
Travailler avec les Chaînes de Caractères en R

| | |
|---|--|
| Introduction | |
| Objectifs d'Apprentissage | |
| Paquets | |
| Définir des Chaînes | |
| Formatage des Chaînes en R avec {stringr} | |
| Changement de Casse | |
| Gestion des Espaces | |
| Mise en Forme du Texte | |
| Habillage du Texte | |
| Application du formatage de chaîne à un ensemble de données | |
| Division des chaînes de caractères avec <code>str_split()</code> et <code>separate()</code> | |
| Utilisation de <code>str_split()</code> | |
| Utilisation de <code>separate()</code> | |
| Séparation des Caractères Spéciaux | |
| Combinaison de Chaînes avec <code>paste()</code> | |
| Sous-référencement de chaînes avec <code>str_sub</code> | |
| Conclusion | |
| Clés de Réponses | |

Notes de Leçon | Travailler avec les Chaînes de Caractères en R

Introduction

La maîtrise de la manipulation des chaînes de caractères est une compétence essentielle pour les scientifiques de données. Des tâches telles que le nettoyage de données désordonnées et la mise en forme des sorties dépendent fortement de la capacité à analyser, combiner et modifier des chaînes de caractères. Cette leçon se concentre sur les techniques de travail avec les chaînes de caractères en R, en utilisant les fonctions du package {stringr} dans le tidyverse. Plongeons dedans !

Objectifs d'Apprentissage

- Comprendre le concept de chaînes de caractères et les règles pour les définir en R
- Utiliser des échappements pour inclure des caractères spéciaux comme des guillemets dans les chaînes
- Utiliser les fonctions de {stringr} pour formater les chaînes :
 - Changer la casse avec `str_to_lower()`, `str_to_upper()`, `str_to_title()`
 - Supprimer les espaces superflus avec `str_trim()` et `str_squish()`
 - Compléter les chaînes pour une largeur égale avec `str_pad()`
 - Envelopper le texte à une certaine largeur en utilisant `str_wrap()`
- Diviser les chaînes en parties en utilisant `str_split()` et `separate()`

- Extraire des sous-chaînes des chaînes en utilisant `str_sub()`

Paquets

```
# Chargement des paquets requis
if(!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse, here, janitor)
```

Définir des Chaînes

Il existe des règles fondamentales pour définir des chaînes de caractères en R.

Les chaînes peuvent être encadrées soit par des guillemets simples soit par des guillemets doubles. Cependant, le type de guillemet utilisé au début doit correspondre à celui utilisé à la fin. Par exemple :

```
string_1 <- "Bonjour" # Utilisation de guillemets doubles
string_2 <- 'Bonjour' # Utilisation de guillemets simples
```

Vous ne pouvez normalement pas inclure de guillemets doubles à l'intérieur d'une chaîne qui commence et se termine par des guillemets doubles. La même règle s'applique aux guillemets simples à l'intérieur d'une chaîne qui commence et se termine par des guillemets simples. Par exemple :

```
will_not_work <- "Guillemets doubles " au sein de guillemets doubles"
will_not_work <- 'Guillemets simples ' au sein de guillemets simples'
```

Mais vous pouvez inclure des guillemets simples à l'intérieur d'une chaîne qui commence et se termine par des guillemets doubles, et vice versa :

```
single_inside_double <- "Guillemets simples ' au sein de guillemets doubles"
```

Alternativement, vous pouvez utiliser le caractère d'échappement `\` pour inclure un guillemet simple ou double littéral à l'intérieur d'une chaîne :

```
single_quote <- 'Guillemets simples \' au sein de guillemets doubles'
double_quote <- "Guillemets doubles \" au sein de guillemets doubles"
```

Pour afficher ces chaînes telles qu'elles apparaîtraient dans la sortie, comme sur un graphique, utilisez `cat()` :

```
cat('Guillemets simples \' au sein de guillemets doubles')
```

```
## Guillemets simples ' au sein de guillemets doubles
```

```
cat("Guillemets doubles \" au sein de guillemets doubles")
```

```
## Guillemets doubles " au sein de guillemets doubles
```

`cat()` imprime ses arguments sans formatage supplémentaire.

SIDE NOTE



Puisque `\` est le caractère d'échappement, vous devez utiliser `\\` pour inclure un antislash littéral dans une chaîne :

```
backslash <- "Ceci est un antislash : \\  
cat(backslash)
```

```
## Ceci est un antislash : \
```

Q : Repérage d'Erreurs dans les Définitions de Chaînes

PRACTICE



(in RMD)

Ci-dessous, des tentatives de définition de chaînes de caractères en R, avec deux lignes sur cinq contenant une erreur. Identifiez et corrigez ces erreurs.

```
ex_a <- 'Elle a dit, "Bonjour !" à lui.'  
ex_b <- "Elle a dit \"Allons sur la lune\\\""  
ex_c <- "Ils ont été \"meilleurs amis\" pendant des années."  
ex_d <- 'Le journal de Jane\\'  
ex_e <- "C'est une journée ensoleillée !
```

Formatage des Chaînes en R avec {stringr}

Le package {stringr} en R fournit des fonctions utiles pour formater les chaînes pour l'analyse et la visualisation. Cela inclut les changements de casse, la gestion des espaces, la standardisation de la longueur et l'habillage du texte.

Changement de Casse

La conversion de la casse est souvent nécessaire pour standardiser les chaînes ou les préparer pour l'affichage. Le package {stringr} fournit plusieurs fonctions de changement de casse :

- `str_to_upper()` convertit les chaînes en majuscules.

```
str_to_upper("bonjour le monde")
```

```
## [1] "BONJOUR LE MONDE"
```

- `str_to_lower()` convertit les chaînes en minuscules.

```
str_to_lower("Au revoir")
```

```
## [1] "au revoir"
```

- `str_to_title()` met en majuscule la première lettre de chaque mot. Idéal pour titrer les noms, sujets, etc.

```
str_to_title("manipulation de chaîne")
```

```
## [1] "Manipulation De Chaîne"
```

Gestion des Espaces

Gérer les espaces rend les chaînes propres et uniformes. Le package {stringr} fournit deux fonctions principales pour cela :

- `str_trim()` supprime les espaces au début et à la fin.

```
str_trim(" espace coupé ")
```

```
## [1] "espace coupé"
```

- `str_squish()` supprime les espaces au début et à la fin, *et* réduit plusieurs espaces internes à un seul.

```
str_squish("  trop  d'espace  interne  ")
```

```
## [1] "trop d'espace interne"
```

```
# remarquez la différence avec str_trim  
str_trim("  trop  d'espace  interne  ")
```

```
## [1] "trop  d'espace  interne"
```

Mise en Forme du Texte

`str_pad()` ajoute des espaces à une chaîne pour obtenir une largeur fixe. Par exemple, nous pouvons ajouter des espaces au nombre 7 pour le forcer à avoir 3 caractères :

```
str_pad("7", width = 3, pad = "0") # Ajouter des espaces à gauche pour une  
longueur de 3 avec 0
```

```
## [1] "007"
```

Le premier argument est la chaîne à mettre en forme. `width` définit la largeur finale de la chaîne et `pad` spécifie le caractère de remplissage.

`side` contrôle si l'ajout d'espaces se fait à gauche ou à droite. L'argument `side` est par défaut "left", donc les espaces seront ajoutés à gauche si non spécifié. Spécifier `side = "right"` ajoute des espaces à droite :

```
str_pad("7", width = 4, side = "right", pad = "_") # Ajouter des espaces à  
droite pour une longueur de 4 avec _
```

```
## [1] "7___"
```

Ou nous pouvons ajouter des espaces des deux côtés :

```
str_pad("7", width = 5, side = "both", pad = "_") # Ajouter des espaces des
deux côtés pour une longueur de 5 avec _
```

```
## [1] "__7__"
```

Habillage du Texte

L'habillage du texte aide à adapter les chaînes dans des espaces restreints comme les titres de graphiques. La fonction `str_wrap()` habille le texte à une largeur définie.

Par exemple, pour habiller un texte à 10 caractères, nous pouvons écrire :

```
example_string <- "Manipulation de chaînes avec str_wrap peut améliorer la
lisibilité dans les graphiques."
wrapped_to_10 <- str_wrap(example_string, width = 10)
wrapped_to_10
```

```
## [1] "Manipulation\nde
chaînes\avec\nstr_wrap\npeut\naméliorer\nla\nlisibilité\ndans
les\graphiques."
```

La sortie peut paraître déroutante. Le `\n` indique un saut de ligne, et pour voir la modification correctement, nous devons utiliser la fonction `cat()`, qui est une version spéciale de `print()` :

```
cat(wrapped_to_10)
```

```
## Manipulation
## de chaînes
## avec
## str_wrap
## peut
## améliorer
## la
## lisibilité
## dans les
## graphiques.
```

Notez que la fonction conserve les mots entiers, donc elle ne divisera pas les mots plus longs comme "manipulation".

Définir la largeur à 1 divise essentiellement la chaîne en mots individuels :


```
cat(str_wrap(example_string, width = 1))
```

```
## Manipulation  
## de  
## chaînes  
## avec  
## str_wrap  
## peut  
## améliorer  
## la  
## lisibilité  
## dans  
## les  
## graphiques.
```

`str_wrap()` est particulièrement utile dans la création de graphiques avec `ggplot2`. Par exemple, en habillant un long titre pour éviter qu'il ne déborde du graphique :

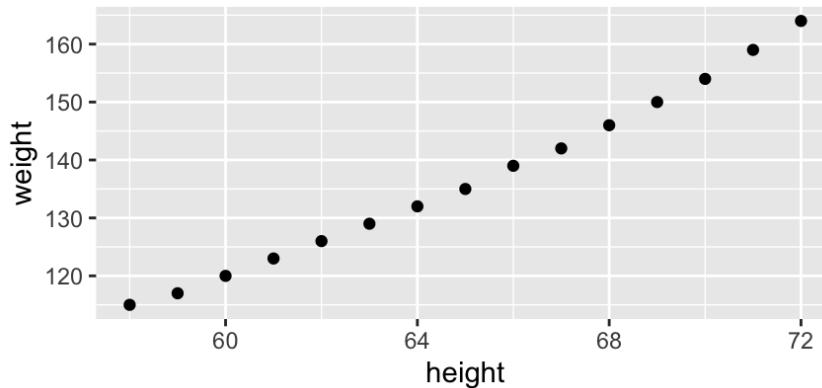
```
long_title <- "Ceci est un exemple de titre très long, qui dépasserait  
normalement de votre ggplot, mais vous pouvez l'habiller avec str_wrap pour  
l'adapter à une limite de caractères spécifiée."
```

```
# Sans habillage  
ggplot(women, aes(height, weight)) +  
  geom_point() +  
  labs(title = long_title)
```



```
# Avec habillage à 80 caractères  
ggplot(women, aes(height, weight)) +  
  geom_point() +  
  labs(title = str_wrap(long_title, width = 50))
```

Ceci est un exemple de titre très long, qui dépasserait normalement de votre ggplot, mais vous pouvez l'habiller avec `str_wrap` pour l'adapter à une limite de caractères spécifiée.



Ainsi, `str_wrap()` maintient les titres soigneusement à l'intérieur du graphique !

Q : Nettoyage des données de noms de patients

Un jeu de données contient des noms de patients avec un formatage inconsistant et des espaces blancs supplémentaires. Utilisez le package `{stringr}` pour standardiser ces informations :

```
patient_names <- c(" john doe", "ANNA SMITH  ", "Emily Davis")  
# 1. Supprimez les espaces blancs de chaque nom.  
# 2. Convertissez chaque nom en casse de titre pour la cohérence.
```

Q : Standardisation des codes de médicaments

PRACTICE



Les codes de médicaments suivants (fictifs) sont formatés de manière inconsistante. Standardisez-les en ajoutant des zéros pour garantir que tous les codes aient 8 caractères de long :

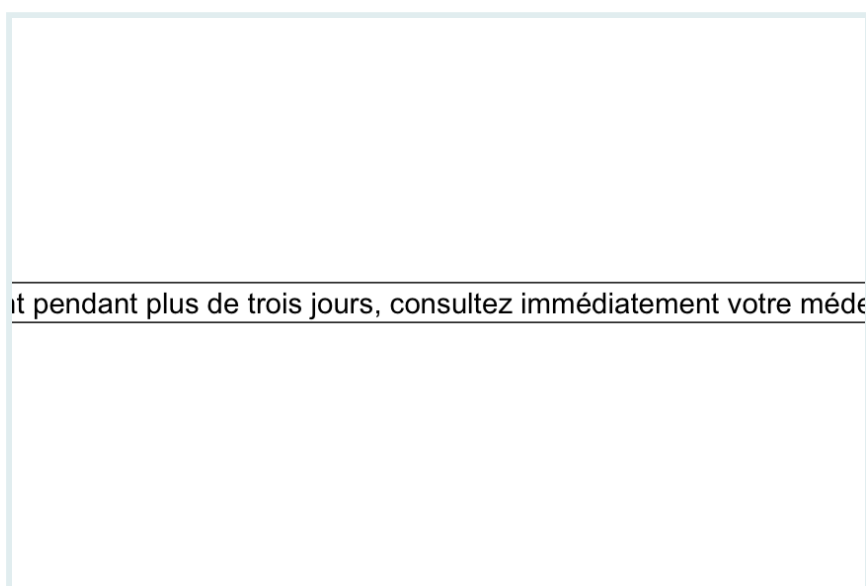
```
drug_codes <- c("12345", "678", "91011")  
# Ajoutez des zéros à chaque code à gauche pour une largeur  
fixe de 8 caractères.
```

Q : Habillage des instructions médicales

Utilisez `str_wrap()` pour formater ce qui suit pour une meilleure lisibilité :

```
instructions <- "Prenez deux comprimés quotidiennement après les repas. Si  
les symptômes persistent pendant plus de trois jours, consultez  
immédiatement votre médecin. Ne prenez pas plus que la dose recommandée.  
Tenir hors de portée des enfants."
```

```
ggplot(data.frame(x = 1, y = 1), aes(x, y, label = instructions)) +  
  geom_label() +  
  theme_void()
```



```
# Maintenant, habillez les instructions à une largeur de 50 caractères puis  
tracez à nouveau.
```

Application du formatage de chaîne à un ensemble de données

Maintenant, appliquons les fonctions de formatage de chaîne du package {stringr} pour nettoyer et standardiser un ensemble de données. Notre focus est sur un ensemble de données issu d'une étude sur les services de soins et de traitement du VIH dans la province de Zambézia, au Mozambique, disponible [ici](#). L'ensemble de données original comportait diverses incohérences de formatage, mais nous avons ajouté des erreurs supplémentaires à des fins éducatives.

D'abord, nous chargeons l'ensemble de données et examinons des variables spécifiques pour des problèmes potentiels.

```
# Charger l'ensemble de données  
hiv_dat_messy_1 <-  
openxlsx::read.xlsx(here::here("data/hiv_dat_messy_1.xlsx")) %>%  
  as_tibble()  
  
# Ces quatre variables contiennent des incohérences de formatage :
```

```
hiv_dat_messy_1 %>% select(district, health_unit, education, regimen)
```

```
## # A tibble: 1,413 × 4
##   district health_unit      education regimen
##   <chr>      <chr>          <chr>      <chr>
## 1 "Rural"    District Hospital Maganj... MISSING    AZT+3TC+NVP
## 2 "Rural"    District Hospital Maganj... secondary  TDF+3TC+EFV
## 3 "Urban"    24th Of July Health ... MISSING    tdf+3tc+efv
## 4 "Urban"    24th Of July Health ... MISSING    TDF+3TC+EFV
## 5 "Urban"    24th Of July Health ... University tdf+3tc+efv
## 6 "Urban"    24th Of July Health Faci... Technical  AZT+3TC+NVP
## 7 "Rural"    District Hospital Maganj... Technical  TDF+3TC+EFV
## 8 "Urban"    24th Of July Health Faci... Technical  azt+3tc+nvp
## 9 "Urban"    24th Of July Health Faci... Technical  AZT+3TC+NVP
## 10 "Urban"   24th Of July Health Faci... Technical  TDF+3TC+EFV
## # i 1,403 more rows
```

En utilisant la fonction `tabyl`, nous pouvons identifier et compter les valeurs uniques, révélant les incohérences :

```
# Comptage des valeurs uniques
hiv_dat_messy_1 %>% tabyl(health_unit)
```

```
##               health_unit    n    percent
##      24th Of July Health Facility 239 0.16914367
##      24th Of July Health Facility 249 0.17622081
## District Hospital Maganja Da Costa 342 0.24203822
## District Hospital Maganja Da Costa 336 0.23779193
##      Nante Health Facility 119 0.08421798
##      Nante Health Facility 128 0.09058740
```

```
hiv_dat_messy_1 %>% tabyl(education)
```

```
##   education    n    percent
##   MISSING 776 0.549186129
##   None 128 0.090587403
##   Primary 178 0.125973107
##   Secondary 82 0.058032555
##   Technical 17 0.012031139
##   University 4 0.002830856
##   primary 157 0.111111111
##   secondary 71 0.050247700
```

```
hiv_dat_messy_1 %>% tabyl(regimen)
```

```
##      regimen    n      percent valid_percent
## AZT+3TC+EFV   24 0.0169851380 0.0179910045
## AZT+3TC+NVP  229 0.1620665251 0.1716641679
## D4T+3TC+ABC    1 0.0007077141 0.0007496252
## D4T+3TC+EFV    2 0.0014154282 0.0014992504
## D4T+3TC+NVP   16 0.0113234253 0.0119940030
##      OTHER    1 0.0007077141 0.0007496252
## TDF+3TC+EFV  404 0.2859164897 0.3028485757
## TDF+3TC+NVP    3 0.0021231423 0.0022488756
## azt+3tc+efv   16 0.0113234253 0.0119940030
## azt+3tc+nvp  231 0.1634819533 0.1731634183
## d4t+3tc+efv    9 0.0063694268 0.0067466267
## d4t+3tc+nvp   18 0.0127388535 0.0134932534
## d4t+4tc+nvp    1 0.0007077141 0.0007496252
## d4t6+3tc+nvp   2 0.0014154282 0.0014992504
##      other    2 0.0014154282 0.0014992504
## tdf+3tc+efv  374 0.2646850672 0.2803598201
## tdf+3tc+nvp    1 0.0007077141 0.0007496252
##      <NA>    79 0.0559094126      NA
```

```
hiv_dat_messy_1 %>% tabyl(district)
```

```
## district    n      percent
##      Rural 234 0.16560510
##      Urban 118 0.08351026
##      Rural 691 0.48903043
##      Urban 370 0.26185421
```

Une autre fonction utile pour visualiser ces problèmes est `tbl_summary` du package `{gtsummary}` :

```
hiv_dat_messy_1 %>%
  select(district, health_unit, education, regimen) %>%
  tbl_summary()
```

| Characteristic | N = 1,413 ¹ |
|------------------------------------|------------------------|
| district | |
| Rural | 234 (17%) |
| Urban | 118 (8.4%) |
| Rural | 691 (49%) |
| Urban | 370 (26%) |
| health_unit | |
| 24th Of July Health Facility | 239 (17%) |
| 24th Of July Health Facility | 249 (18%) |
| District Hospital Maganja Da Costa | 342 (24%) |
| District Hospital Maganja Da Costa | 336 (24%) |
| Nante Health Facility | 119 (8.4%) |
| Nante Health Facility | 128 (9.1%) |
| education | |
| MISSING | 776 (55%) |
| None | 128 (9.1%) |
| primary | 157 (11%) |
| Primary | 178 (13%) |
| secondary | 71 (5.0%) |
| Secondary | 82 (5.8%) |
| Technical | 17 (1.2%) |
| University | 4 (0.3%) |
| regimen | |
| azt+3tc+efv | 16 (1.2%) |
| AZT+3TC+EFV | 24 (1.8%) |

| Characteristic | N = 1,413 ¹ |
|--------------------|------------------------|
| AZT+3TC+NVP | 229 (17%) |
| D4T+3TC+ABC | 1 (<0.1%) |
| d4t+3tc+efv | 9 (0.7%) |
| D4T+3TC+EFV | 2 (0.1%) |
| d4t+3tc+nvp | 18 (1.3%) |
| D4T+3TC+NVP | 16 (1.2%) |
| d4t+4tc+nvp | 1 (<0.1%) |
| d4t6+3tc+nvp | 2 (0.1%) |
| other | 2 (0.1%) |
| OTHER | 1 (<0.1%) |
| tdf+3tc+efv | 374 (28%) |
| TDF+3TC+EFV | 404 (30%) |
| tdf+3tc+nvp | 1 (<0.1%) |
| TDF+3TC+NVP | 3 (0.2%) |
| Unknown | 79 |
| ¹ n (%) | |

La sortie montre clairement des incohérences dans la casse, l'espacement et le format, donc nous devons les standardiser.

Ensuite, nous abordons ces problèmes de manière systématique :

```
hiv_dat_clean_1 <- hiv_dat_messy_1 %>%
  mutate(
    district = str_to_title(str_trim(district)), # Standardiser les noms de
    health_unit = str_squish(health_unit),      # Supprimer les espaces
    education = str_to_title(education),        # Standardiser les niveaux
    regimen = str_to_upper(regimen)             # Consistance dans la
  )
```

Et nous pouvons vérifier l'efficacité de ces changements en réexécutant la fonction `tbl_summary()` :

```
hiv_dat_clean_1 %>%
  select(district, health_unit, education, regimen) %>%
  tbl_summary()
```

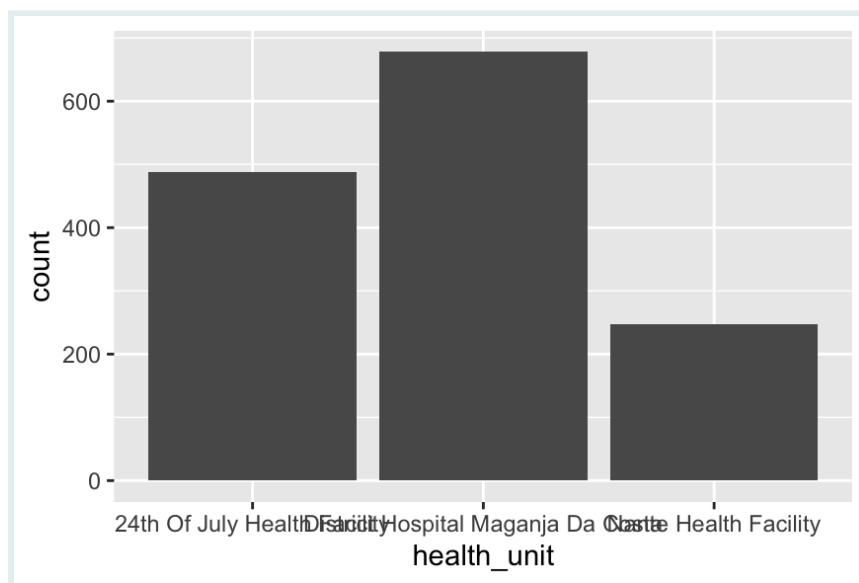
| Characteristic | N = 1,413 ¹ |
|------------------------------------|------------------------|
| district | |
| Rural | 925 (65%) |
| Urban | 488 (35%) |
| health_unit | |
| 24th Of July Health Facility | 488 (35%) |
| District Hospital Maganja Da Costa | 678 (48%) |
| Nante Health Facility | 247 (17%) |
| education | |
| Missing | 776 (55%) |
| None | 128 (9.1%) |
| Primary | 335 (24%) |
| Secondary | 153 (11%) |
| Technical | 17 (1.2%) |
| University | 4 (0.3%) |
| regimen | |
| AZT+3TC+EFV | 40 (3.0%) |
| AZT+3TC+NVP | 460 (34%) |
| D4T+3TC+ABC | 1 (<0.1%) |
| D4T+3TC+EFV | 11 (0.8%) |
| D4T+3TC+NVP | 34 (2.5%) |
| D4T+4TC+NVP | 1 (<0.1%) |
| D4T6+3TC+NVP | 2 (0.1%) |
| OTHER | 3 (0.2%) |
| TDF+3TC+EFV | 778 (58%) |

| Characteristic | N = 1,413 ¹ |
|--------------------|------------------------|
| Unknown | 79 |
| ¹ n (%) | |

Super !

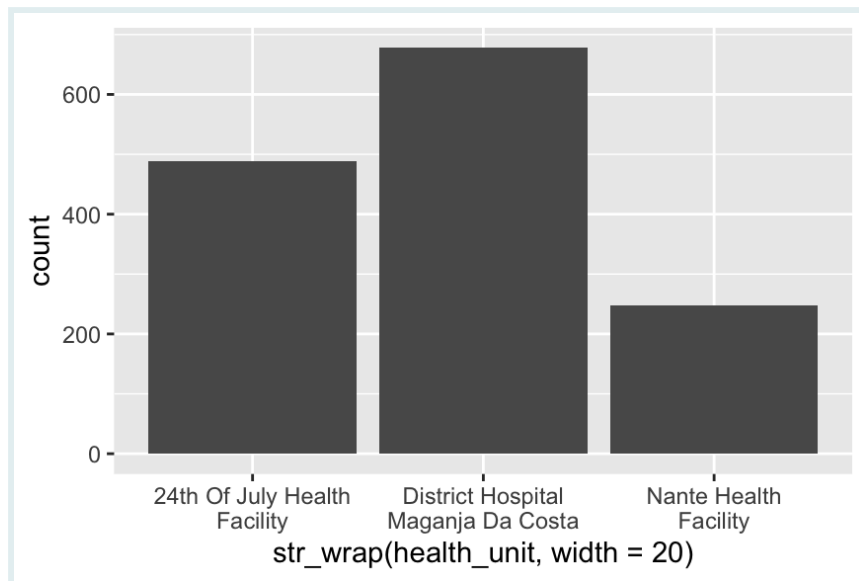
Enfin, essayons de tracer des comptages de la variable `health_unit`. Pour le style de tracé ci-dessous, nous rencontrons un problème avec des étiquettes longues :

```
ggplot(hiv_dat_clean_1, aes(x = health_unit)) +  
  geom_bar()
```



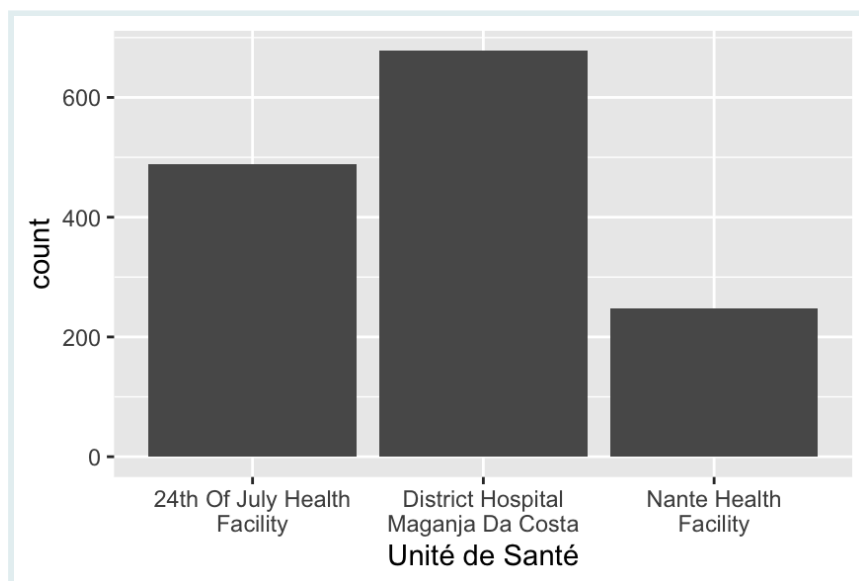
Pour résoudre cela, nous pouvons ajuster les étiquettes en utilisant `str_wrap()` :

```
hiv_dat_clean_1 %>%  
  ggplot(aes(x = str_wrap(health_unit, width = 20))) +  
  geom_bar()
```



Beaucoup plus propre, bien que nous devrions probablement corriger le titre de l'axe :

```
hiv_dat_clean_1 %>%
  ggplot(aes(x = str_wrap(health_unit, width = 20))) +
  geom_bar() +
  labs(x = "Unité de Santé")
```



Maintenant, essayez votre main sur des opérations de nettoyage similaires dans les questions de pratique ci-dessous.

PRACTICE



(in RMD)

Q : Formatage d'un ensemble de données sur la tuberculose

Dans cet exercice, vous nettoierez un ensemble de données, `lima_messy`, provenant d'une étude sur l'adhésion au traitement de la tuberculose à Lima, au Pérou. Plus de détails sur l'étude et l'ensemble de données sont disponibles [ici](#).

Commencez par importer l'ensemble de données :

```
lima_messy_1 <-  
openxlsx::read.xlsx(here::here("data/lima_messy_1.xlsx")) %>%  
  as_tibble()  
lima_messy_1
```



```
## # A tibble: 1,293 × 18  
##   id      age      sex marital_status  
##   <chr> <chr>   <chr> <chr>  
## 1 pe-1008 38 and older M      Single  
## 2 lm-1009 38 and older M      Married / cohabitating  
## 3 pe-1010 27 to 37   m      Married / cohabitating  
## 4 lm-1011 27 to 37   m      Married / cohabitating  
## 5 pe-1012 38 and older m      Married / cohabitating  
## 6 lm-1013 27 to 37   M      Single  
## 7 pe-1014 27 To 37   m      Married / cohabitating  
## 8 lm-1015 22 To 26   m      Single  
## 9 pe-1016 27 to 37   m      Single  
## 10 lm-1017 22 to 26   m      Single  
## # i 1,283 more rows  
## # i 14 more variables: poverty_level <chr>, ...
```

Votre tâche est de nettoyer les variables `marital_status`, `sex` et `age` dans `lima_messy`. Après le processus de nettoyage, générez un tableau récapitulatif en utilisant la fonction `tbl_summary()`. Visez à ce que votre sortie s'aligne sur cette structure :

| Caractéristique | N = 1,293 |
|--------------------|-----------|
| marital_status | |
| Divorcé / Séparé | 93 (7.2%) |
| Marié / Cohabitant | 486 (38%) |
| Célibataire | 677 (52%) |
| Veuf | 37 (2.9%) |
| sex | |
| F | 503 (39%) |

| Caractéristique | N = 1,293 |
|-----------------|-----------|
| age | |
| 21 ans et moins | 338 (26%) |
| 22 à 26 ans | 345 (27%) |
| 27 à 37 ans | 303 (23%) |
| 38 ans et plus | 307 (24%) |

Mettez en œuvre le nettoyage et résumez :

PRACTICE



```
# Créer un nouvel objet pour les données nettoyées
lima_clean <- lima_messy %>%
  mutate(
    # Nettoyer marital_status

    # Nettoyer sex

    # Nettoyer age

  )

# Vérifier le nettoyage
lima_clean %>%
  select(marital_status, sex, age) %>%
  tbl_summary()
```

Q : Habillage des étiquettes d'axe dans un graphique

À l'aide du jeu de données nettoyé `lima_clean` de la tâche précédente, créez un diagramme en barres pour afficher le nombre de participants par `statut_marital`. Ensuite, habillez les étiquettes de l'axe des x pour qu'elles n'aient pas plus de 15 caractères par ligne, afin d'améliorer la lisibilité.

```
# Créez votre diagramme en barres avec du texte habillé ici :
```

Division des chaînes de caractères avec `str_split()` et `separate()`

Diviser des chaînes de caractères est une tâche courante dans la manipulation de données. Le tidyverse offre des fonctions efficaces pour cette tâche, notamment `stringr::str_split()` et `tidyr::separate()`.

Utilisation de `str_split()`

La fonction `str_split()` est utile pour diviser des chaînes en parties. Par exemple :

```
exemple_chaine <- "diviser-cette-chaine"  
str_split(exemple_chaine, pattern = "-")
```

```
## [[1]]  
## [1] "diviser" "cette"  "chaine"
```

Ce code divise `exemple_chaine` à chaque trait d'union.

Cependant, appliquer `str_split()` directement à un dataframe peut être plus complexe.

Essayons-le avec le jeu de données IRS du Malawi comme étude de cas. Vous devriez déjà être familier avec ce jeu de données d'une leçon précédente. Il est disponible [ici](#). Pour l'instant, nous nous concentrerons sur la colonne `start_date_long` :

```
irs <- read_csv(here::here("data/Illovo_data.csv"))  
irs_dates_1 <- irs %>% select(village, start_date_long)  
irs_dates_1
```

```
## # A tibble: 112 × 2  
##   village      start_date_long  
##   <chr>      <chr>  
## 1 Mess      April 07 2014  
## 2 Nkombedzi April 22 2014  
## 3 B Compound May 13 2014  
## 4 D Compound May 13 2014  
## 5 Post Office May 13 2014  
## 6 Mangulenje May 15 2014  
## 7 Mangulenje Senior May 27 2014  
## 8 Old School May 27 2014  
## 9 Mwanza    May 28 2014  
## 10 Alumenda June 18 2014  
## # i 102 more rows
```

Supposons que nous voulions diviser la variable `start_date_long` pour extraire le jour, le mois et l'année. Nous pouvons écrire :

```
irs_dates_1 %>%  
  mutate(start_date_parts = str_split(start_date_long, " "))
```

```
## # A tibble: 112 × 3  
##   village      start_date_long start_date_parts
```

```
##      <chr>          <chr>          <list>
## 1 Mess            April 07 2014    <chr [3]>
## 2 Nkombedzi       April 22 2014    <chr [3]>
## 3 B Compound      May 13 2014     <chr [3]>
## 4 D Compound      May 13 2014     <chr [3]>
## 5 Post Office     May 13 2014     <chr [3]>
## 6 Mangulenje      May 15 2014     <chr [3]>
## 7 Mangulenje Senior May 27 2014    <chr [3]>
## 8 Old School      May 27 2014     <chr [3]>
## 9 Mwanza          May 28 2014     <chr [3]>
## 10 Alumenda       June 18 2014    <chr [3]>
## # i 102 more rows
```

Cela résulte en une colonne de liste, qui peut être difficile à utiliser. Pour la rendre plus lisible, nous pouvons utiliser `unnest_wider()` :

```
irs_dates_1 %>%
  mutate(start_date_parts = str_split(start_date_long, " ")) %>%
  unnest_wider(start_date_parts, names_sep = "_")
```

```
## # A tibble: 112 × 5
##   village      start_date_long start_date_parts_1
##   <chr>        <chr>          <chr>
## 1 Mess        April 07 2014    April
## 2 Nkombedzi   April 22 2014    April
## 3 B Compound  May 13 2014     May
## 4 D Compound  May 13 2014     May
## 5 Post Office May 13 2014     May
## 6 Mangulenje  May 15 2014     May
## 7 Mangulenje Senior May 27 2014    May
## 8 Old School  May 27 2014     May
## 9 Mwanza      May 28 2014     May
## 10 Alumenda   June 18 2014     June
## # i 102 more rows
## # i 2 more variables: start_date_parts_2 <chr>, ...
```

Ça fonctionne ! Nos parties de date sont maintenant séparées. Cependant, cette approche est assez encombrante. Une meilleure solution pour diviser les composants est la fonction `separate()`.

Utilisation de `separate()`

Essayons la même tâche avec `separate()` :

```
irs_dates_1 %>%
  separate(start_date_long, into = c("mois", "jour", "année"), sep = " ")
```

```
## # A tibble: 112 × 4
##   village      mois jour année
##   <chr>      <chr> <chr> <chr>
## 1 Mess      April 07    2014
## 2 Nkombedzi April 22    2014
## 3 B Compound May    13    2014
## 4 D Compound May    13    2014
## 5 Post Office May    13    2014
## 6 Mangulenje May    15    2014
## 7 Mangulenje Senior May    27    2014
## 8 Old School May    27    2014
## 9 Mwanza    May    28    2014
## 10 Alumenda June   18    2014
## # i 102 more rows
```

Bien plus simple !

Cette fonction nécessite de spécifier :

- La colonne à diviser.
- into - Noms des nouvelles colonnes.
- sep - Le caractère séparateur.

Pour conserver la colonne originale, utilisez `remove = FALSE` :

```
irs_dates_1 %>%
  separate(start_date_long, into = c("mois", "jour", "année"), sep = " ",
  remove = FALSE)
```

```
## # A tibble: 112 × 5
##   village      start_date_long mois jour année
##   <chr>      <chr>      <chr> <chr> <chr>
## 1 Mess      April 07 2014    April 07    2014
## 2 Nkombedzi April 22 2014    April 22    2014
## 3 B Compound May 13 2014      May    13    2014
## 4 D Compound May 13 2014      May    13    2014
## 5 Post Office May 13 2014      May    13    2014
## 6 Mangulenje May 15 2014      May    15    2014
## 7 Mangulenje Senior May 27 2014      May    27    2014
## 8 Old School May 27 2014      May    27    2014
## 9 Mwanza    May 28 2014      May    28    2014
## 10 Alumenda June 18 2014      June   18    2014
## # i 102 more rows
```

SIDE NOTE



Alternativement, le package `lubridate` offre des fonctions pour extraire les composants des dates :

```

irs_dates_1 %>%
  mutate(start_date_long = mdy(start_date_long)) %>%
  mutate(jour = day(start_date_long),
         mois = month(start_date_long, label = TRUE),
         année = year(start_date_long))

```

SIDE NOTE



```

## # A tibble: 112 × 5
##   village      start_date_long   jour mois   année
##   <chr>      <date>         <int> <ord> <dbl>
## 1 Mess      2014-04-07         7 Apr   2014
## 2 Nkombedzi 2014-04-22        22 Apr   2014
## 3 B Compound 2014-05-13        13 May   2014
## 4 D Compound 2014-05-13        13 May   2014
## 5 Post Office 2014-05-13        13 May   2014
## 6 Mangulenje 2014-05-15        15 May   2014
## 7 Mangulenje Senior 2014-05-27        27 May   2014
## 8 Old School 2014-05-27        27 May   2014
## 9 Mwanza     2014-05-28        28 May   2014
## 10 Alumenda  2014-06-18        18 Jun   2014
## # i 102 more rows

```

Lorsque certaines lignes manquent de toutes les parties nécessaires, `separate()` émettra un avertissement. Démontrons cela en supprimant artificiellement toutes les instances du mot "April" de nos dates :

```

irs_dates_with_problem <-
  irs_dates_1 %>%
  mutate(start_date_missing = str_replace(start_date_long, "April ", ""))
irs_dates_with_problem

```

```

## # A tibble: 112 × 3
##   village      start_date_long start_date_missing
##   <chr>      <chr>         <chr>
## 1 Mess      April 07 2014  07 2014
## 2 Nkombedzi April 22 2014  22 2014
## 3 B Compound May 13 2014    May 13 2014
## 4 D Compound May 13 2014    May 13 2014
## 5 Post Office May 13 2014    May 13 2014
## 6 Mangulenje May 15 2014    May 15 2014
## 7 Mangulenje Senior May 27 2014    May 27 2014
## 8 Old School May 27 2014    May 27 2014
## 9 Mwanza     May 28 2014    May 28 2014
## 10 Alumenda  June 18 2014   June 18 2014
## # i 102 more rows

```

Maintenant, essayons de diviser les parties de la date :


```

irs_dates_with_problem %>%
  separate(start_date_missing, into = c
("mois", "jour", "année"), sep = " ")

```

```

## # A tibble: 112 × 5
##   village      start_date_long mois   jour année
##   <chr>      <chr>          <chr> <chr> <chr>
## 1 Mess      April 07 2014    07    2014 <NA>
## 2 Nkombedzi April 22 2014    22    2014 <NA>
## 3 B Compound May 13 2014     May    13    2014
## 4 D Compound May 13 2014     May    13    2014
## 5 Post Office May 13 2014     May    13    2014
## 6 Mangulenje May 15 2014     May    15    2014
## 7 Mangulenje Senior May 27 2014     May    27    2014
## 8 Old School May 27 2014     May    27    2014
## 9 Mwanza    May 28 2014     May    28    2014
## 10 Alumenda  June 18 2014    June    18    2014
## # i 102 more rows

```

Comme vous pouvez le voir, les lignes manquant de parties produiront des avertissements. Gérez ces avertissements avec soin, car ils peuvent conduire à des données inexactes. Dans ce cas, nous avons maintenant l'information du jour et du mois pour ces lignes dans les mauvaises colonnes.

Q : Division des chaînes de tranches d'âge

Considérez le jeu de données `esoph_ca`, du package `{medicaldata}`, qui implique une étude cas-témoins sur le cancer de l'œsophage en France.

```

medicaldata::esoph_ca %>% as_tibble()

```

```

## # A tibble: 88 × 5
##   agegp alcgp      tobgp   ncases ncontrols
##   <ord> <ord>      <ord>   <dbl>   <dbl>
## 1 25-34 0-39g/day 0-9g/day     0      40
## 2 25-34 0-39g/day 10-19       0      10
## 3 25-34 0-39g/day 20-29       0       6
## 4 25-34 0-39g/day 30+         0       5
## 5 25-34 40-79    0-9g/day     0      27
## 6 25-34 40-79    10-19       0       7
## 7 25-34 40-79    20-29       0       4
## 8 25-34 40-79    30+         0       7
## 9 25-34 80-119   0-9g/day     0       2
## 10 25-34 80-119   10-19       0       1
## # i 78 more rows

```

Divisez les tranches d'âge dans la colonne `agegp` en deux colonnes distinctes : `agegp_inferieur` et `agegp_superieur`.

Après avoir utilisé la fonction `separate()`, le groupe d'âge "75+" nécessitera un traitement spécial. Utilisez `readr::parse_number()` ou une autre méthode pour convertir la limite d'âge inférieure ("75+") en nombre.

```
medicaldata::esoph_ca %>%
  separate(_____) %>%
  # convertir 75+ en nombre
  mutate(_____)
```

Séparation des Caractères Spéciaux

Pour utiliser la fonction `separate()` sur des caractères spéciaux comme le point (.), nous devons les échapper avec un double antislash (\\).

Considérez le scénario où les dates sont formatées avec des points :

```
irs_with_period <- irs_dates_1 %>%
  mutate(start_date_long = format(lubridate::mdy(start_date_long),
    "%d.%m.%Y"))
irs_with_period
```

```
## # A tibble: 112 × 2
##   village      start_date_long
##   <chr>      <chr>
## 1 Mess      07.04.2014
## 2 Nkombedzi 22.04.2014
## 3 B Compound 13.05.2014
## 4 D Compound 13.05.2014
## 5 Post Office 13.05.2014
## 6 Mangulenje 15.05.2014
## 7 Mangulenje Senior 27.05.2014
## 8 Old School 27.05.2014
## 9 Mwanza     28.05.2014
## 10 Alumenda 18.06.2014
## # i 102 more rows
```

Tenter de séparer ce format de date directement avec `sep = "."` ne fonctionnera pas :

```
irs_with_period %>%
  separate(start_date_long, into = c("day", "month", "year"), sep = ".")
```

```
## # A tibble: 112 × 4
##   village      day month year
```

```
##      <chr>          <chr> <chr> <chr>
## 1 Mess            ""      ""      ""
## 2 Nkombedzi       ""      ""      ""
## 3 B Compound      ""      ""      ""
## 4 D Compound      ""      ""      ""
## 5 Post Office     ""      ""      ""
## 6 Mangulenje      ""      ""      ""
## 7 Mangulenje Senior ""      ""      ""
## 8 Old School      ""      ""      ""
## 9 Mwanza          ""      ""      ""
## 10 Alumenda       ""      ""      ""
## # i 102 more rows
```

Cela ne fonctionne pas comme prévu car, dans les expressions régulières (regex), le point est un caractère spécial. Nous en apprendrons davantage à ce sujet en temps voulu. La bonne approche consiste à échapper le point en utilisant un double antislash (\) :

```
irs_with_period %>%
  separate(start_date_long, into = c("day", "month", "year"), sep = "\\.")
```

```
## # A tibble: 112 × 4
##   village      day month year
##   <chr>      <chr> <chr> <chr>
## 1 Mess      07     04  2014
## 2 Nkombedzi 22     04  2014
## 3 B Compound 13     05  2014
## 4 D Compound 13     05  2014
## 5 Post Office 13     05  2014
## 6 Mangulenje 15     05  2014
## 7 Mangulenje Senior 27     05  2014
## 8 Old School 27     05  2014
## 9 Mwanza     28     05  2014
## 10 Alumenda  18     06  2014
## # i 102 more rows
```

Maintenant, la fonction comprend qu'elle doit diviser la chaîne à chaque point littéral.

De même, lors de l'utilisation d'autres caractères spéciaux comme +, * ou ?, nous devons également les précéder d'un double antislash (\) dans l'argument sep.

Qu'est-ce qu'un Caractère Spécial ?

SIDE NOTE



Dans les expressions régulières, qui aident à trouver des motifs dans le texte, les caractères spéciaux ont des rôles spécifiques. Par exemple, un point (.) est un caractère générique qui peut représenter n'importe quel caractère. Ainsi, dans une recherche, "do.t" pourrait correspondre à

SIDE NOTE



exemple, "ho+se" correspondrait à "hose" ou "hooose" mais pas à "hse." Lorsque nous avons besoin d'utiliser ces caractères dans leurs rôles ordinaires, nous utilisons un double antislash (\\) devant eux, comme "\\." ou "\\+." Nous en apprendrons plus sur ces caractères spéciaux dans une leçon future.

Q : Séparation des Caractères Spéciaux

Votre prochaine tâche concerne le jeu de données `hiv_dat_clean_1`. Concentrez-vous sur la colonne `regimen`, qui liste les régimes de médicaments séparés par un signe `+`. Votre objectif est de diviser cette colonne en trois nouvelles colonnes : `drug_1`, `drug_2` et `drug_3` en utilisant la fonction `separate()`. Faites très attention à la façon dont vous gérez le séparateur `+`. Voici la colonne :

PRACTICE



(in RMD)

```
hiv_dat_clean_1 %>%  
  select(regimen)
```

```
## # A tibble: 1,413 × 1  
##   regimen  
##   <chr>  
## 1 AZT+3TC+NVP  
## 2 TDF+3TC+EFV  
## 3 TDF+3TC+EFV  
## 4 TDF+3TC+EFV  
## 5 TDF+3TC+EFV  
## 6 AZT+3TC+NVP  
## 7 TDF+3TC+EFV  
## 8 AZT+3TC+NVP  
## 9 AZT+3TC+NVP  
## 10 TDF+3TC+EFV  
## # i 1,403 more rows
```

Combinaison de Chaînes avec `paste()`

La fonction `paste()` dans R concatène ou joint ensemble des chaînes de caractères. Cela vous permet de combiner plusieurs chaînes en une seule.

```
string1 <- "Hello"
string2 <- "World"
paste(string1, string2)
```

```
## [1] "Hello World"
```

Le séparateur par défaut est un espace, donc cela renvoie "Hello World".

Démontrons comment utiliser cela sur un ensemble de données, avec les données de date de l'IRS. D'abord, nous séparerons la date de début en colonnes individuelles :

```
irs_dates_separated <- # stocker pour une utilisation ultérieure
  irs_dates_1 %>%
  separate(start_date_long, into = c("month", "day", "year"), sep = " ",
    remove = FALSE)
irs_dates_separated
```

```
## # A tibble: 112 × 5
##   village      start_date_long month day   year
##   <chr>      <chr>          <chr> <chr> <chr>
## 1 Mess      April 07 2014    April 07   2014
## 2 Nkombedzi April 22 2014    April 22   2014
## 3 B Compound May 13 2014      May 13     2014
## 4 D Compound May 13 2014      May 13     2014
## 5 Post Office May 13 2014      May 13     2014
## 6 Mangulenje May 15 2014      May 15     2014
## 7 Mangulenje Senior May 27 2014    May 27     2014
## 8 Old School May 27 2014      May 27     2014
## 9 Mwanza     May 28 2014      May 28     2014
## 10 Alumenda  June 18 2014     June 18     2014
## # i 102 more rows
```

Ensuite, nous pouvons recombinaer jour, mois et année avec `paste()` :

```
irs_dates_separated %>%
  select(day, month, year) %>%
  mutate(start_date_long_2 = paste(day, month, year))
```

```
## # A tibble: 112 × 4
##   day month year start_date_long_2
##   <chr> <chr> <chr> <chr>
## 1 07 April 2014 07 April 2014
## 2 22 April 2014 22 April 2014
## 3 13 May 2014 13 May 2014
## 4 13 May 2014 13 May 2014
## 5 13 May 2014 13 May 2014
## 6 15 May 2014 15 May 2014
```

```
## 7 27 May 2014 27 May 2014
## 8 27 May 2014 27 May 2014
## 9 28 May 2014 28 May 2014
## 10 18 June 2014 18 June 2014
## # i 102 more rows
```

L'argument `sep` spécifie le séparateur entre les éléments. Pour un séparateur différent, comme un trait d'union, nous pouvons é

crire :

```
irs_dates_separated %>%
  mutate(start_date_long_2 = paste(day, month, year, sep = "-"))
```

```
## # A tibble: 112 × 6
##   village      start_date_long month day   year
##   <chr>      <chr>      <chr> <chr> <chr>
## 1 Mess      April 07 2014   April 07   2014
## 2 Nkombedzi April 22 2014   April 22   2014
## 3 B Compound May 13 2014     May 13     2014
## 4 D Compound May 13 2014     May 13     2014
## 5 Post Office May 13 2014     May 13     2014
## 6 Mangulenje May 15 2014     May 15     2014
## 7 Mangulenje Senior May 27 2014     May 27     2014
## 8 Old School May 27 2014     May 27     2014
## 9 Mwanza     May 28 2014     May 28     2014
## 10 Alumenda   June 18 2014     June 18     2014
## # i 102 more rows
## # i 1 more variable: start_date_long_2 <chr>
```

Pour concaténer sans espaces, nous pouvons définir `sep = ""` :

```
irs_dates_separated %>%
  select(day, month, year) %>%
  mutate(start_date_long_2 = paste(day, month, year, sep = ""))
```

```
## # A tibble: 112 × 4
##   day month year start_date_long_2
##   <chr> <chr> <chr> <chr>
## 1 07 April 2014 07April2014
## 2 22 April 2014 22April2014
## 3 13 May 2014 13May2014
## 4 13 May 2014 13May2014
## 5 13 May 2014 13May2014
## 6 15 May 2014 15May2014
## 7 27 May 2014 27May2014
## 8 27 May 2014 27May2014
## 9 28 May 2014 28May2014
```

```
## 10 18    June 2014 18June2014
## # i 102 more rows
```

Où nous pouvons utiliser la fonction `paste0()`, qui est équivalente à `paste(..., sep = "")` :

```
irs_dates_separated %>%
  select(day, month, year) %>%
  mutate(start_date_long_2 = paste0(day, month, year))
```

```
## # A tibble: 112 × 4
##   day  month year start_date_long_2
##   <chr> <chr> <chr> <chr>
## 1 07    April 2014 07April2014
## 2 22    April 2014 22April2014
## 3 13    May   2014 13May2014
## 4 13    May   2014 13May2014
## 5 13    May   2014 13May2014
## 6 15    May   2014 15May2014
## 7 27    May   2014 27May2014
## 8 27    May   2014 27May2014
## 9 28    May   2014 28May2014
## 10 18   June 2014 18June2014
## # i 102 more rows
```

Essayons de combiner `paste()` avec d'autres fonctions de chaîne pour résoudre un problème de données réaliste. Considérez la colonne ID dans le jeu de données `hiv_dat_messy_1` :

```
hiv_dat_messy_1 %>%
  select(patient_id)
```

```
## # A tibble: 1,413 × 1
##   patient_id
##   <chr>
## 1 pd-10037
## 2 pd-10537
## 3 pd-5489
## 4 id-5523
## 5 pd-4942
## 6 pd-4742
## 7 pd-10879
## 8 id-2885
## 9 pd-4861
## 10 pd-5180
## # i 1,403 more rows
```

Imaginez que nous voulions standardiser ces ID pour qu'ils aient le même nombre de caractères. C'est souvent une exigence pour les ID (pensez aux numéros de téléphone, par exemple).

Pour cela, nous pouvons utiliser `separate()` pour diviser les ID en parties, puis utiliser `paste()` pour les recombinaison dans un format standardisé.

```
hiv_dat_messy_1 %>%
  select(patient_id) %>% # pour la visibilité
  separate(patient_id, into = c("prefix", "patient_num"), sep = "-", remove
= F) %>%
  mutate(patient_num = str_pad(patient_num, width = 5, side = "left", pad =
"0")) %>%
  mutate(patient_id_padded = paste(prefix, patient_num, sep = "-"))
```

```
## # A tibble: 1,413 × 4
##   patient_id prefix patient_num patient_id_padded
##   <chr>      <chr>   <chr>      <chr>
## 1 pd-10037   pd      10037      pd-10037
## 2 pd-10537   pd      10537      pd-10537
## 3 pd-05489   pd       05489      pd-05489
## 4 id-05523   id       05523      id-05523
## 5 pd-04942   pd       04942      pd-04942
## 6 pd-04742   pd       04742      pd-04742
## 7 pd-10879   pd      10879      pd-10879
## 8 id-02885   id       02885      id-02885
## 9 pd-04861   pd       04861      pd-04861
## 10 pd-05180  pd       05180      pd-05180
## # i 1,403 more rows
```

Dans cet exemple, `patient_id` est divisé en un préfixe et un numéro. Le numéro est ensuite complété par des zéros pour assurer une longueur cohérente, et enfin, les deux parties sont concaténées à nouveau en utilisant `paste()` avec un trait d'union comme séparateur. Ce processus standardise le format des ID des patients.

Excellent travail !

Q : Standardisation des ID dans le Jeu de Données `lima_messy_1`

PRACTICE



Dans le jeu de données `lima_messy_1`, les ID ne sont pas complétés par des zéros, ce qui les rend difficiles à trier.

Par exemple, l'ID `pe-998` est en haut de la liste après un tri par ordre décroissant, ce qui n'est pas ce que nous voulons.


```
lima_messy_1 %>%
  select(id) %>%
  arrange(desc(id)) # trier par ordre décroissant (les ID les
plus élevés devraient être en haut)
```

```
## # A tibble: 1,293 × 1
##   id
##   <chr>
## 1 pe-998
## 2 pe-996
## 3 pe-951
## 4 pe-900
## 5 pe-2347
## 6 pe-2337
## 7 pe-2335
## 8 pe-2333
## 9 pe-2331
## 10 pe-2329
## # i 1,283 more rows
```

PRACTICE



(in RMD)

Essayez de résoudre ce problème en utilisant une procédure similaire à celle utilisée pour `hiv_dat_messy_1`.

Votre Tâche :

- Séparer l'ID en parties.
- Compléter la partie numérique pour la standardisation.
- Recombiner les parties en utilisant `paste()`.
- Retrier les ID par ordre décroissant. Le plus haut ID devrait se terminer par 2347

```
lima_messy_1 %>%
  _____
```

Q : Création de déclarations récapitulatives

PRACTICE



(in RMD)

Créez une colonne contenant des déclarations récapitulatives combinant `village`, `start_date_default` et `coverage_p` de l'ensemble de données `irs`. La déclaration doit décrire la couverture de pulvérisation pour chaque village.

Sortie souhaitée : "Pour le village X, la couverture de pulvérisation était de Y % à la date Z."

PRACTICE



Votre tâche : - Sélectionnez les colonnes nécessaires de l'ensemble de données `irs`. - Utilisez `paste()` pour créer la déclaration récapitulative.

```
irs %>%  
  select(village, start_date_default, coverage_p) %>%  
  _____
```

Au fur et à mesure que nous avançons dans cette leçon, rappelez-vous que l'auto-complétion de RStudio peut vous aider à trouver des fonctions dans le package `stringr`.

REMINDER



Tapez simplement `str_` et une liste de fonctions `stringr` apparaîtra. Toutes les fonctions `stringr` commencent par `str_`.

Ainsi, au lieu d'essayer de toutes les mémoriser, vous pouvez utiliser l'auto-complétion comme référence si nécessaire.

Sous-référencement de chaînes avec `str_sub`

`str_sub` vous permet d'extraire des parties d'une chaîne de caractères en fonction des positions des caractères. La syntaxe de base est `str_sub(chaine, debut, fin)`.

Exemple : Extraction des 2 premiers caractères des identifiants de patients :

```
patient_ids <- c("ID12345-abc", "ID67890-def")  
str_sub(patient_ids, 1, 2) # Retourne "ID", "ID"
```

```
## [1] "ID" "ID"
```

Ou les 5 premiers :

```
str_sub(patient_ids, 1, 5) # Retourne "ID123", "ID678"
```

```
## [1] "ID123" "ID678"
```

Les valeurs négatives comptent à rebours depuis la fin de la chaîne. Cela est utile pour extraire des suffixes.

Par exemple, pour obtenir les 4 derniers caractères des identifiants de patients.

```
str_sub(patient_ids, -4, -1) # Retourne "-abc", "-def"
```

```
## [1] "-abc" "-def"
```

Assurez-vous de faire une pause et de comprendre ce qui s'est passé ci-dessus.

Lorsque les indices sont en dehors de la longueur de la chaîne, `str_sub` le gère avec grâce sans erreurs :

```
str_sub(patient_ids, 1, 30) # Retourne en toute sécurité la chaîne complète  
lorsque la plage dépasse la longueur de la chaîne
```

```
## [1] "ID12345-abc" "ID67890-def"
```

Dans un dataframe, nous pouvons utiliser `str_sub` dans `mutate()`. Par exemple, ci-dessous, nous extrayons l'année et le mois de la colonne `start_date_default` et créons une nouvelle colonne appelée `year_month` :

```
irs %>%  
  select(start_date_default) %>%  
  mutate(year_month = str_sub(start_date_default, start = 1, end = 7))
```

```
## # A tibble: 112 × 2  
##   start_date_default year_month  
##   <date>             <chr>  
## 1 2014-04-07         2014-04  
## 2 2014-04-22         2014-04  
## 3 2014-05-13         2014-05  
## 4 2014-05-13         2014-05  
## 5 2014-05-13         2014-05  
## 6 2014-05-15         2014-05  
## 7 2014-05-27         2014-05  
## 8 2014-05-27         2014-05  
## 9 2014-05-28         2014-05  
## 10 2014-06-18        2014-06  
## # i 102 more rows
```

Q : Extraction de sous-chaînes d'ID

PRACTICE



(in RMD)

Utilisez `str_sub()` pour isoler uniquement la partie numérique de la colonne `patient_id` dans l'ensemble de données `hiv_dat_messy_1`.

```
hiv_dat_messy_1 %>%  
  select(patient_id) %>%  
  # votre code ici :  
  _____
```

Conclusion

Félicitations pour avoir atteint la fin de cette leçon ! Vous avez appris sur les chaînes de caractères en R et diverses fonctions pour les manipuler efficacement.

Le tableau ci-dessous offre un rapide récapitulatif des fonctions clés que nous avons abordées. Rappelez-vous, vous n'avez pas besoin de mémoriser toutes ces fonctions. Savoir qu'elles existent et comment les rechercher (comme utiliser Google) est plus que suffisant pour des applications pratiques.

| Fonction | Description | Exemple | Résultat de l'exemple |
|-----------------------------|--|--|-----------------------|
| <code>str_to_upper()</code> | Convertir les caractères en majuscules | <code>str_to_upper("hiv")</code> | "HIV" |
| <code>str_to_lower()</code> | Convertir les caractères en minuscules | <code>str_to_lower("HIV")</code> | "hiv" |
| <code>str_to_title()</code> | Convertir en majuscules le premier caractère de chaque mot | <code>str_to_title("hiv awareness")</code> | "Hiv Awareness" |
| <code>str_trim()</code> | Supprimer les espaces au début et à la fin | <code>str_trim(" hiv ")</code> | "hiv" |
| <code>str_squish()</code> | Supprimer les espaces au début et à la fin et réduire les espaces internes <code>str_squish(" hiv cases ")</code> "hiv cases" | | |
| <code>str_pad()</code> | Remplir une chaîne à une largeur fixe | <code>str_pad("45", width = 5)</code> | "00045" |

| | | | |
|--------------------------|---|--|----------------------------------|
| <code>str_wrap()</code> | Formater une chaîne à une largeur donnée (pour formater la sortie) <code>str_wrap("HIV awareness", width = 5) "HIV </code> | | |
| <code>str_split()</code> | Séparer les éléments d'un vecteur de caractères | <code>str_split("Hello-World", "-")</code> | <code>c("Hello", "World")</code> |
| <code>paste()</code> | Concaténer des vecteurs après les avoir convertis en caractères <code>paste("Hello", "World") "Hello World" </code> | | |
| <code>str_sub()</code> | Extraire et remplacer des sous-chaînes d'un vecteur de caractères <code>str_sub("HelloWorld", 1, 4) "Hell" </code> | | |
| <code>separate()</code> | Séparer une colonne de caractères en plusieurs colonnes | <pre> separate(tibble(a = "Hello-World"), a, b c Hello into = c("b", World "c"), sep = "-") </pre> | |

Notez que bien que ces fonctions couvrent des tâches courantes telles que la standardisation des chaînes, la division et la jonction des chaînes, cette introduction n'effleure que la surface de ce qui est possible avec le package `{stringr}`. Si vous travaillez avec beaucoup de données textuelles brutes, vous voudrez peut-être explorer davantage sur le site web [stringr](#).

Clés de Réponses

Q : Identification d'Erreurs dans les Définitions de Chaînes

1. **ex_a**: Correct.
2. **ex_b**: Correct.
3. **ex_c**: Erreur. Version corrigée : `ex_c <- "They've been \"best friends\" for years."` 4

5. **ex_d**: Erreur. Version corrigée : `ex_d <- 'Jane\'s diary'` 5. **ex_e**: Erreur. Guillemet de fermeture manquant. Version corrigée : `ex_e <- "It's a sunny day!"`

Q : Nettoyage des Données de Noms de Patients

```
patient_names <- c("  john doe", "ANNA SMITH  ", "Emily Davis")

patient_names <- str_trim(patient_names) # Supprimer les espaces
patient_names <- str_to_title(patient_names) # Convertir en casse de titre
```

Q : Standardisation des Codes de Médicaments

```
drug_codes <- c("12345", "678", "91011")

# Remplir chaque code avec des zéros à gauche pour une largeur fixe de 8 caractères.
drug_codes_padded <- str_pad(drug_codes, 8, pad = "0")
```

Q : Formatage des Instructions Médicales

```
instructions <- "Prenez deux comprimés quotidiennement après les repas. Si
les symptômes persistent pendant plus de trois jours, consultez
immédiatement votre médecin. Ne prenez pas plus que la dose recommandée.
Tenir hors de portée des enfants."

# Formater les instructions
wrapped_instructions <- str_wrap(instructions, width = 50)

ggplot(data.frame(x = 1, y = 1), aes(x, y, label = wrapped_instructions)) +
  geom_label() +
  theme_void()
```

Q : Formatage d'un Jeu de Données sur la Tuberculose

Les étapes pour nettoyer le jeu de données lima_messy comprendraient :

```
lima_clean <- lima_messy %>%
  mutate(
    marital_status = str_squish(str_to_title(marital_status)), # Nettoyer et
    # standardiser marital_status
    sex = str_squish(str_to_upper(sex)), # Nettoyer
    # et standardiser sex
    age = str_squish(str_to_lower(age)) # Nettoyer
    # et standardiser age
  )

lima_clean %>%
```

```
select(marital_status, sex, age) %>% tbl_summary()
```

Ensuite, utilisez la fonction `tbl_summary()` pour créer le tableau récapitulatif.

Q : Formatage des Étiquettes d'Axe dans un Graphique

```
# En supposant que lima_clean est déjà créé et contient marital_status
ggplot(lima_clean, aes(x = str_wrap(marital_status, width = 15))) +
  geom_bar() +
  labs(x = "État Civil")
```

Q : Séparation des Chaînes de Plages d'Âge

```
esoph_ca %>%
  select(agegp) %>% # pour illustration
  separate(agegp, into = c("agegp_lower", "agegp_upper"), sep = "-") %>%
  mutate(agegp_lower = readr::parse_number(agegp_lower))
```

Q : Création de Déclarations Sommaires

```
irs %>%
  select(village, start_date_default, coverage_p) %>%
  mutate(summary_statement = paste0("Pour le village ", village, ", la
couverture de pulvérisation était de ", coverage_p, "% le ",
start_date_default))
```

Q : Extraction de Sous-chaînes d'ID

```
hiv_dat_messy_1 %>%
  select(patient_id) %>%
  mutate(numeric_part = str_sub(patient_id, 4))
```

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
