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

```
## [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 formatter 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
## # ℹ 5 more variables: date_debut_defaut <date>,
## #   date_fin_defaut <date>, date_debut_typique <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
## # ℹ 3 more variables: date_adm_abrege <chr>,
## #   date_adm_mixte <chr>, date_sortie_defaut <date>
```

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")
```

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"
```

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.

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
## # ℹ 102 more rows
## # ℹ 2 more variables: date_debut_longue <chr>, …
```

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
👉<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 courrament 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 précisement 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 0007-04-20
## 2 0022-04-20
## 3 0013-05-20
## 4 0013-05-20
## 5 0013-05-20
```

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.

Souvent, R vous retournera un message d'erreur si vous essayez de convertir des charactères ambiguës en dates sans spécifier l'ordre de

toujours le cas ! Vérifiez toujours que votre code s'est exécuté comme prévu et ne vous fiez jamais seulment 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 numérqiue (1-7 en 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
## # ℹ 5 more variables: date_debut_defaut <date>,
## #   date_fin_defaut <date>, date_debut_typique <date>, …
```

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 !

```r
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 ?

```r
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 :

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

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

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

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

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

```r
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.

```r
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.

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"))
```

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

```
## [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 changiez 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
NA
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 meilleur 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

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 numérqiue (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

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!)