.4
## 3 B Compound    16         16         100
## 4 D Compound     3          2          66.7
## 5 Post Office    6          3          50
##   date_debut_defaut date_fin_defaut date_debut_typique
##   <date>            <date>          <chr>
## 1 2014-04-07      2014-04-17      07/04/2014
## 2 2014-04-22      2014-04-27      22/04/2014
## 3 2014-05-13      2014-05-13      13/05/2014
## 4 2014-05-13      2014-05-13      13/05/2014
## 5 2014-05-13      2014-05-13      13/05/2014
## # ℹ 2 more variables: date_debut_longue <chr>,
## #   date_debut_mixte <chr>
```

Les variables incluses dans le jeu de données sont les suivantes :

- **village**: nom du village où la PID a eu lieu
- **cible_PID**: nombre de structures ciblées pour la PID
- **reelle_PID**: nombre de structures réellement PID
- **date_debut_defaut**: le jour où la PID a commencé, au format par défaut “aaaa-mm-jj”
- **date_fin_defaut**: jour où la PID s'est terminée, au format par défaut “aaaa-mm-jj”
- **date_debut_typique**: jour où la pulvérisation a commencé, au format “jj/mm/aaaa”
- **date_debut_longue**: jour où la PID a commencé, avec le mois écrit en entier, jour à deux chiffres, puis année à quatre chiffres
- **date_debut_mixte**: jour où la PID a commencé avec un mélange de différents formats

Séjours hospitaliers

Le deuxième jeu de données constitue des données factices de séjours hospitaliers simulés. Il contient les dates d'admission et de sortie de 150 patients. Tout comme le jeu de données PID, les dates d'admission sont formatées de différentes manières afin que vous puissiez exercer vos compétences en matière de formatage.

```
sej_hosp <- read_csv(here("data/sejours_hospitaliers.csv"))
```

```
sej_hosp
```

```
## # A tibble: 5 × 6
##   num_patient date_adm_defaut date_adm_courant
##       <dbl> <date>           <chr>
## 1 1 2021-05-23 05/23/2021
## 2 2 2022-12-07 12/07/2022
## 3 3 2022-03-27 03/27/2022
## 4 4 2022-04-28 04/28/2022
## 5 5 2023-06-28 06/28/2023
## #> #>   date_adm_abrege date_adm_mixte date_sortie_defaut
## #>   <chr>           <chr>           <date>
## #> 1 23 mai 2021    2021/05/23 2021-06-27
## #> 2 07 déc. 2022   12-07-2022 2023-02-19
## #> 3 27 mars 2022  2022/03/27 2022-05-31
## #> 4 28 avr. 2022  28.04.2022 2022-07-04
## #> 5 28 juin 2023  06-28-2023 2023-07-31
```

Introduction aux Dates en R

Dans R, il existe une classe spécifique conçue pour gérer les dates, appelée `Date`. Le format par défaut pour cette classe est "aaaa-mm-jj". Par exemple, le 31 décembre 2000 serait représenté comme `2000-12-31`.

Cependant, si vous entrez simplement une telle chaîne de caractères de date, R va initialement la considérer comme un caractère :

```
class("2000-12-31")
```

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

Si nous voulons créer une `Date`, nous pouvons utiliser la fonction `as.Date()` et écrire la date en suivant le format par défaut :

```
my_date <- as.Date("2000-12-31")
class(my_date)
```

```
## [1] "Date"
```

WATCH OUT



Notez le "D" majuscule dans la fonction `as.Date()` !

Maintenant que vos objets sont de la classe `Date`, vous pouvez maintenant faire des calculs simples comme trouver la différence entre deux dates :

```
as.Date("2000-12-31") - as.Date("2000-12-20")
```

```
## Time difference of 11 days
```

Ceci ne serait bien sûr pas possible si vous aviez de simples caractères :

```
"2000-12-31" - "2000-12-20"
```

```
Error in "2000-12-31" - "2000-12-20" :
non-numeric argument to binary operator
```

De nombreuses autres opérations s'appliquent uniquement à la classe Date. Nous les explorerons en détail plus tard.

Le format par défaut pour `as.Date()` est “aaaa-mm-jj”. D'autres formats courants comme “mm/jj/aaaa” ou “jj mois, aaaa” ne fonctionneront pas par défaut :

```
as.Date("12/31/2000")
as.Date("31 dec, 2000")
```

Cependant, R acceptera également “/” au lieu de “-” tant que l'ordre est toujours “aaaa/mm/jj”. Les dates s'afficheront au format par défaut “aaaa-mm-jj” :

```
as.Date("2000/12/31")
```

```
## [1] "2000-12-31"
```

En résumé, les seuls formats qui fonctionnent par défaut sont “aaaa-mm-jj” et “aaaa/mm/jj”. Plus tard dans cette leçon, nous allons apprendre à gérer différents formats de date et on vous donnerons des conseils sur la coercion de dates importées sous forme de chaînes de caractères dans la classe Date. Pour l'instant, l'essentiel est de comprendre que les dates ont leur propre classe avec ses propres propriétés de formatage.

SIDE NOTE



Il existe une autre classe de données utilisée pour les dates, appelée `POSIXct`. Cette classe gère les dates et heures ensemble, et le format par défaut est “aaaa-mm-jj hh:mm:ss”. Cependant, dans le cadre de ce cours, nous ne travaillerons pas avec cette classe car ce niveau d'analyse est beaucoup moins courant dans le domaine de la santé publique.

Conversion de chaînes de caractères en dates

Revenons à nos données PID et regardons comment R a classé nos variables de date !

```
pid %>%
  select(contains("date"))
```

```
## # A tibble: 112 × 5
##   date_debut_defaut date_fin_defaut date_debut_typique
##   <date>            <date>          <chr>
## 1 2014-04-07        2014-04-17      07/04/2014
```

```

## 2 2014-04-22      2014-04-27      22/04/2014
## 3 2014-05-13      2014-05-13      13/05/2014
## 4 2014-05-13      2014-05-13      13/05/2014
## 5 2014-05-13      2014-05-13      13/05/2014
## 6 2014-05-15      2014-05-26      15/05/2014
## 7 2014-05-27      2014-05-27      27/05/2014
## 8 2014-05-27      2014-05-27      27/05/2014
## 9 2014-05-28      2014-06-16      28/05/2014
## 10 2014-06-18     2014-06-27     18/06/2014
##   date_debut_longue date_debut_mixte
##   <chr>           <chr>
## 1 07 avril 2014    07-04-2014
## 2 22 avril 2014    22-04-2014
## 3 13 mai 2014     13-05-2014
## 4 13 mai 2014     13 mai 2014
## 5 13 mai 2014     13-05-2014
## 6 15 mai 2014     15 mai 2014
## 7 27 mai 2014     27 mai 2014
## 8 27 mai 2014     27 mai 2014
## 9 28 mai 2014     28-05-2014
## 10 18 juin 2014    2014/06/18
## # i 102 more rows

```

Comme nous pouvons le voir, les deux colonnes reconnues comme des dates sont `date_debut_defaut` et `date_fin_defaut`, qui suivent le format “aaaa-mm-jj” de R :

	date_debut_defaut	date_fin_defaut	date_debut_typique
1	👉<date>👉	👉<date>👉	<chr>
1	2014-04-07	2014-04-17	07/04/2014
2	2014-04-22	2014-04-27	22/04/2014

Toutes les autres colonnes de date dans notre jeu de données ont été importées comme des chaînes de caractères (“chr”), et si nous voulons les transformer en dates, nous devons indiquer à R qu’elles sont des dates, ainsi que spécifier l’ordre des composants de la date.

Vous vous demandez peut-être pourquoi il est nécessaire de spécifier l’ordre. Eh bien, imaginez qu’on ait une date écrite 01-02-03. Est-ce le 2 janvier 2003 ? Le 1er février 2003 ? Ou peut-être le 2 mars 2001 ? Il existe tellement de conventions différentes pour écrire les dates que si R devait deviner le format, il y aurait inévitablement des cas où il se tromperait.

Pour résoudre ce problème, il existe deux façons principales de convertir des chaînes en dates qui impliquent de spécifier l’ordre des composants. La première approche s’appuie sur les fonctions natives de R (appelé couramment par son nom anglais, “Base R”), et la seconde utilise un package appelé `lubridate` de la bibliothèque `tidyverse`. Regardons d’abord la fonction native de R !

Avec les fonctions natives de R (Base R)

Dans l'introduction nous avons vu comment convertir des chaînes de caractères en dates avec les fonctions natives de R, plus précisément avec la fonction `as.Date()`. Essayons de l'appliquer à notre colonne `date_debut_typique` sans spécifier l'ordre des composants pour voir ce qui se passe.

```
pid %>%
  mutate(date_debut_typique = as.Date(date_debut_typique)) %>%
  select(date_debut_typique)
```

```
## # A tibble: 5 × 1
##   date_debut_typique
##   <date>
## 1 2007-04-20
## 2 2022-04-20
## 3 2013-05-20
## 4 2013-05-20
## 5 2013-05-20
```

Évidemment, ce n'est pas du tout ce que nous voulions ! Si nous regardons la variable d'origine, nous pouvons voir qu'elle est formatée "jj/mm/aaaa". R a essayé d'appliquer son format par défaut à ces dates, ce qui donne ces résultats étranges.

WATCH OUT



Souvent, R vous retournera un message d'erreur si vous essayez de convertir des caractères ambiguës en dates sans spécifier l'ordre de leurs composants. Mais, comme nous venons de le voir, ce n'est pas toujours le cas ! Vérifiez toujours que votre code s'est exécuté comme prévu et ne vous fiez jamais seulement aux messages d'erreur pour vous assurer que vos transformations de données ont fonctionné correctement.

Pour que R interprète correctement nos dates, nous devons utiliser l'option `format` et spécifier les composants de notre date à l'aide d'une série de symboles. Le tableau ci-dessous montre les symboles pour les composants de format les plus courants :

Composant	Symbol	Exemple
Année, en format long (4 chiffres)	%Y	2023
Année, format abrégé (2 chiffres)	%y	23
Mois, en format numérique (1-12)	%m	01
Mois écrit en format long	%B	janvier
Mois écrit en format abrégé	%b	janv
Jour du mois	%d	31
Jour de la semaine, en format numérique (1-7 en		

Composant	Symbol	Exemple
commençant par dimanche)	%u	5
Jour de la semaine écrit en format long	%A	vendredi
Jour de la semaine écrit en format abrégé	%a	ven

**Add note about systems computer language

Si on revient à notre variable d'origine `date_debut_typique`, on voit qu'elle est formatée "jj/mm/aaaa", ce qui correspond au jour du mois, suivi du mois représenté par un nombre (01-12), puis de l'année en format long (4 chiffres). Si nous utilisons ces symboles, nous devrions obtenir les résultats que nous cherchons.

```
pid %>%
  mutate(date_debut_typique = as.Date(date_debut_typique, format="%d%m%Y"))
```

```
## # A tibble: 5 × 9
##   village      cible_PID reelle_PID couverture_p
##   <chr>        <dbl>     <dbl>       <dbl>
## 1 Mess          87        64         73.6
## 2 Nkombedzi    183       169        92.4
## 3 B Compound    16        16         100
## 4 D Compound     3         2         66.7
## 5 Post Office    6         3          50
##   date_debut_defaut date_fin_defaut date_debut_typique
##   <date>           <date>           <date>
## 1 2014-04-07     2014-04-17      NA
## 2 2014-04-22     2014-04-27      NA
## 3 2014-05-13     2014-05-13      NA
## 4 2014-05-13     2014-05-13      NA
## 5 2014-05-13     2014-05-13      NA
## # i 2 more variables: date_debut_longue <chr>,
## #   date_debut_mixte <chr>
```

Bon, ce n'est toujours pas ce que nous voulions. Avez-vous une idée de la raison pour laquelle ça n'a pas marché ? C'est parce que les composants de nos dates sont séparés par un slash "/", que nous devons inclure dans notre option de format. Essayons à nouveau !

```
pid %>%
  mutate(date_debut_typique = as.Date(date_debut_typique,
                                         format="%d/%m/%Y")) %>%
  select(date_debut_typique)
```

```
## # A tibble: 5 × 1
##   date_debut_typique
##   <date>
## 1 2014-04-07
## 2 2014-04-22
## 3 2014-05-13
```

```
## 4 2014-05-13  
## 5 2014-05-13
```

Cette fois ça a parfaitement fonctionné ! Maintenant nous savons comment convertir des chaînes de caractères en dates en utilisant la fonction native de R `as.Date()` avec l'option `format`.

Convertir date longue

Essayez de convertir la colonne `date_debut_longue` des données PID en classe `Date`. N'oubliez pas d'inclure tous les éléments dans l'option de format, y compris les symboles qui séparent les composants de la date !

Trouver les erreurs de code

Est-ce que vous arrivez à trouver toutes les erreurs dans le code suivant ?

```
as.Date("26 juin, 1987", format = "%d%b%y")
```

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

Avec lubridate

Le package `lubridate` nous donne une façon beaucoup plus simple de convertir des chaînes de caractères en dates que les fonctions natives de R. Avec ce package, il suffit de spécifier l'ordre dans lequel apparaissent l'année, le mois, et le jour en utilisant respectivement les lettres "y" pour l'année, "m" pour le mois et "d" pour le jour (correspondant à "year", "month" et "day" en anglais). Avec ces fonctions, ce n'est pas nécessaire de spécifier les caractères qui séparent les différents composants de la date.

Regardons quelques exemples :

```
mdy("04/30/2002")
```

```
## [1] "2002-04-30"
```

```
dmy("30 avril 2002")
```

```
## Warning: All formats failed to parse. No formats found.
```

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

```
ymd("2002-04-03")
```

```
## [1] "2002-04-03"
```

Facile ! Et comme nous pouvons le voir, nos dates sont affichées en utilisant le format R par défaut. Maintenant que nous arrivons à utiliser les fonctions du package lubridate, essayons de les appliquer à la variable `date_debut_longue` de notre jeu de données.

```
pid %>%
  mutate(date_debut_longue = mdy(date_debut_longue)) %>%
  select(date_debut_longue)
```

```
## Warning: There was 1 warning in `mutate()` .
## i In argument: `date_debut_longue =
##   mdy(date_debut_longue)` .
## Caused by warning:
## ! 77 failed to parse.
```

```
## # A tibble: 5 × 1
##   date_debut_longue
##   <date>
## 1 2014-07-20
## 2 NA
## 3 NA
## 4 NA
## 5 NA
```

Parfait, c'est exactement ce qu'on voulait !

Convertir date typique Essayez de convertir la colonne `date_debut_typique` du jeu de données PID en classe `Date` en utilisant les fonctions du package `lubridate`.

Formatage de base et lubridate

Le tableau suivant contient les formats trouvés dans les colonnes `date_adm_abrege` et `date_adm_mixte` de notre jeu de données de patients hospitalisés. Est ce que vous arrivez à remplir les cellules vides ?

Exemple	Base R	Lubridate
07 déc 2022		
03-27-2022		mdy
28.04.2022		%Y/%m/%d

Maintenant que nous connaissons deux façons de convertir des chaînes de caractères en classe `Date` en spécifiant l'ordre des composants ! Mais que faire si nous avons plusieurs formats de date dans la même colonne ? Passons à la section suivante pour le découvrir !

Gérer des dates mixtes avec `lubridate::parse_date_time()`

Lorsque l'on travaille avec des dates, il arrive parfois d'avoir différents formats au sein de la même colonne. Heureusement, lubridate dispose d'une fonction pratique à cet effet ! La fonction `parse_date_time()` est similaire aux fonctions que nous avons vues précédemment dans le package lubridate, mais avec plus de flexibilité et la possibilité d'inclure plusieurs formats de date dans le même appel en utilisant l'argument `orders`. Jetons un rapide coup d'œil à son fonctionnement avec quelques exemples simples.

Pour comprendre comment utiliser `parse_date_time()`, appliquons-le à une seule chaîne de caractères que nous voulons convertir en date.

```
parse_date_time("30/07/2001", orders="dmy")
```

```
## [1] "2001-07-30 UTC"
```

C'est parfait ! Utiliser la fonction de cette façon est équivalent à utiliser la fonction `dmy()`. Cependant, la vraie puissance de `parse_date_time()` se révèle lorsque nous avons plusieurs dates avec des formats différents.

SIDE NOTE



La partie "UTC" est le fuseau horaire par défaut utilisé pour analyser la date. Celui-ci peut être modifié avec l'argument `tz=`, mais changer le fuseau horaire par défaut est rarement nécessaire lorsqu'on traite uniquement de dates, contrairement à des dates-heures.

Regardons un autre exemple avec deux formats différents :

```
parse_date_time(c("1 janv 2000", "07/30/2001"), orders=c("dmy", "mdy"))
```

```
## Warning: 1 failed to parse.
```

```
## [1] NA                 "2001-07-30 UTC"
```

Notez que cet exemple spécifique fonctionnera toujours si vous changez l'ordre dans lequel vous présentez les formats :

```
parse_date_time(c("1 janv 2000", "07/30/2001"), orders=c("mdy", "dmy"))
```

```
## [1] NA           "2001-07-30 UTC"
```

Le dernier bloc de code fonctionne toujours car `parse_date_time()` vérifie chaque format spécifié dans l'argument `orders` jusqu'à trouver une correspondance. Cela signifie que, que vous listiez "dmy" en premier ou "mdy" en premier, il essaiera les deux formats sur chaque chaîne de date pour voir lequel convient. L'ordre n'a pas d'importance pour des chaînes de dates distinctes qui ne peuvent correspondre qu'à un seul format.

Cependant, lorsque l'on traite des dates ambiguës comme "01/02/2000" ou "01/03/2000", qui pourraient être interprétées soit comme le 2 janvier et le 3 janvier, soit comme le 1er février et le 1er mars respectivement, l'ordre dans `orders` a vraiment de l'importance :

```
parse_date_time(c("01/02/2000", "01/03/2000"), orders=c("mdy", "dmy"))
```

```
## [1] "2000-01-02 UTC" "2000-01-03 UTC"
```

Dans l'exemple ci-dessus, parce que "mdy" est listé en premier, la fonction interprète les dates comme étant le 2 janvier et le 3 janvier. Mais, si vous changez l'ordre et listiez "dmy" en premier, elle interpréterait les dates comme étant le 1er février et le 1er mars :

```
parse_date_time(c("01/02/2000", "01/03/2000"), orders=c("dmy", "mdy"))
```

```
## [1] "2000-02-01 UTC" "2000-03-01 UTC"
```

Par conséquent, lorsqu'il y a une ambiguïté potentielle dans les chaînes de dates, l'ordre dans lequel vous spécifiez les formats devient très important.

Utilisation de `parse_date_time`

Les dates dans le code ci-dessous sont le 9 novembre 2002, le 4 décembre 2001 et le 5 juin 2003. Complétez le code pour les convertir de caractères en dates.

```
parse_date_time(c("11/09/2002", "12/04/2001", "2003-06-05"), orders=c(...))
```

Revenons à notre jeu de données, cette fois sur la colonne `date_debut_mixte`.

```
pid %>%  
  select("date_debut_mixte")
```

```
## # A tibble: 5 × 1  
##   date_debut_mixte  
##   <chr>  
## 1 07-04-2014  
## 2 22-04-2014
```

```
## 3 13-05-2014
## 4 13 mai 2014
## 5 13-05-2014
```

Étant donné que cette colonne a été créée spécifiquement pour ce cours, nous connaissons les différents formats de date qu'elle contient. Dans votre propre travail, assurez-vous toujours de connaître le format de vos dates, car nous savons que certaines peuvent être ambiguës.

Ici, nous travaillons avec quatre formats différents, plus précisément :

- aaaa/mm/jj
- jj mois aaaa
- jj-mm-aaaa
- mm/jj/aaaa

Voyons à quoi cela ressemble dans lubridate par rapport au R de base :

Example date	Base R	Lubridate
2014/05/13	%Y/%m/%d	ymd
13 mai 2014	%B%d%Y	dmy
27-05-2014	%d-%m-%Y	dmy
07/21/14	%m/%d/%y	mdy

Ici, lubridate considère qu'il n'y a que trois formats différents ("ymd", "mdy" et "dmy"). Maintenant que nous savons comment nos données sont formatées, nous pouvons utiliser la fonction `parse_date_time()` pour les changer en classe `Date`.

```
pid %>%
  select(date_debut_mixte) %>%
  mutate(date_debut_mixte = parse_date_time(date_debut_mixte, orders =
    c("mdy", "ymd", "dmy")))
```

```
## Warning: There was 1 warning in `mutate()` .
## i In argument: `date_debut_mixte =
##   parse_date_time(date_debut_mixte, orders = c("mdy",
##     "ymd", "dmy"))` .
## Caused by warning:
## ! 25 failed to parse.
```

date_debut_mixte
2014-04-07
2014-04-22
2014-05-13

date_debut_mixte

2014-05-13

C'est beaucoup mieux ! R a correctement formaté notre colonne et elle est désormais reconnue comme une variable de type date. Vous vous demandez peut-être si l'ordre des formats est nécessaire dans ce cas. Essayons un ordre différent pour le découvrir !

```
pid %>%
  select(date_debut_mixte) %>%
  mutate(date_debut_mixte = parse_date_time(date_debut_mixte, orders =
    c("dmy", "mdy", "ymd")))
```

```
## Warning: There was 1 warning in `mutate()` .
## i In argument: `date_debut_mixte =
##   parse_date_time(date_debut_mixte, orders = c("dmy",
##     "mdy", "ymd"))` .
## Caused by warning:
## ! 25 failed to parse.
```

date_debut_mixte

2014-04-07

2014-04-22

2014-05-13

NA

2014-05-13

Cela ne semble pas avoir fait de différence, les dates sont toujours formatées correctement ! Si vous vous demandez pourquoi l'ordre importait dans notre exemple précédent mais pas ici, c'est lié au fonctionnement de la fonction `parse_date_time()`. Lorsqu'elle reçoit plusieurs ordres, la fonction tente de trouver la meilleure correspondance pour un sous-ensemble d'observations en considérant les séparateurs de dates et en favorisant l'ordre dans lequel les formats ont été fournis. Dans notre dernier exemple, les deux dates étaient séparées par un "/" et les deux formats fournis ("dmy" et "mdy") étaient des formats possibles, la fonction a donc favorisé le premier donné.

```
parse_date_time(c("01/02/2000", "01/03/2000"), orders=c("mdy", "dmy"))
```

```
## [1] "2000-01-02 UTC" "2000-01-03 UTC"
```

```
parse_date_time(c("01/02/2000", "01/03/2000"), orders=c("dmy", "mdy"))
```

```
## [1] "2000-02-01 UTC" "2000-03-01 UTC"
```

Dans nos données PID, nous avions aussi des formats qui pouvaient être ambigus comme jj-mm-aaaa et mm/jj/aaaa. Mais ici, la fonction peut utiliser les séparateurs comme indice pour trouver des règles de formatage et distinguer les différents formats. Par exemple, si nous avons une date ambiguë comme 01-02-2000, mais aussi une date avec le même séparateur qui n'est pas ambiguë comme 30-05-2000, la fonction déterminera que la réponse la plus probable est que toutes les dates séparées par un “-” sont au format jj-mm-aaaa, et appliquera cette règle de manière récursive aux données d'entrée. Si vous voulez en savoir plus sur les détails de la fonction `parse_date_time()`, cliquez ici ou exécutez `?parse_date_time` dans R !

Utilisation de `parse_date_time` avec `adm_date_messy` A l'aide du tableau que vous avez rempli pour l'exercice de la [Section 6.2 Lubridate](#), utilisez la fonction `parse_date_time()` pour changer la classe de la colonne `date_adm_mixte` du jeu de données de patients hospitalisés en Date, ip!

Modifier l’Affichage des Dates

Jusqu'à présent, nous avons converti des chaînes de caractères de divers formats en classe `Date` qui suit un format par défaut “aaaa-mm-jj”. Mais que faire si nous voulons que nos dates s'affichent dans un format spécifique qui est différent de ce format par défaut, comme lorsque nous créons des rapports ou des graphiques ? Cela est rendu possible en reconvertissant les dates en chaînes de caractères en utilisant la fonction `format()` !

La fonction `format()` vous offre une grande flexibilité pour personnaliser l'apparence de vos dates selon vos préférences. Vous pouvez accomplir cela en utilisant les mêmes symboles que nous avons vu avec la fonction `as.Date()`, en les ordonnant pour correspondre à l'apparence souhaitée de votre date. Revenons au tableau pour rafraîchir notre mémoire sur la façon dont les différentes parties d'une date sont représentées dans R.

Composant	Symbol	Exemple
Année, en format long (4 chiffres)	%Y	2023
Année, format abrégé (2 chiffres)	%y	23
Mois, en format numérique (1-12)	%m	01
Mois écrit en format long	%B	janvier
Mois écrit en format abrégé	%b	janv
Jour du mois	%d	31
Jour de la semaine, en format numérique (1-7 en commençant par dimanche)	%u	6
Jour de la semaine écrit en format long	%A	vendredi
Jour de la semaine écrit en format abrégé	%a	ven

Très bien, essayons maintenant d'appliquer cette fonction à une seule date. Disons que nous voulons que la date 2000-01-31 s'affiche comme "31 janv. 2000".

```
my_date <- as.Date("2000-01-31")
format(my_date, "%d %b. %Y")
```

```
## [1] "31 Jan. 2000"
```

Créer un vecteur de dates Créez un vecteur de dates contenant la date du 7 mai 2018. Formatez ensuite la date en jj/mm/aaaa en tant que caractère.

Maintenant, essayons de l'utiliser sur nos données PID. Créons une nouvelle variable appelée `date_debut_char` à partir de la colonne `date_debut_defaut`. Nous allons la formater pour qu'elle s'affiche comme jj-mm-aaaa.

```
pid %>%
  mutate(date_debut_char = format(date_debut_defaut, "%d-%m-%Y")) %>%
  select(date_debut_defaut)
```

```
## # A tibble: 5 × 1
##   date_debut_defaut
##   <date>
## 1 2014-04-07
## 2 2014-04-22
## 3 2014-05-13
## 4 2014-05-13
## 5 2014-05-13
```

Super ! Faisons un dernier exemple en utilisant notre variable `date_fin_defaut` et en la formatant comme jj mois aaaa.

```
pid %>%
  mutate(end_date_char = format(date_fin_defaut, "%d %B %Y")) %>%
  select(date_fin_defaut)
```

```
## # A tibble: 5 × 1
##   date_fin_defaut
##   <date>
## 1 2014-04-17
## 2 2014-04-27
## 3 2014-05-13
## 4 2014-05-13
## 5 2014-05-13
```

Génial !

EN RÉSUMÉ

Félicitations pour avoir terminé la première leçon sur les dates ! Maintenant que vous comprenez comment les dates sont stockées, affichées et formatées dans R, vous pouvez passer à la section suivante où vous apprendrez à effectuer des manipulations avec les dates et à créer des graphiques de séries temporelles de base.

Answer Key

Convertir une date longue

```
irs <- irs %>%
  mutate(start_date_long = as.Date(start_date_long, format="%B, %d %Y"))
```

Trouver les erreurs de code

```
as.Date("June 26, 1987", format = "%B %d, %Y")
```

Convertir une date typique

```
irs %>%
  mutate(start_date_typical = dmy(start_date_typical))
```

Formatage de base et lubridate

Date example	Base R	Lubridate
07 dec, 2022	%b %d, %Y	dmy
03-27-2022	%m-%d-%Y	mdy
28.04.2022	%d.%m.%Y	dmy
2021/05/23	%Y/%m/%d	ymd

Utilisation de parse_date_time

```
parse_date_time(c("11/09/2002", "12/04/2001", "2003-06-05"), orders=c("mdy",
  "ymd"))
```

Utilisation de parse_date_time avec adm_date_messy

```
ip %>%
  mutate(adm_date_messy = parse_date_time(adm_date_messy, orders = c("mdy",
    "dmy", "ymd")))
```

Créer un vecteur de dates

```
my_date <- as.Date("2018-05-07")
format(my_date, "%m/%d/%Y")
```

Contributors

The following team members contributed to this lesson:

(make sure to update the contributor list accordingly!)

Dates 2 : Manipuler les dates dans R

GRAPH Network & WHO, supported by the Global Fund to fight HIV, TB & Malaria

October 2023

This document is a draft of a lesson made by the GRAPH Network, a non-profit headquartered at the University of Geneva Global Health Institute, in collaboration with the World Health Organization, under a Global Fund 2023 grant to create e-learning modules to build in-country data capacity for epidemiological and impact analysis for National HIV, TB and malaria programs

Introduction
Objectifs d'apprentissage
Packages
Données
Calcul des intervalles de date
Utilisation de l'opérateur “-”
Utilisation de l'opérateur d'intervalle du package lubridate
Comparaison
Extraction des composants de la date
Arrondir
WRAP UP!
Answer Key

Introduction

Vous avez maintenant une bonne compréhension de la façon dont les dates sont stockées, affichées et formatées dans R. Dans cette leçon, vous apprendrez à effectuer des analyses simples avec les dates, telles que le calcul de l'intervalle de temps entre les intervalles de dates et la création de graphiques de séries chronologiques ! Ces compétences sont cruciales pour toute personne travaillant avec des données de santé publique, car elles sont la base de la compréhension des tendances temporelles telles que la propagation des maladies, les changements dans les indicateurs de santé au niveau de la population et l'impact des mesures préventives !

Objectifs d'apprentissage

- Vous savez comment calculer les intervalles entre les dates
- Vous savez comment extraire des composants des colonnes de date
- Vous savez comment arrondir les dates
- Vous êtes capable de créer des graphiques de séries chronologiques simples

Packages

Veuillez charger les packages nécessaires pour cette leçon avec le code ci-dessous :

```
if(!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse,
  here,
  lubridate,
  ggplot2)
```

Données

Les premières données avec lesquelles nous travaillerons sont les données IRS de la leçon précédente. Consultez la première leçon sur les dates pour plus d'informations sur le contenu de ces données de pulvérisation intradomiciliaire d'insecticide (IRS).

```
données_pulvérisation <- read_csv(here("data/Iollovo_data.csv"))
```

```
données_pulvérisation
```

```
## # A tibble: 5 × 9
##   village      target_spray sprayed coverage_p
##   <chr>        <dbl>    <dbl>     <dbl>
## 1 Mess          87       64      73.6
## 2 Nkombedzi    183      169      92.4
## 3 B Compound    16       16      100
## 4 D Compound     3        2      66.7
## 5 Post Office    6        3      50
##   start_date_default end_date_default start_date_typical
##   <date>            <date>           <chr>
## 1 2014-04-07      2014-04-17      07/04/2014
## 2 2014-04-22      2014-04-27      22/04/2014
## 3 2014-05-13      2014-05-13      13/05/2014
## 4 2014-05-13      2014-05-13      13/05/2014
## 5 2014-05-13      2014-05-13      13/05/2014
## # i 2 more variables: start_date_long <chr>,
## #   start_date_messy <chr>
```

Le deuxième ensemble de données que nous utiliserons contient des données de la même étude à partir de laquelle les données IRS ont été extraites. Ici, nous avons le taux d'incidence mensuel moyen du paludisme pour 1000 personnes de 2015 à 2019 pour les villages ayant reçu l'IRS (cas) et les villages n'ayant pas reçu l'IRS (témoins). Le jeu de données contient également la température minimale moyenne et la température maximale moyenne à Illovo pour chaque mois de 2015 à 2019.

```
données_météo_paludisme <- read_csv(here("data/Iollovo_ir_weather.csv"))
```

```
données_météo_paludisme
```

```

## # A tibble: 5 × 5
##   date      ir_case ir_control avg_min avg_max
##   <date>     <dbl>     <dbl>    <dbl>    <dbl>
## 1 2015-01-10     42.9     19.6    21.2    31.6
## 2 2015-02-03     61.0     10.1    21.5    32.9
## 3 2015-03-11     74.1     56.8    20.6    33.4
## 4 2015-04-15     95.2     34.7    18.5    32.3
## 5 2015-05-05     89.8     31.9    15.9    31.4

```

Les variables incluses dans ces données sont les suivantes :

- `date` : Points temporels mensuels allant de 2015 à 2019, avec le jour du mois générée de manière aléatoire
- `cas_IRS` : Taux d'incidence moyen du paludisme pour 1000 personnes pour les villages ayant reçu l'IRS
- `témoins` : Taux d'incidence moyen du paludisme pour 1000 personnes pour les villages n'ayant pas reçu l'IRS
- `moyenne_min` : Température minimale moyenne mensuelle en degrés Celsius
- `moyenne_max` : Température maximale moyenne mensuelle en degrés Celsius

MÉTÉO

```
météo <- read_csv(here("data/Illovo_weather.csv"))
```

```
météo
```

```

## # A tibble: 5 × 4
##   date      min_temp max_temp rain
##   <date>     <dbl>    <dbl>  <dbl>
## 1 2015-01-01     21.5    29.9  21.7
## 2 2015-01-02     19.6    30.4   2.2
## 3 2015-01-03     21.6    29.9  25.8
## 4 2015-01-04     20.0    29.5    1
## 5 2015-01-05     20.0    32.2   53

```

Calcul des intervalles de date

Pour commencer, nous allons examiner deux façons de calculer les intervalles, la première en utilisant l'opérateur “-” en R de base, et la seconde en utilisant l'opérateur d'intervalle du package `lubridate`. Jetons un coup d’œil à ces deux méthodes et comparons-les.

Utilisation de l'opérateur “-”

La première façon de calculer les différences de temps consiste à utiliser l'opérateur “-” pour soustraire une date d'une autre. Créons deux variables de date et essayons !

```
date_1 <- as.Date("2000-01-01") # 1er janvier 2000  
date_2 <- as.Date("2000-01-31") # 31 janvier 2000  
date_2 - date_1
```

```
## Time difference of 30 days
```

C'est aussi simple que ça ! Ici, nous pouvons voir que R renvoie la différence de temps en jours.

Utilisation de l'opérateur d'intervalle du package lubridate

La deuxième façon de calculer des intervalles de temps consiste à utiliser l'opérateur %--% du package `lubridate`. Nous pouvons voir ici que la sortie est légèrement différente de la sortie de R de base.

```
date_1 %--% date_2
```

```
## [1] 2000-01-01 UTC--2000-01-31 UTC
```

Notre sortie est un intervalle entre deux dates. Si nous voulons savoir combien de jours se sont écoulés, nous devons utiliser la fonction `days()`. Le (1) ici indique à `lubridate` de compter par incrément de 1 jour à la fois.

```
date_1 %--% date_2/days(1)
```

```
## [1] 30
```

Techniquement, spécifier `days(1)` n'est pas vraiment nécessaire, nous pouvons également laisser les parenthèses vides (c'est-à-dire `days()`) et obtenir le même résultat, car la valeur par défaut de `lubridate` est de compter par incrément de 1. Cependant, si nous voulons compter par incrément de 5 jours par exemple, nous pouvons spécifier `days(5)` et le résultat retourné sera de 6, car $5 \times 6 = 30$.

```
date_1 %--% date_2/days(5)
```

```
## [1] 6
```

Utilisez les trois méthodes différentes pour calculer le temps entre les dates ci-dessous. Utilisez la fonction `weeks()` à la place de `days()` dans la méthode lubridate pour obtenir la réponse en semaines.

```
oct_31 <- as.Date("2023-10-31")
jul_20 <- as.Date("2023-07-20")
```

Comparaison

Alors, quelle est la meilleure méthode à utiliser ? Eh bien, vous pouvez probablement voir que lubridate offre plus de flexibilité, et grâce à la manière dont il gère les dates (les détails étant hors du champ de cette formation), il est plus précis !

Jetons un coup d'œil à un exemple de cela en calculant un intervalle de 6 ans. Avec R de base, les résultats sont donnés en jours. Si nous voulons obtenir le résultat en années, nous devons utiliser la fonction `as.numeric()` pour transformer les résultats en un nombre sans l'unité "jours" et diviser par 365,25. Nous utilisons 365,25 parce qu'il y a un jour supplémentaire tous les 4 ans (c'est-à-dire les années bissextiles), donc en moyenne, il y a 365,25 jours par an. Avec lubridate, nous pouvons simplement spécifier `years()`. Voyons comment chaque méthode se comporte dans l'exemple suivant.

```
date_1 <- as.Date("2000-01-01") # 1er janvier 2000
date_2 <- as.Date("2006-01-01") # 1er janvier 2006
as.numeric(date_2-date_1)/365.25
```

```
## [1] 6.001369
```

```
date_1 %--% date_2/years()
```

```
## [1] 6
```

D'accord, ils sont très similaires, mais pas exactement les mêmes ! Évidemment, la réponse correcte est que 6 ans se sont écoulés entre les deux dates données. Cependant, les dates peuvent devenir compliquées, car le nombre de jours dans une année, voire dans un mois, n'est pas toujours le même. Essayer de calculer des intervalles de dates avec R de base est une approximation plutôt qu'une réponse exacte. Dans l'exemple ci-dessus, nous n'avons pas pu obtenir exactement 6 ans en utilisant l'opérateur moins en R de base, même si nous divisons par 365, 365,25 ou 366.

```
as.numeric(date_2-date_1)/365
```

```
## [1] 6.005479
```

```
as.numeric(date_2-date_1)/365.25
```

```
## [1] 6.001369  
  
as.numeric(date_2-date_1) / 366
```

```
## [1] 5.989071
```

Lubridate, en revanche, peut gérer ces complexités, il donnera donc une réponse exacte. Les différences sont légères, mais en termes de flexibilité et de précision, lubridate est le grand gagnant !

Bien que nous ne couvrions pas les date-heures dans ce cours, il est bon de savoir que lubridate est particulièrement utile pour traiter les fuseaux horaires et les changements d'heure d'été.

Pouvez-vous appliquer cela à notre jeu de données de pulvérisation ? Créez une nouvelle variable appelée `durée_pulvérisation` et utilisez l'opérateur `%--%` de lubridate pour calculer le nombre de jours entre `start_date_default` et `end_date_default`.

Extraction des composants de la date

Parfois, lors de votre nettoyage ou de votre analyse de données, vous devrez peut-être extraire un composant spécifique de votre variable de date. Un ensemble de fonctions utiles dans le package `lubridate` vous permet de le faire exactement. Par exemple, si nous voulions créer une colonne avec seulement le mois où la pulvérisation a commencé à chaque intervalle, nous pourrions utiliser la fonction `month()` de la manière suivante :

```
données_pulvérisation %>%  
  mutate(mois_début = month(start_date_default)) %>%  
  select(village, start_date_default, mois_début)
```

```
## # A tibble: 5 × 3  
##   village      start_date_default  mois_début  
##   <chr>        <date>                <dbl>  
## 1 Mess        2014-04-07             4  
## 2 Nkombedzi  2014-04-22             4  
## 3 B Compound 2014-05-13             5  
## 4 D Compound  2014-05-13            5  
## 5 Post Office 2014-05-13            5
```

Comme nous pouvons le voir ici, cette fonction renvoie le mois sous forme de numéro de 1 à 12. Pour notre première observation, la pulvérisation a commencé au cours du quatrième mois, donc en avril. C'est aussi simple que ça ! Si nous voulons que R affiche le

mois écrit plutôt que le numéro en dessous, nous pouvons utiliser l'argument `label=TRUE`.

```
données_pulvérisation %>%
  mutate(mois_début = month(start_date_default, label=TRUE)) %>%
  select(village, start_date_default, mois_début)
```

```
## # A tibble: 5 × 3
##   village      start_date_default mois_début
##   <chr>        <date>            <ord>
## 1 Mess         2014-04-07       Apr
## 2 Nkombedzi   2014-04-22       Apr
## 3 B Compound  2014-05-13       May
## 4 D Compound  2014-05-13       May
## 5 Post Office 2014-05-13       May
```

De même, si nous voulions extraire l'année, nous utiliserions la fonction `year()`.

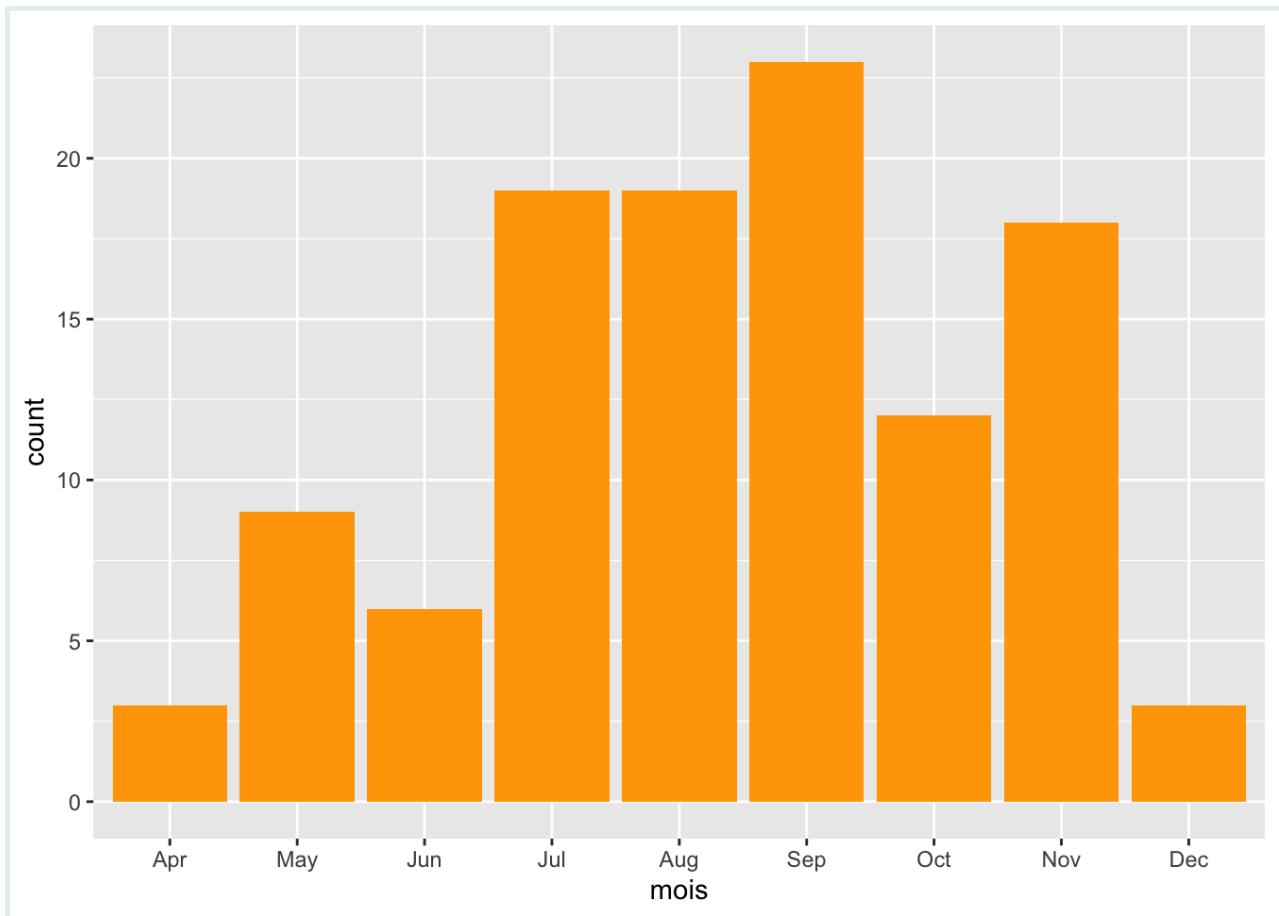
```
données_pulvérisation %>%
  mutate(année_début = year(start_date_default)) %>%
  select(village, start_date_default, année_début)
```

```
## # A tibble: 5 × 3
##   village      start_date_default année_début
##   <chr>        <date>            <dbl>
## 1 Mess         2014-04-07       2014
## 2 Nkombedzi   2014-04-22       2014
## 3 B Compound  2014-05-13       2014
## 4 D Compound  2014-05-13       2014
## 5 Post Office 2014-05-13       2014
```

Créez une nouvelle variable appelée `jour_semaine_début` et extrayez le jour de la semaine où la pulvérisation a commencé de la même manière qu'indiqué ci-dessus, mais avec la fonction `wday()`. Essayez d'afficher les jours de la semaine écrits plutôt que numériquement.

Une raison pour laquelle vous pourriez vouloir extraire des composants de date spécifiques est lorsque vous souhaitez visualiser vos données, ce qui est très simple à faire dans R en utilisant le package `ggplot2` ! Par exemple, disons que nous voulions comparer les mois où la pulvérisation commence, nous pouvons le faire en créant une nouvelle variable de mois avec la fonction `month()` et en traçant un graphique à barres avec `geom_bar`.

```
données_pulvérisation %>%
  mutate(mois = month(start_date_default, label = TRUE)) %>%
  ggplot(aes(x = mois)) +
  geom_bar(fill = "orange")
```



Ici, nous pouvons voir que la plupart des campagnes de pulvérisation ont commencé entre juillet et novembre, sans aucune en janvier, février et mars. Les auteurs de l'article à partir duquel ces données ont été extraites ont déclaré que les campagnes de pulvérisation visaient à se terminer juste au début de la saison des pluies (novembre-avril) au Malawi. Cela était à la fois pour des raisons pratiques et en prévision d'une transmission plus élevée du paludisme. Nous pouvons voir ce schéma temporel de pulvérisation reflété dans notre graphique !

Créez un nouveau graphique représentant les mois où la campagne de pulvérisation s'est terminée et comparez-le au graphique de son commencement. Cela correspond-il à vos attentes compte tenu de la variable `durée_pulvérisation` que vous avez créée dans l'exercice de la section précédente ?

Arrondir

Parfois, il est nécessaire d'arrondir nos dates vers le haut ou vers le bas si nous voulons analyser ou visualiser nos données de manière significative. Tout d'abord, voyons ce que nous entendons par "arrondir" avec quelques exemples simples.

Prenons la date du 17 mars 2012. Si nous voulions arrondir vers le bas au mois le plus proche, alors nous utiliserions la fonction `floor_date()` de `lubridate` avec l'argument `unit="month"`.

```
ma_date_inf <- as.Date("2012-03-17")
floor_date(ma_date_inf, unit="month")
```

```
## [1] "2012-03-01"
```

Comme nous pouvons le voir, notre date est maintenant le 1er mars 2012.

Si nous voulions arrondir vers le haut, nous pouvons utiliser la fonction `ceiling_date()`. Essayons ceci avec la date du 3 janvier 2020.

```
ma_date_sup <- as.Date("2020-01-03")
ceiling_date(ma_date_sup, unit="month")
```

```
## [1] "2020-02-01"
```

Avec `ceiling_date()`, le 3 janvier a été arrondi au 1er février.

Enfin, nous pouvons également simplement arrondir sans spécifier vers le haut ou vers le bas, et les dates sont automatiquement arrondies à l'unité spécifiée la plus proche.

```
mes_dates <- as.Date(c("2000-11-03", "2000-11-27"))
round_date(mes_dates, unit="month")
```

```
## [1] "2000-11-01" "2000-12-01"
```

Ici, nous pouvons voir qu'en arrondissant au mois le plus proche, le 3 novembre est arrondi au 1er novembre, et le 27 novembre est arrondi au 1er décembre.

Nous pouvons également arrondir vers le haut ou vers le bas à l'année la plus proche. Que pensez-vous que sera la sortie si nous arrondissons vers le bas la date du 29 novembre 2001 à l'année la plus proche :

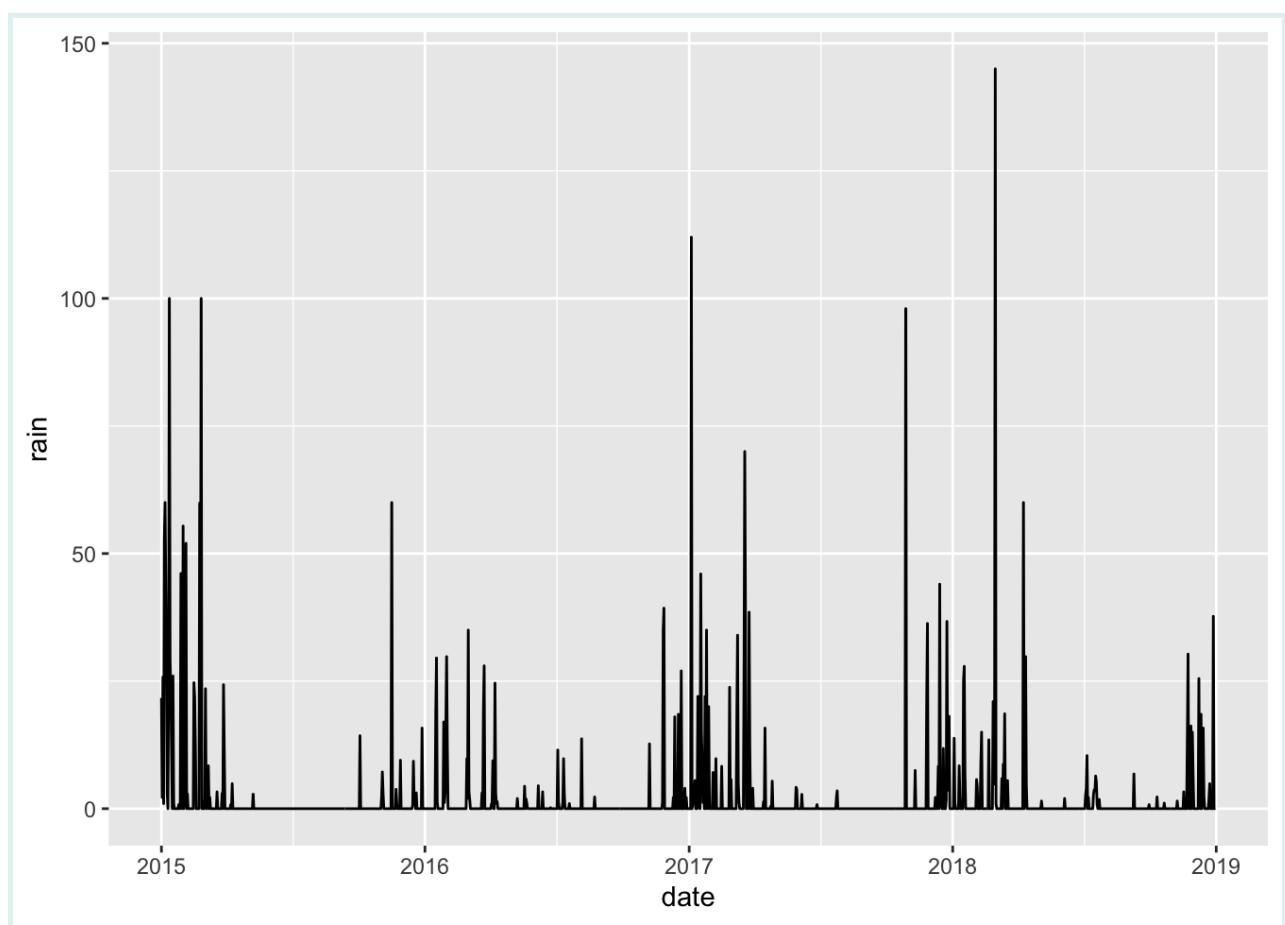
```
date_arrondie <- as.Date("2001-11-29")
floor_date(date_arrondie, unit="year")
```

J'espère que ce que nous entendons par "arrondir" est un peu plus clair ! Alors, pourquoi cela pourrait-il être utile avec nos données ? Eh bien, passons maintenant à nos données météorologiques.

```
météo
```

Comme nous pouvons le voir, nos données météorologiques sont enregistrées quotidiennement, mais ce niveau de détail n'est pas idéal pour étudier comment les schémas météorologiques affectent la transmission du paludisme, qui suit un schéma saisonnier. Les données météorologiques quotidiennes peuvent être assez bruyantes compte tenu de la variation significative d'un jour à l'autre. Par exemple, un graphique des précipitations quotidiennes ne serait pas très informatif. Essayons pour voir :

```
météo %>%
  ggplot() +
  geom_line(aes(date, rain))
```



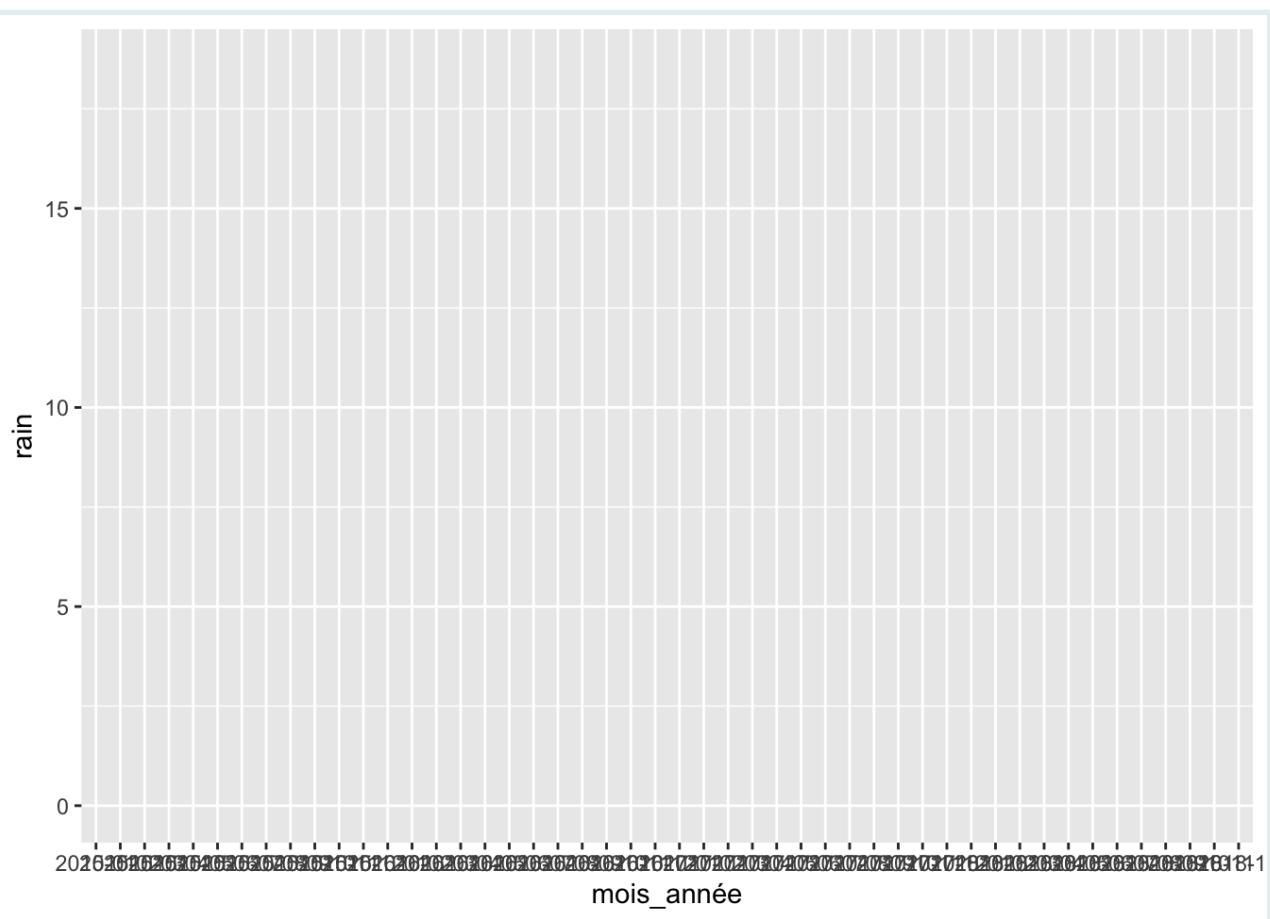
Outre le fait qu'il soit visuellement confus, ce graphique ne permet pas d'illustrer efficacement les tendances saisonnières. L'agrégation mensuelle est une approche plus efficace pour capturer les variations saisonnières et réduire le bruit. Si nous voulions représenter les précipitations mensuelles moyennes, notre première tentative pourrait être d'utiliser la fonction `str_sub()` pour extraire les sept premiers caractères de notre date (le mois et l'année).

```
météo_mauvaise <- météo %>%
  mutate(mois_année=str_sub(date, 1, 7)) %>%
  group_by(mois_année) %>%
  summarise(rain=mean(rain))
météo_mauvaise
```

Cependant, si nous essayons de le représenter, notre nouvelle variable `mois_année` n'est plus une variable de date, nous ne pouvons donc pas représenter des graphiques dans le temps car elle n'est pas continue !

```
météo_mauvaise %>%
  ggplot() +
  geom_line(aes(mois_année, rain))
```

```
## `geom_line()` : Each group consists of only one
## observation.
## i Do you need to adjust the group aesthetic?
```

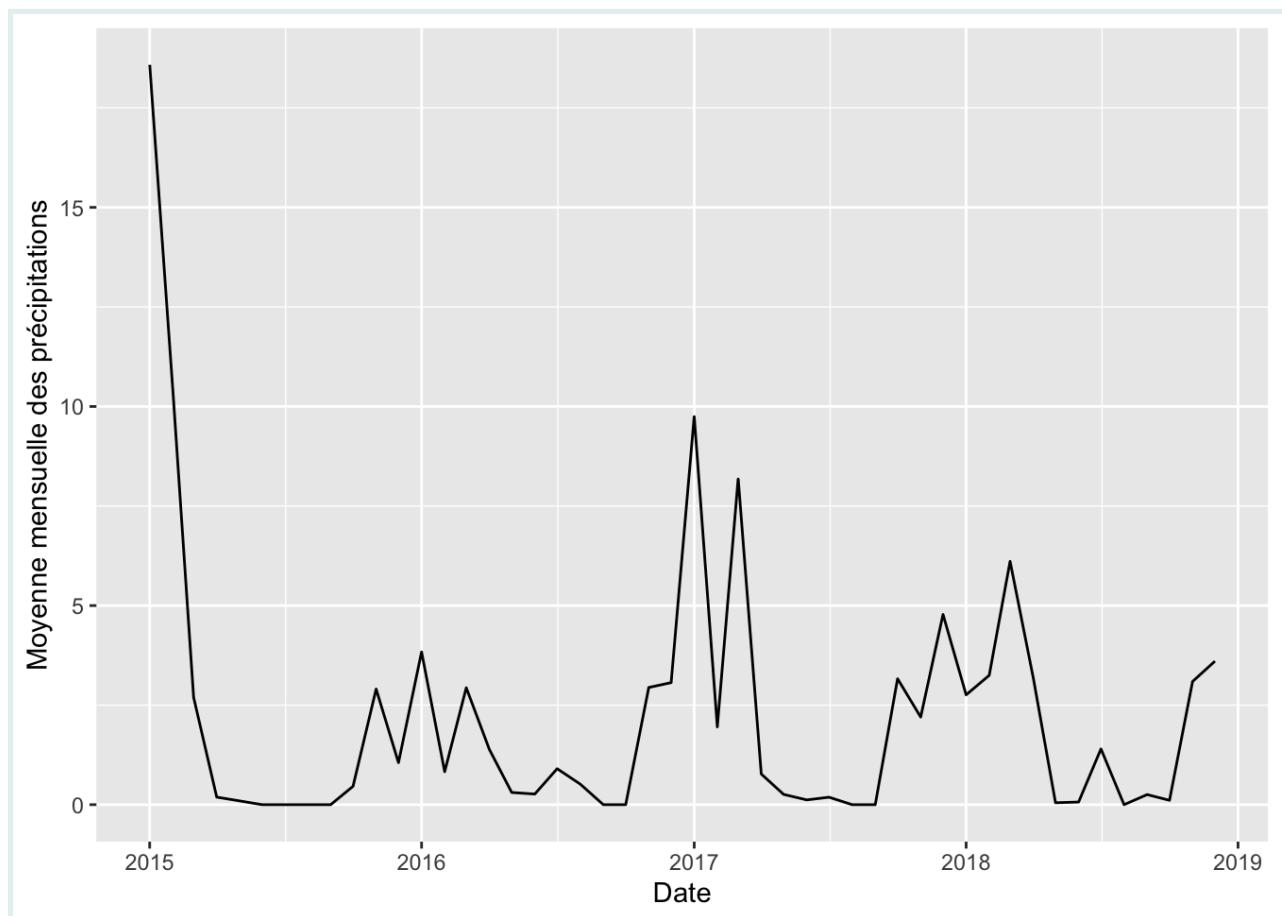


La meilleure façon de faire cela est d'abord d'arrondir nos dates au mois à l'aide de la fonction `floor_date()`, puis de regrouper nos données par notre nouvelle variable `mois_année`, et enfin de calculer la moyenne mensuelle. Essayons maintenant.

```
météo <- météo %>%
  mutate(mois_année=floor_date(date, unit="month")) %>%
  group_by(mois_année) %>%
  summarise(rain=mean(rain))
météo
```

Maintenant nous pouvons représenter nos données et nous aurons un graphique de la pluviométrie mensuelle moyenne sur la période de pulvérisation de 4 ans.

```
météo %>%  
  ggplot() +  
  geom_line(aes(mois_année, rain)) +  
  labs(x="Date", y="Moyenne mensuelle des précipitations")
```



Cela semble beaucoup mieux ! Maintenant, nous obtenons une image beaucoup plus claire des tendances saisonnières et des variations annuelles.

À l'aide des données météorologiques, créez un nouveau graphique représentant les températures minimales et maximales moyennes mensuelles de 2015 à 2019.

WRAP UP!

[XXX NICE WRAP UP MESSAGE OR SUMMARY IF NEEDED HERE XXX]

Answer Key

```
données_pulvérisation %>%
  mutate(jour_semaine_début = wday(start_date_default, label=TRUE)) %>%
  select(village, start_date_default, jour_semaine_début)
```

```
## # A tibble: 5 × 3
##   village      start_date_default jour_semaine_début
##   <chr>        <date>                <ord>
## 1 Mess         2014-04-07            Mon
## 2 Nkombedzi    2014-04-22            Tue
## 3 B Compound   2014-05-13            Tue
## 4 D Compound   2014-05-13            Tue
## 5 Post Office  2014-05-13            Tue
```

Contributors

The following team members contributed to this lesson:

(make sure to update the contributor list accordingly!)

Les factors dans R

GRAPH Network & WHO, supported by the Global Fund to fight HIV, TB & Malaria

October 2023

This document is a draft of a lesson made by the GRAPH Network, a non-profit headquartered at the University of Geneva Global Health Institute, in collaboration with the World Health Organization, under a Global Fund 2023 grant to create e-learning modules to build in-country data capacity for epidemiological and impact analysis for National HIV, TB and malaria programs

Introduction
Objectifs d'apprentissage
Packages
Jeu de données : Mortalité VIH
Qu'est-ce que les facteurs ?
Les facteurs en action
Manipuler les facteurs avec <code>forcats</code>
<code>fct_relevel</code>
<code>fct_reorder</code>
<code>fct_recode</code>
<code>fct_lump</code>
Conclusion
Corrigé
Annexe : Codebook

Introduction

Les facteurs sont une classe de données importante dans R pour représenter et travailler avec des variables catégorielles. Dans cette leçon, nous allons apprendre à créer des facteurs et à les manipuler avec des fonctions du package `forcats`, qui fait partie du `tidyverse`. Plongeons-nous dedans !

Objectifs d'apprentissage

- Vous comprenez ce que sont les facteurs et en quoi ils diffèrent des caractères dans R.
- Vous êtes capable de modifier **l'ordre** des niveaux des facteurs.
- Vous êtes capable de modifier **la valeur** des niveaux des facteurs.

Packages

```
# Load packages
if(!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse,
               here)
```

Jeu de données : Mortalité VIH

Nous allons utiliser un jeu de données contenant des informations sur la mortalité VIH en Colombie de 2010 à 2016, hébergé sur la plateforme de données ouvertes 'Datos Abiertos Colombia'. Vous pouvez en savoir plus et accéder à l'ensemble du jeu de données [ici](#).

Chaque ligne correspond à un individu décédé du SIDA ou de complications liées au SIDA.

```
hiv_mort <- read_csv(here("data/colombia_hiv_deaths_2010_to_2016"))
```

```
## # A tibble: 5 × 25
##   municipality_type death_location birth_date birth_year
##   <chr>              <chr>          <date>        <dbl>
## 1 Municipal head    Hospital/clinic 1956-05-26  1956
## 2 Municipal head    Hospital/clinic 1983-10-10  1983
## 3 Municipal head    Hospital/clinic 1967-11-22  1967
## 4 Municipal head    Home/address    1964-03-14  1964
## 5 Municipal head    Hospital/clinic 1960-06-27  1960
## # ... with 16 more variables: birth_month <dbl>, birth_day <dbl>,
## #   death_year <dbl>, death_month <dbl>, death_day <dbl>,
## #   <chr> <dbl> <chr> <dbl> <chr> <dbl>
## #   1 May      26   2012 Sep   14
## #   2 Oct      10   2012 Mar   17
## #   3 Nov      22   2011 Oct   19
## #   4 Mar      14   2012 Nov   19
## #   5 Jun      27   2012 Jan   13
## # ... with 16 more variables: age_at_death <dbl>, gender <chr>,
## #   education_level <chr>, occupation <chr>, ...
```

Voir l'annexe au bas pour le dictionnaire de données décrivant toutes les variables.

Qu'est-ce que les facteurs ?

Les facteurs sont une classe de données importante dans R utilisée pour représenter des variables catégorielles.

Une variable catégorielle prend un ensemble limité de valeurs ou niveaux possibles. Par exemple, pays, race ou affiliation politique. Celles-ci diffèrent des variables texte libre qui prennent des valeurs arbitraires, comme des noms de personnes, titres de livres ou commentaires de médecins.



Rappel des principales classes de données dans R

RECAP



- Numérique : Représente des données numériques continues, incluant des nombres décimaux.
- Entier : Spécifiquement pour les nombres entiers sans décimales.
- Caractère : Utilisé pour les données textuelles ou chaînes de caractères.
- Logique : Représente des valeurs booléennes (VRAI ou FAUX).
- Facteur : Utilisé pour les données catégorielles avec des niveaux ou catégories prédéfinis.
- Date : Représente des dates sans heures.

Les facteurs ont quelques avantages clés par rapport aux vecteurs de caractères pour travailler avec des données catégorielles dans R :

- Les facteurs sont stockés dans R de manière légèrement plus efficace que les caractères.
- Certaines fonctions statistiques, comme `lm()`, nécessitent que les variables catégorielles soient passées en paramètre sous forme de facteurs.
- Les facteurs permettent de contrôler l'ordre des catégories ou niveaux. Cela permet de trier et tracer correctement les données catégorielles.

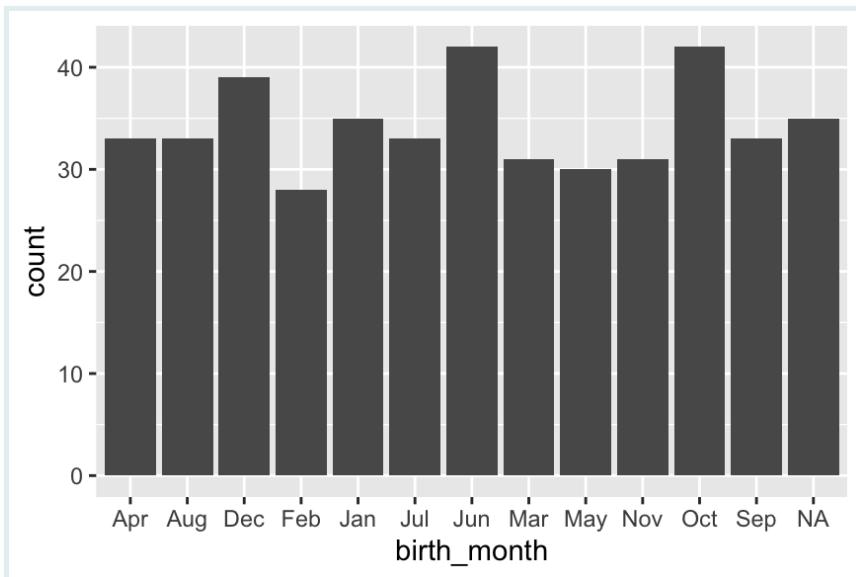
Ce dernier point, le contrôle de l'ordre des niveaux de facteurs, sera notre objectif principal.

Les facteurs en action

Voyons un exemple concret de l'intérêt des facteurs en utilisant le jeu de données `hiv_mort` que nous avons chargé précédemment.

Supposons que vous souhaitez visualiser les patients du jeu de données par leur mois de naissance. Nous pouvons le faire avec `ggplot` :

```
ggplot(hiv_mort) +  
  geom_bar(aes(x = birth_month))
```



Cependant, il y a un problème : l'axe des x (qui représente les mois) est classé alphabétiquement, avec Avril en premier à gauche, puis Août, etc. Mais les mois devraient suivre un ordre chronologique spécifique !

Nous pouvons arranger le graphique dans l'ordre souhaité en créant un facteur avec la fonction `factor()` :

```
hiv_mort_modified <-  
  hiv_mort %>%  
  mutate(birth_month = factor(x = birth_month,  
                             levels = c("Jan", "Feb", "Mar", "Apr",  
                                      "May", "Jun", "Jul", "Aug",  
                                      "Sep", "Oct", "Nov", "Dec")))
```

La syntaxe est simple : l'argument `x` prend la colonne de caractères d'origine, `birth_month`, et l'argument `levels` prend la séquence désirée de mois.

Lorsque nous inspectons le type de données de la variable `birth_month`, nous pouvons voir sa transformation :

```
# Modified dataset  
class(hiv_mort_modified$birth_month)
```

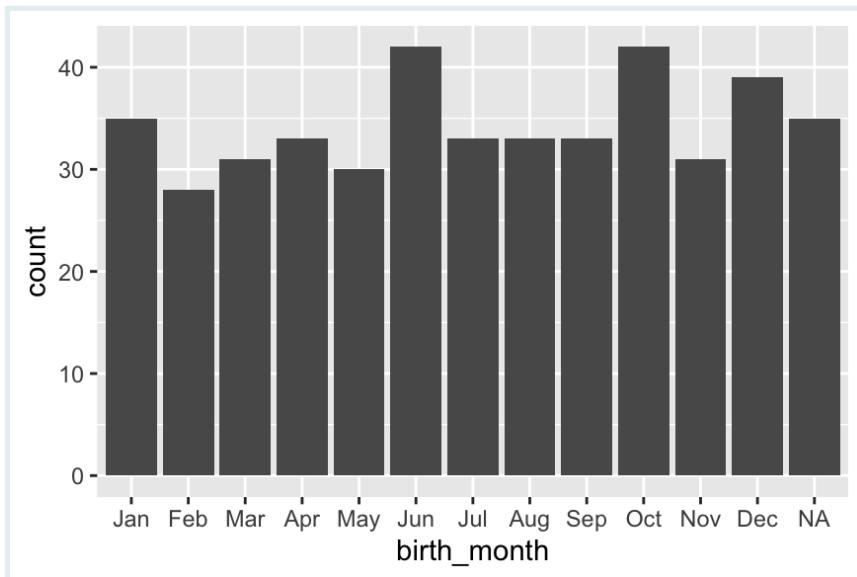
```
## [1] "factor"
```

```
# Original dataset  
class(hiv_mort$birth_month)
```

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

Maintenant, nous pouvons régénérer le ggplot avec le jeu de données modifié :

```
ggplot(hiv_mort_modified) +
  geom_bar(aes(x = birth_month))
```



Les mois sur l'axe des x sont maintenant affichés dans l'ordre que nous avons spécifié.

La nouvelle variable de facteur respectera également l'ordre défini dans d'autres contextes. Par exemple, comparez comment la fonction `count()` affiche les deux tableaux de fréquences ci-dessous :

```
# Original dataset
count(hiv_mort, birth_month)
```

```
## # A tibble: 13 × 2
##   birth_month     n
##   <chr>       <int>
## 1 Apr          33
## 2 Aug          33
## 3 Dec          39
## 4 Feb          28
## 5 Jan          35
## 6 Jul          33
## 7 Jun          42
## 8 Mar          31
## 9 May          30
## 10 Nov         31
## 11 Oct         42
```

```
## 12 Sep          33
## 13 <NA>         35
```

```
# Modified dataset
count(hiv_mort_modified, birth_month)
```

```
## # A tibble: 13 × 2
##   birth_month     n
##   <fct>       <int>
## 1 Jan            35
## 2 Feb            28
## 3 Mar            31
## 4 Apr            33
## 5 May            30
## 6 Jun            42
## 7 Jul            33
## 8 Aug            33
## 9 Sep            33
## 10 Oct           42
## 11 Nov           31
## 12 Dec           39
## 13 <NA>          35
```

Soyez vigilant lorsque vous créez des niveaux de facteurs ! Toutes les valeurs de la variable **qui ne sont pas incluses** dans l'ensemble des niveaux fournis à l'argument `levels` seront converties en NA.

Par exemple, si nous avons manqué certains mois dans notre exemple :

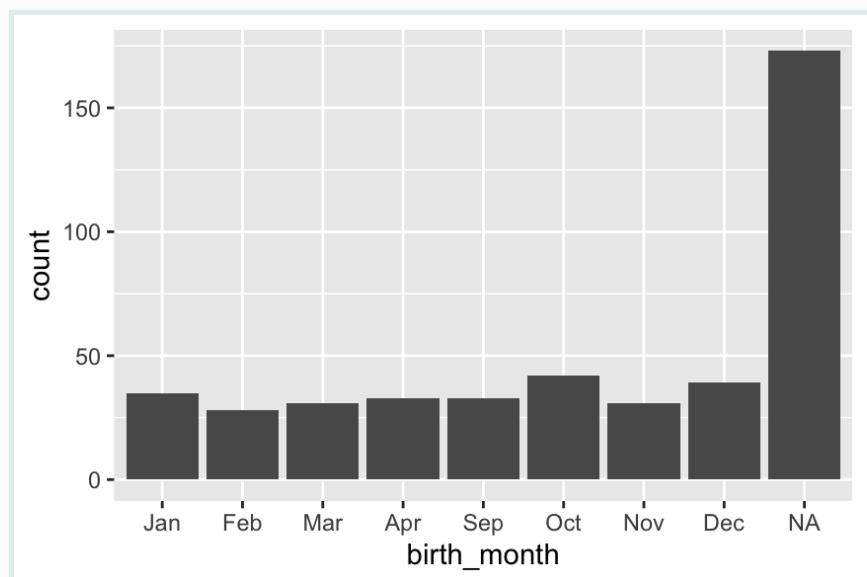
WATCH OUT



```
hiv_mort_missing_months <-
  hiv_mort %>%
  mutate(birth_month = factor(x = birth_month,
                             levels = c("Jan", "Feb", "Mar",
                                       "Apr",
                                       # missing months
                                       "Sep", "Oct", "Nov",
                                       "Dec")))
```

Nous finissons avec beaucoup de valeurs NA :

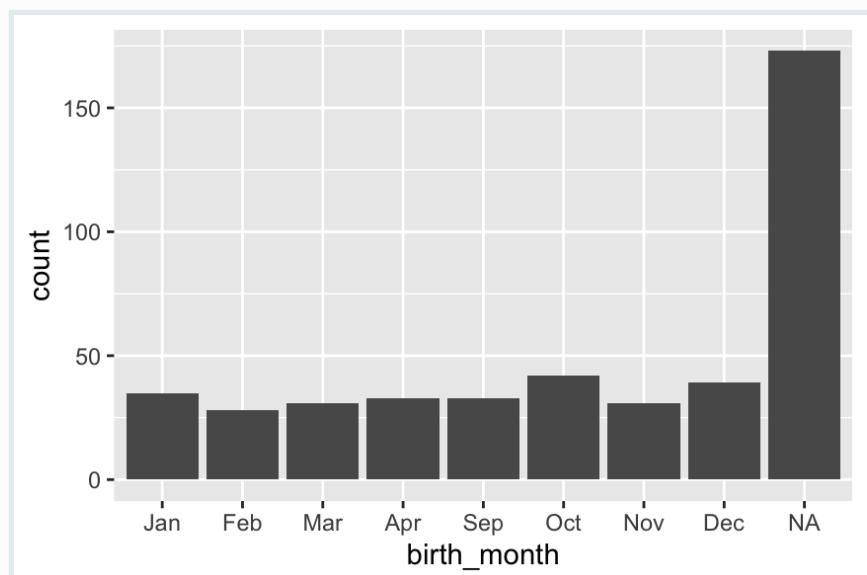
```
ggplot(hiv_mort_missing_months) +
  geom_bar(aes(x = birth_month))
```



Vous aurez le même problème s'il y a des erreurs de frappe :

```
hiv_mort_with_typos <-
  hiv_mort %>%
  mutate(birth_month = factor(x = birth_month,
                             levels = c("Jan", "Feb", "Mar",
                                         "Apr",
                                         "Moy", "Jon", "Jol",
                                         "Aog", # typos
                                         "Sep", "Oct", "Nov",
                                         "Dec")))

ggplot(hiv_mort_with_typos) +
  geom_bar(aes(x = birth_month))
```

WATCH OUT

Vous pouvez utiliser le facteur sans niveaux. Il utilise simplement l'arrangement par défaut (alphabétique) des niveaux.

```
hiv_mort_default_factor <- hiv_mort %>%
  mutate(birth_month = factor(x = birth_month))
```

SIDE NOTE

```
class(hiv_mort_default_factor$birth_month)
```

```
## [1] "factor"
```

```
levels(hiv_mort_default_factor$birth_month)
```

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

Q: Facteur de genre

En utilisant le jeu de données `hiv_mort`, convertissez la variable `gender` en un facteur avec les niveaux “Femme” et “Homme”, dans cet ordre.

Q: Repérage des erreurs

Quelles erreurs pouvez-vous repérer dans l'extrait de code suivant ? Quelles sont les conséquences de ces erreurs ?

```
hiv_mort <-
  hiv_mort %>%
  mutate(birth_month = factor (x = birth_month,
                               levels = c("Jan", "Feb", "Mar", "Apr",
                                         "Mai", "Jun", "Jul", "Sep",
                                         "Oct", "Nov.", "Dec")))
```

Q: Avantage des facteurs

Quel est l'avantage principal de l'utilisation de facteurs par rapport aux caractères pour les données catégorielles dans R ?

- a. Il est plus facile d'effectuer des manipulations de chaînes sur les facteurs.
 - b. Les facteurs permettent un meilleur contrôle de l'ordre des données catégorielles.
 - c. Les facteurs augmentent la précision des modèles statistiques.
-

Manipuler les facteurs avec `forcats`

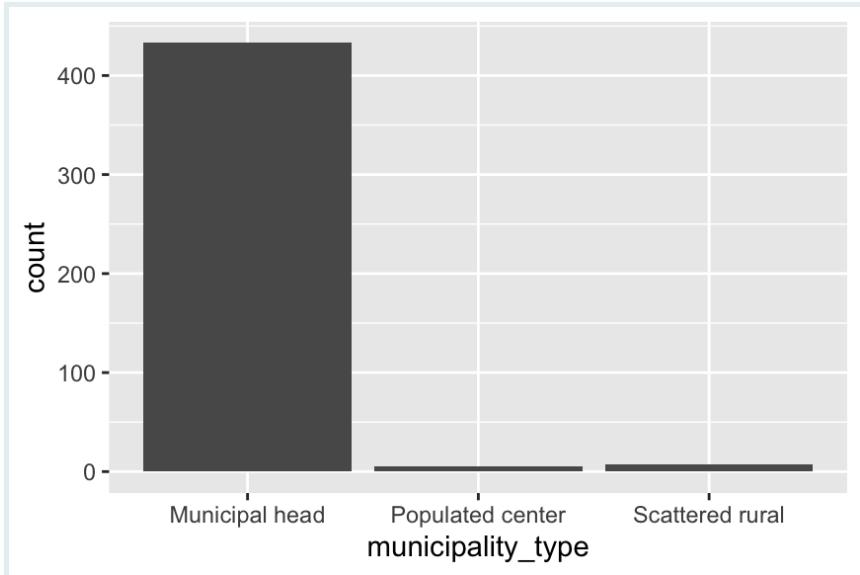
Les facteurs sont très utiles, mais ils peuvent parfois être un peu fastidieux à manipuler uniquement avec les fonctions de base R. Heureusement, le package `forcats`, membre du tidyverse, propose un ensemble de fonctions qui simplifient grandement la manipulation des facteurs. Nous allons examiner quatre fonctions ici, mais il y en a beaucoup d'autres, donc nous vous encourageons à explorer le site web de `forcats` par vous-même [ici](#)!

`fct_relevel`

La fonction `fct_relevel()` est utilisée pour changer manuellement l'ordre des niveaux de facteurs.

Par exemple, disons que nous voulons visualiser la fréquence des individus de notre jeu de données par type de municipalité. Lorsque nous créons un graphique en barres, les valeurs sont classées par ordre alphabétique par défaut :

```
ggplot(hiv_mort) +
  geom_bar(aes(x = municipality_type))
```



Mais que se passerait-il si nous voulions qu'une valeur spécifique, disons "Populated center", apparaisse en premier dans le graphique ?

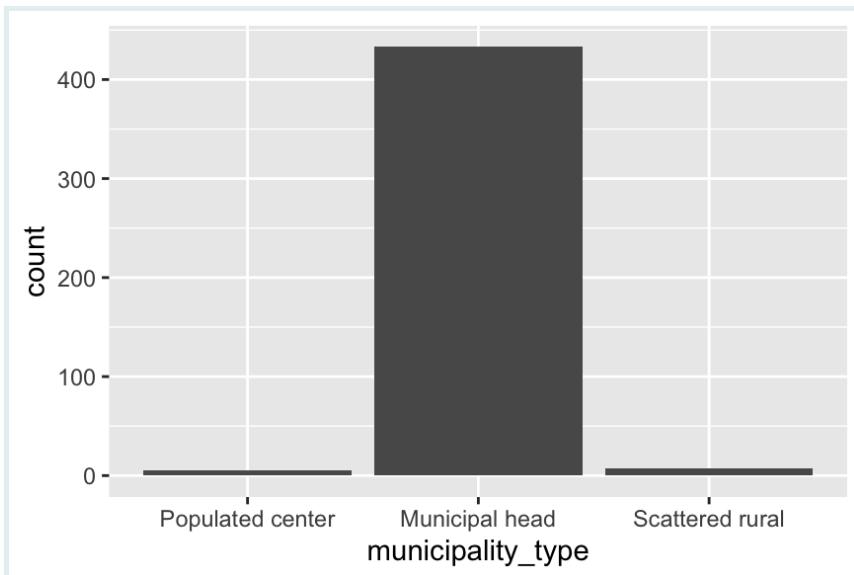
Cela peut être réalisé en utilisant `fct_relevel()`. Voici comment :

```
hiv_mort_pop_center_first <-
  hiv_mort %>%
  mutate(municipality_type = fct_relevel(municipality_type, "Populated
  center"))
```

La syntaxe est simple : nous passons la variable factorielle en premier argument, et le niveau que nous voulons déplacer au début en second argument.

Maintenant lorsque nous traçons :

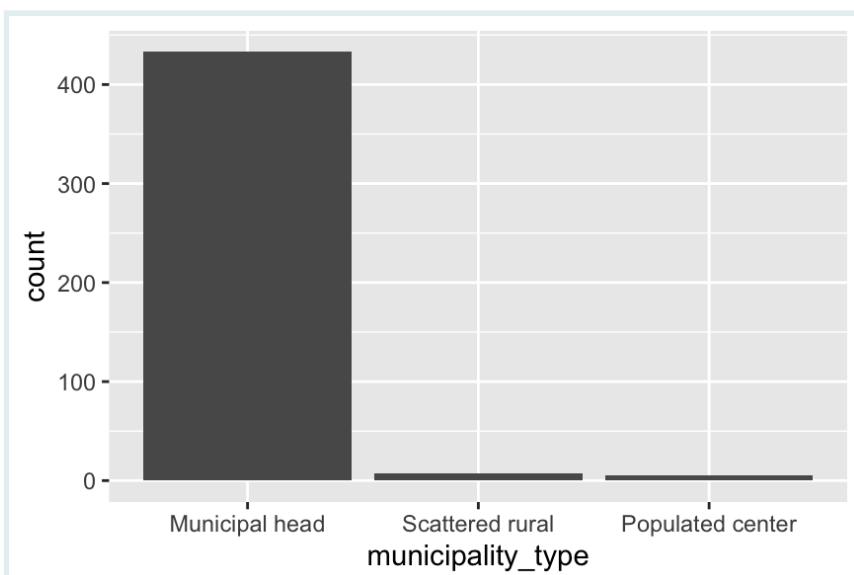
```
ggplot(hiv_mort_pop_center_first) +
  geom_bar(aes(x = municipality_type))
```



Le niveau "Centre peuplé" est maintenant le premier.

Nous pouvons déplacer le niveau "Populated center" à une position différente avec l'argument `after` :

```
hiv_mort %>%
  mutate(municipality_type = fct_relevel(municipality_type, "Populated
    center",
                                             after = 2)) %>%
# pipe directly into to plot to visualize change
  ggplot() +
  geom_bar(aes(x = municipality_type))
```

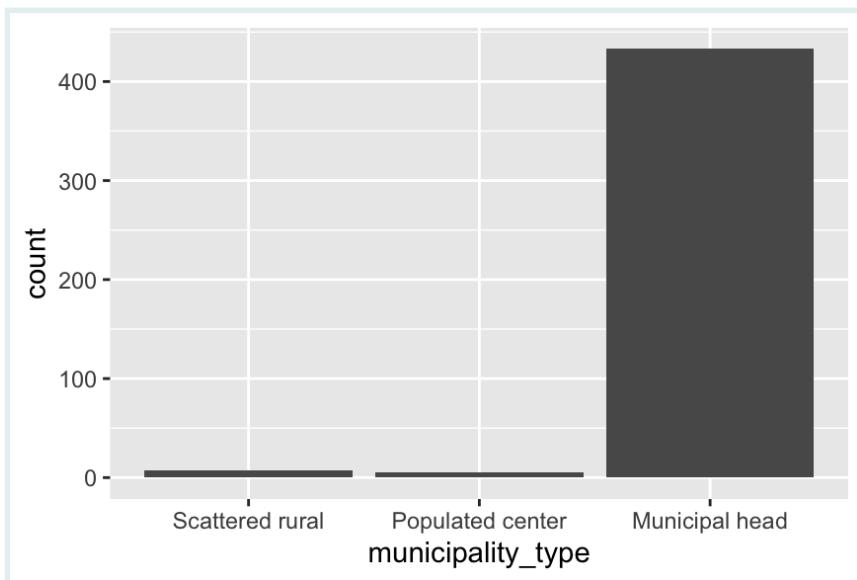


La syntaxe est : spécifier le facteur, le niveau à déplacer, et utiliser l'argument `after` pour définir à quelle position le placer après.

Nous pouvons également déplacer plusieurs niveaux à la fois en fournissant ces niveaux à `fct_relevel()` :

Ci-dessous, nous disposons tous les niveaux de facteurs pour le type de municipalité dans l'ordre souhaité :

```
hiv_mort %>%
  mutate(municipality_type = fct_relevel(municipality_type,
                                            "Scattered rural",
                                            "Populated center",
                                            "Municipal head")) %>%
  ggplot() +
  geom_bar(aes(x = municipality_type))
```



C'est similaire à la création d'un facteur depuis le début avec des niveaux dans cet ordre :

```
hiv_mort %>%
  mutate(municipality = factor(municipality_type,
                               levels = c("Scattered rural",
                                         "Populated center",
                                         "Municipal head")))
```



PRACTICE Q: Utiliser `fct_relevel`

En utilisant le jeu de données `hiv_mort`, convertissez la variable `death_location` en facteur de sorte que 'Home/address' soit le premier



niveau. Ensuite, créez un graphique en barres montrant le décompte des individus du jeu de données par `death_location`.

fct_reorder

`fct_reorder()` est utilisé pour réorganiser les niveaux d'un facteur en fonction des valeurs d'une autre variable.

Pour illustrer, créons un tableau récapitulatif avec le nombre de décès, l'âge moyen et médian au décès pour chaque municipalité :

```
summary_per_muni <-  
  hiv_mort %>%  
  group_by(municipality_name) %>%  
  summarise(n_deceased = n(),  
            mean_age_death = mean(age_at_death, na.rm = T),  
            med_age_death = median(age_at_death, na.rm = T))  
  
summary_per_muni  
  
## # A tibble: 25 × 4  
##   municipality_name n_deceased mean_age_death  
##   <chr>                <int>             <dbl>  
## 1 Aguadas                  2                 42  
## 2 Anserma                 15                37.4  
## 3 Aranzazu                 2                 37.5  
## 4 Belalcázar                4                 38.8  
## 5 Chinchiná                 62                43.6  
## 6 Filadelfia                 5                 42.6  
## 7 La Dorada                 46                41.0  
## 8 La Merced                 3                 27  
## 9 Manizales                 199               41.0  
## 10 Manzanares                 3                 38.3  
##   med_age_death  
##   <dbl>  
## 1 42  
## 2 37.5  
## 3 37.5  
## 4 41  
## 5 42.5  
## 6 43  
## 7 41  
## 8 28  
## 9 41  
## 10 34  
## # i 15 more rows
```

Lorsque nous traçons l'une des variables, nous voudrons peut-être arranger les niveaux de facteurs par cette variable numérique. Par exemple, pour ordonner la municipalité par la colonne de l'âge moyen :

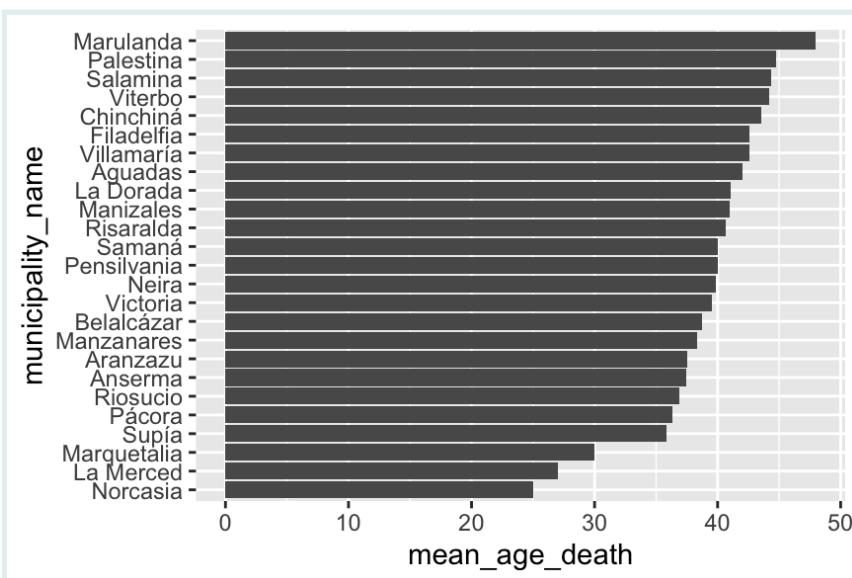
```
summary_per_muni_reordered <-
  summary_per_muni %>%
  mutate(municipality_name = fct_reorder(.f = municipality_name,
                                         .x = mean_age_death))
```

La syntaxe est :

- `.f` - le facteur à réorganiser
- `.x` - le vecteur numérique déterminant le nouvel ordre

Nous pouvons maintenant tracer un joli graphique en barres :

```
ggplot(summary_per_muni_reordered) +
  geom_col(aes(y = municipality_name, x = mean_age_death))
```



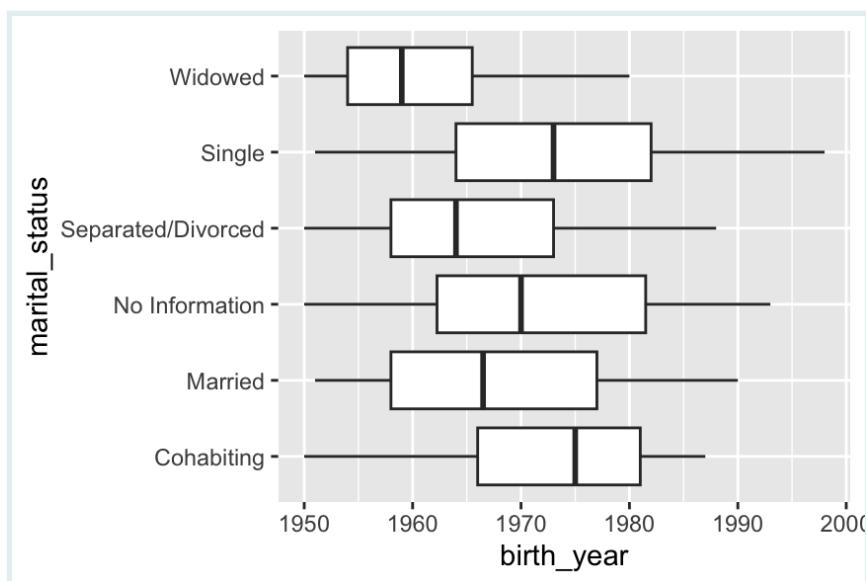
Q: Utiliser `fct_reorder`

En partant du dataframe `summary_per_muni`, réorganisez la municipalité (`municipality_name`) par la colonne `med_age_death` et tracez le graphique en barres réorganisé.

L'argument .fun

Parfois, nous voulons que les catégories de notre graphique apparaissent dans un ordre spécifique déterminé par une statistique sommaire. Par exemple, considérons le diagramme en boîte de `birth_year` par `marital_status` :

```
ggplot(hiv_mort, aes(y = marital_status, x = birth_year)) +  
  geom_boxplot()
```



Le diagramme en boîte affiche la médiane `birth_year` pour chaque catégorie de `marital_status` comme une ligne au milieu de chaque boîte. Nous voudrions peut-être arranger les catégories `marital_status` dans l'ordre de ces médianes. Mais si nous créons un tableau récapitulatif avec les médianes, comme nous l'avons fait précédemment avec `summary_per_muni`, nous ne pouvons pas créer de diagramme en boîte avec lui (allez regarder le dataframe `summary_per_muni` pour le vérifier vous-même).

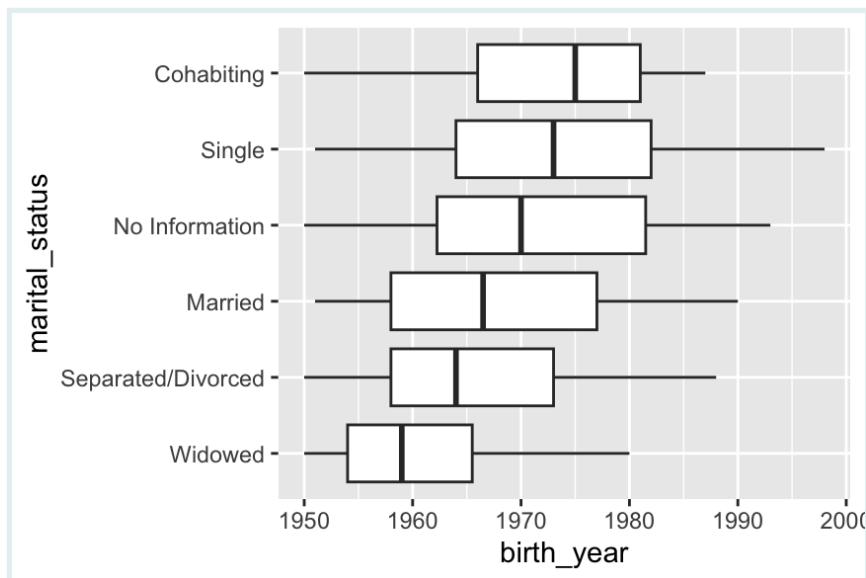
C'est là que intervient l'argument `.fun` de `fct_reorder()`. L'argument `.fun` nous permet de spécifier une fonction de résumé qui sera utilisée pour calculer le nouvel ordre des niveaux :

```
hiv_mort_arranged_marital <-  
  hiv_mort %>%  
  mutate(marital_status = fct_reorder(.f = marital_status,  
                                     .x = birth_year,  
                                     .fun = median,  
                                     na.rm = TRUE))
```

Dans ce code, nous réorganisons le facteur `marital_status` en fonction de la médiane de `birth_year`. Nous incluons l'argument `na.rm = TRUE` pour ignorer les valeurs NA lors du calcul de la médiane.

Maintenant, lorsque nous créons notre diagramme en boîte, les catégories `marital_status` sont classées par la médiane `birth_year` :

```
ggplot(hiv_mort_arranged_marital, aes(y = marital_status, x = birth_year)) +  
  geom_boxplot()
```



Nous pouvons voir que les individus ayant le statut marital “cohabiting” ont tendance à être les plus jeunes (ils sont nés les dernières années).



Q: Utiliser `.fun`

En utilisant le jeu de données `hiv_mort`, faites un diagramme en boîte de `birth_year` par `health_insurance_status`, où les catégories `health_insurance_status` sont disposées par la médiane `birth_year`.

fct_recode

La fonction `fct_recode()` nous permet de modifier manuellement les valeurs des niveaux de facteurs. Cette fonction peut être particulièrement utile lorsque vous devez renommer des catégories ou lorsque vous souhaitez fusionner plusieurs catégories en une seule.

Par exemple, nous pouvons renommer ‘Municipal head’ en ‘City’ dans la variable `municipality_type` :

```
hiv_mort_muni_recode <- hiv_mort %>%  
  mutate(municipality_type = fct_recode(municipality_type,
```

```
"City" = "Municipal head"))
# View the changelevels(hiv_mort_muni_recode$municipality_type)
```

```
## [1] "City"           "Populated center"
## [3] "Scattered rural"
```

Dans le code ci-dessus, `fct_recode()` prend deux arguments : la variable factorielle que vous souhaitez modifier (`municipality_type`) et l'ensemble des paires nom-valeur qui définissent le recodage. Le nouveau niveau ("City") est à gauche du signe égal et l'ancien niveau ("Municipal head") est à droite.

`fct_recode()` est particulièrement utile pour compresser plusieurs catégories en moins de niveaux.

Nous pouvons explorer cela en utilisant la variable `education_level`. Actuellement, elle possède six catégories :

```
count(hiv_mort, education_level)
```

```
## # A tibble: 6 × 2
##   education_level     n
##   <chr>             <int>
## 1 No information     88
## 2 None                22
## 3 Post-secondary      29
## 4 Preschool            3
## 5 Primary              187
## 6 Secondary             116
```

Par souci de simplicité, regroupons-les en seulement trois catégories - "primary & below", "secondary & above" et "others":

```
hiv_mort_educ_simple <-
  hiv_mort %>%
  mutate(education_level = fct_recode(education_level,
                                       "primary & below" = "Primary",
                                       "primary & below" = "Preschool",
                                       "secondary & above" = "Secondary",
                                       "secondary & above" = "Post-
                                       secondary",
                                       "others" = "No information",
                                       "others" = "None"))
```

Cela condense joliment les catégories :

```
count(hiv_mort_educ_simple, education_level)
```

```
## # A tibble: 3 × 2
##   education_level     n
##   <fct>             <int>
## 1 others              110
## 2 secondary & above  145
## 3 primary & below   190
```

Par mesure de précaution, nous pouvons arranger les niveaux dans un ordre raisonnable, avec "others" comme dernier niveau :

```
hiv_mort_educ_sorted <-
  hiv_mort_educ_simple %>%
  mutate(education_level = fct_relevel(education_level,
                                         "primary & below",
                                         "secondary & above",
                                         "others"))
```

Cela condense joliment les catégories :

```
count(hiv_mort_educ_sorted, education_level)
```

```
## # A tibble: 3 × 2
##   education_level     n
##   <fct>             <int>
## 1 primary & below   190
## 2 secondary & above 145
## 3 others              110
```

Q: Utiliser fct_recode



En utilisant le jeu de données `hiv_mort`, convertissez `death_location` en facteur.

Ensuite, utilisez `fct_recode()` pour renommer 'Public way' dans `death_location` en 'Public place'. Tracez les fréquences de la variable mise à jour.



`fct_recode` vs `case_when/if_else`

Vous vous demandez peut-être pourquoi nous avons besoin de `fct_recode()` alors que nous pouvons utiliser `case_when()` ou `if_else()` voire même `recode()` pour substituer des valeurs spécifiques. Le problème est que ces autres fonctions peuvent perturber votre variable factorielle.

Pour illustrer, disons que nous choisissons d'utiliser `case_when()` pour apporter une modification à la variable `education_level` du dataframe `hiv_mort_educ_sorted`.

Pour rappel, cette variable est un facteur avec trois niveaux, arrangés dans un ordre spécifié, avec "primary & below" en premier et "others" en dernier :

```
count(hiv_mort_educ_sorted, education_level)
```

```
## # A tibble: 3 × 2
##   education_level     n
##   <fct>             <int>
## 1 primary & below    190
## 2 secondary & above  145
## 3 others              110
```

SIDE NOTE



Disons que nous voulions remplacer "others" par "other", en enlevant le "s". Nous pouvons écrire :

```
hiv_mort_educ_other <-
  hiv_mort_educ_sorted %>%
  mutate(education_level = if_else(education_level ==
    "others",
    "other", education_level))
```

Après cette opération, la variable n'est plus un facteur :

```
class(hiv_mort_educ_other$education_level)
```

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

```
count(hiv_mort_educ_other, education_level)
```

```
## # A tibble: 3 × 2
##   education_level     n
##   <chr>             <int>
## 1 other              110
## 2 primary & below    190
## 3 secondary & above  145
```

Cependant, si nous avions utilisé `fct_recode()` pour le recodage, nous n'aurions pas ce problème :

```
hiv_mort_educ_other_fct <-
  hiv_mort_educ_simple %>%
  mutate(education_level = fct_recode(education_level, "other" =
  "others"))
```

SIDE NOTE



La variable reste un facteur :

```
class(hiv_mort_educ_other_fct$education_level)
```

```
## [1] "factor"
```

Et si nous créons un tableau ou un graphique, notre ordre est préservé : “primary”, “secondary”, puis “other”:

```
count(hiv_mort_educ_other_fct, education_level)
```

```
## # A tibble: 3 × 2
##   education_level     n
##   <fct>             <int>
## 1 other              110
## 2 secondary & above  145
## 3 primary & below   190
```

fct_lump

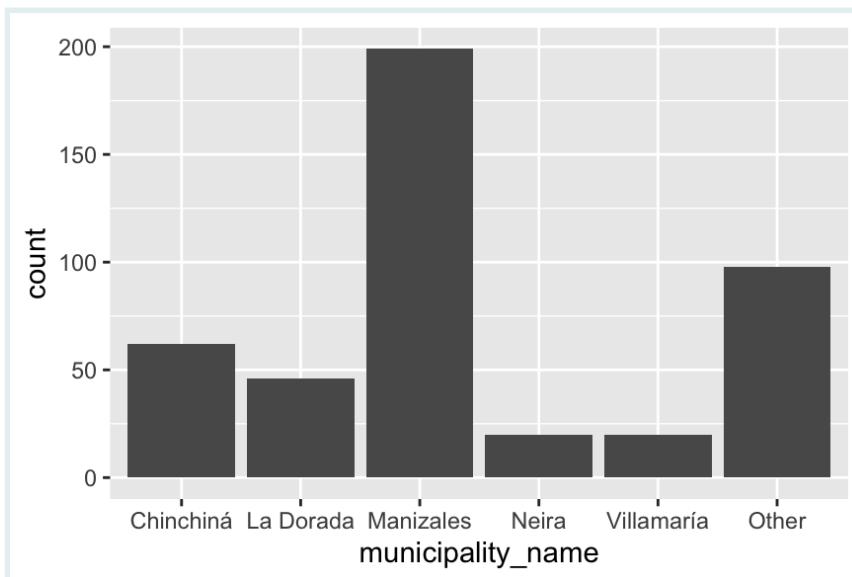
Parfois, nous avons trop de niveaux pour un tableau d'affichage ou un graphique, et nous voulons regrouper les niveaux les moins fréquents dans une seule catégorie, généralement appelée 'Other'.

C'est là que la fonction pratique `fct_lump()` intervient.

Dans l'exemple ci-dessous, nous regroupons les municipalités les moins fréquentes dans 'Other', en ne conservant que les 5 municipalités les plus fréquentes :

```
hiv_mort_lump_muni <- hiv_mort %>%
  mutate(municipality_name = fct_lump(municipality_name, n = 5))

ggplot(hiv_mort_lump_muni, aes(x = municipality_name)) +
  geom_bar()
```

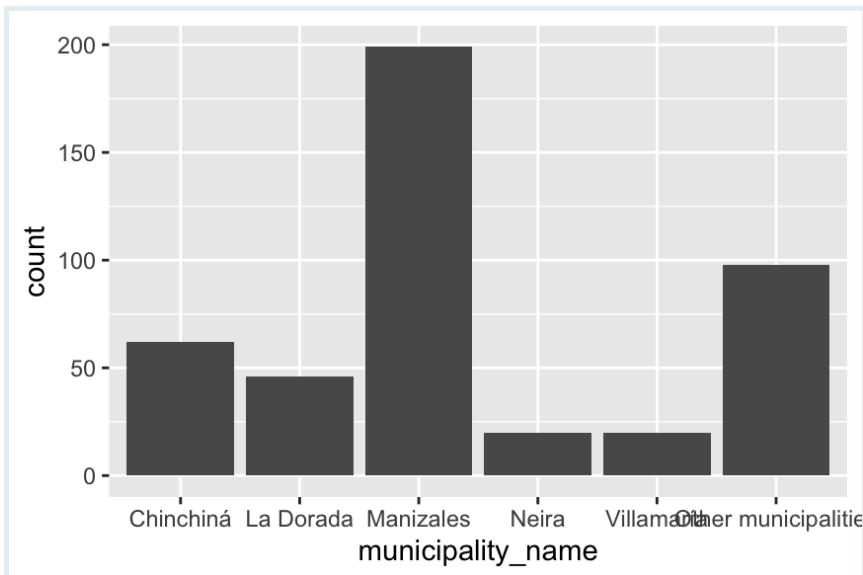


Dans l'utilisation ci-dessus, le paramètre `n = 5` signifie que les cinq municipalités les plus fréquentes sont conservées, et le reste est regroupé dans 'Other'.

Nous pouvons fournir un nom personnalisé pour l'autre catégorie avec l'argument `other_level`. Ci-dessous, nous utilisons le nom "Other municipalities".

```
hiv_mort_lump_muni_other_name <- hiv_mort %>%
  mutate(municipality_name = fct_lump(municipality_name, n = 5,
                                         other_level = "Other municipalities"))

ggplot(hiv_mort_lump_muni_other_name, aes(x = municipality_name)) +
  geom_bar()
```



De cette façon, `fct_lump()` est un outil pratique pour condenser les facteurs avec de nombreux niveaux peu fréquents en un nombre plus gérable de catégories.



Q: Utiliser `fct_lump`

En partant du jeu de données `hiv_mort`, utilisez `fct_lump()` pour créer un diagramme en barres avec la fréquence des 10 occupations les plus courantes.

Regroupez les occupations restantes dans une catégorie 'Other'.

Mettez `occupation` sur l'axe des y, et non sur l'axe des x, pour éviter le chevauchement des étiquettes.

Conclusion

Félicitations d'être arrivé jusqu'au bout. Dans cette leçon, vous avez appris les détails sur la classe de données, **les facteurs**, et comment les manipuler en utilisant des opérations de base comme `fct_relevel()`, `fct_reorder()`, `fct_recode()` et `fct_lump()`.

Bien qu'elles couvrent des tâches courantes comme le réordonnancement, le recodage et la fusion de niveaux, cette introduction ne fait qu'effleurer la surface de ce qui est

possible avec le package `forcats`. N'hésitez pas à explorer davantage sur le site web [forcats website](#).

Maintenant que vous comprenez les bases du travail avec les facteurs, vous êtes équipé pour représenter correctement vos données catégorielles dans R pour l'analyse et la visualisation en aval.

Corrigé

Q: Facteur de genre

```
hiv_mort_q1 <- hiv_mort %>%
  mutate(gender = factor(x = gender,
    levels = c("Female", "Male")))
```

Q: Repérage des erreurs

Erreurs : - "Mai" devrait être "May". - "Nov." a un point en trop. - "Aug" est manquant dans la liste des mois.

Conséquences :

Toutes les lignes avec les valeurs "May", "Nov" ou "Aug" pour `death_month` seront converties en NA dans la nouvelle variable `death_month`. Si vous créez des graphiques, `ggplot` supprimera ces niveaux avec seulement des valeurs NA.

Q: Avantage des facteurs

- Les facteurs permettent un meilleur contrôle de l'ordre des données catégorielles.

Les deux autres déclarations ne sont pas vraies.

Si vous voulez appliquer des opérations sur les chaînes de caractères comme `substr()`, `strsplit()`, `paste()`, etc., il est en fait plus simple d'utiliser des vecteurs de caractères que des facteurs.

Et bien que de nombreuses fonctions statistiques attendent des facteurs, et non des caractères, pour les prédicteurs catégoriels, cela ne les rend pas plus "précises".

Q: Utiliser `fct_relevel`

Q: Utiliser `fct_reorder`

[Q: Utiliser .fun](#)

[Q: Utiliser fct_recode](#)

[Q: Utiliser fct_lump](#)

Annexe : Codebook

Les variables du jeu de données sont :

- municipality : localisation municipale générale du patient [chr]
- death_location : lieu où le patient est décédé [chr]
- birth_date : date de naissance complète, formatée "YYYY-MM-DD" [date]
- birth_year : année de naissance du patient [dbl]
- birth_month : mois de naissance du patient [chr]
- birth_day : jour de naissance du patient [dbl]
- death_year : année de décès du patient [dbl]
- death_month : mois de décès du patient [chr]
- death_day : jour de décès du patient [dbl]
- gender : genre du patient [chr]
- education_level : plus haut niveau d'études atteint par le patient [chr]
- occupation : profession du patient [chr]
- racial_id : race du patient [chr]
- municipality_code : localisation municipale spécifique du patient [chr]
- primary_cause_death_description : cause primaire de décès du patient [chr]
- primary_cause_death_code : code de la cause primaire de décès [chr]
- secondary_cause_death_description : cause secondaire de décès du patient [chr]
- secondary_cause_death_code : code de la cause secondaire de décès [chr]
- tertiary_cause_death_description : cause tertiaire de décès du patient [chr]
- tertiary_cause_death_code : code de la cause tertiaire de décès [chr]
- quaternary_cause_death_description : cause quaternaire de décès du patient [chr]
- quaternary_cause_death_code : code de la cause quaternaire de décès [chr]

Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



CAMILLE BEATRICE VALERA

Project Manager and Scientific Collaborator, The GRAPH Network



KENE DAVID NWOSU

Data analyst, the GRAPH Network

Passionate about world improvement
