Manipuler du texte avec stringr

Malick SENE

2025-04-11

6. Manipuler du texte avec stringr

```
library(tidyverse)
library(tibble)
library(stringr)
library(tidyr)
```

Les fonctions de `forcats` vues précédemment permettent de modifier les modalités d'une variable qualitative **dans leur ensemble**.
Mais parfois, on a besoin de manipuler **le contenu même du texte** d'une variable de type chaîne de caractères :
combiner, rechercher, remplacer, etc.

Pour cela, on va utiliser les fonctions de l'extension **`stringr`**.
Cette extension fait partie du **coeur du `tidyverse`**, elle est donc **automatiquement chargée** avec : library(tidyverse)

```
d <- tibble(
  nom = c("M. Mamadou Diop", "Mme Awa Ndiaye", "M. Ibrahima Sarr", "Mme Fatou Bâ"),
  adresse = c("10 rue des Manguiers", "47 avenue Blaise Diagne",
              "12 rue du Souvenir Africain", "221 avenue Cheikh Anta Diop"),
  ville = c("Dakar", "Thiès", "Kaolack", "Saint-Louis")
)
```

6.1 Expressions régulières

Manipuler du texte avec `stringr` et les expressions régulières

De nombreuses fonctions de l'extension `stringr` sont conçues pour travailler avec des **expressions régulières** (ou *regular expressions* en anglais).
Les expressions régulières forment un **mini-langage spécialisé** qui permet de **décrire des motifs dans du texte** de façon très précise.

Elles sont **très puissantes** pour effectuer des opérations telles que :

- chercher un mot ou une suite de caractères,
- détecter une structure particulière (comme un numéro de téléphone ou une adresse email),
- extraire ou remplacer certaines parties d'un texte.

À quoi servent les expressions régulières ?

Voici quelques **exemples concrets** de ce que l'on peut faire avec des expressions régulières :

Objectif	Expression régulière	Description rapide
Trouver un mot en fin de phrase	`\\w+$`	Le dernier mot d'une chaîne
Détecter une majuscule en début de mot	`\\b[A-Z]\\w*`	Mot commençant par une majuscule
Identifier des nombres à 3 ou 4 chiffres au début	`^\\d{3,4}`	Nombre en début de chaîne (ex. 221, 2023…)
Détecter une adresse e-mail	`[\\w.+-]+@[\\w.-]+\\.[a-zA-Z]{2,}`	Motif générique pour les emails

Exemple concret : Détection d'e-mail avec `str_detect()`

Imaginons une base de données avec des commentaires ou des messages.
On souhaite **identifier les lignes contenant une adresse email**.

```
texte <- tibble(
  message = c(
    "Contactez-nous à info@entreprise.sn",
    "Pas d'adresse ici",
    "Email secondaire : support.tech@domaine.org"
  )
)

# Détection des adresses email
texte %>%
  mutate(contient_email = str_detect(message, "[\\w.+-]+@[\\w.-]+\\.[a-zA-Z]{2,}"))
```

```
## # A tibble: 3 x 2
##   message                                    contient_email
##   <chr>                                      <lgl>
## 1 Contactez-nous à info@entreprise.sn        TRUE
## 2 Pas d'adresse ici                          FALSE
## 3 Email secondaire : support.tech@domaine.org TRUE
```

6.2 Concaténer des chaînes

La première opération de base consiste à **concaténer des chaînes de caractères entre elles**, c'est-à-dire **les coller bout à bout** pour former une seule chaîne.
Cela se fait avec la fonction `paste()`.

Imaginons qu'on veuille créer une colonne `adresse_complete` qui combine l'adresse et la ville, à partir du tableau `d`.

```
paste(d$adresse, d$ville)
```

```
## [1] "10 rue des Manguiers Dakar"
## [2] "47 avenue Blaise Diagne Thiès"
## [3] "12 rue du Souvenir Africain Kaolack"
## [4] "221 avenue Cheikh Anta Diop Saint-Louis"
```

Par défaut, la fonction `paste()` **concatène les chaînes de caractères en insérant un espace** entre elles.

Si l'on souhaite utiliser un autre séparateur, il faut le **spécifier avec l'argument `sep`**.

```
paste(d$adresse, d$ville, sep = "- ")
```

```
## [1] "10 rue des Manguiers- Dakar"
## [2] "47 avenue Blaise Diagne- Thiès"
## [3] "12 rue du Souvenir Africain- Kaolack"
## [4] "221 avenue Cheikh Anta Diop- Saint-Louis"
```

Il existe une variante de la fonction `paste()` appelée **`paste0()`**,
qui **concatène les chaînes sans insérer de séparateur par défaut**

```
paste0(d$adresse, d$ville)
```

```
## [1] "10 rue des ManguiersDakar"
## [2] "47 avenue Blaise DiagneThiès"
## [3] "12 rue du Souvenir AfricainKaolack"
## [4] "221 avenue Cheikh Anta DiopSaint-Louis"
```

À noter : `paste()` et `paste0()` sont des fonctions de **base R**.

L'équivalent dans l'extension **`stringr`** (qui fait partie du tidyverse) s'appelle **`str_c()`** s'utulise de la même façon que **`paste()`**.

Parfois, on peut ne pas chercher à concaténer les éléments d'un vecteur **avec ceux d'un autre vecteur**, comme on l'a fait précédemment, mais plutôt à **coller tous les éléments d'un seul vecteur entre eux**.

Dans ce cas, **`paste()` seul ne fera rien de particulier** : il collera les éléments **en parallèle**, élément par élément.

```
paste(d$ville)
```

```
## [1] "Dakar"       "Thiès"       "Kaolack"     "Saint-Louis"
```

Pour concaténer les **éléments d'un vecteur entre eux**, il faut utiliser l'argument **`collapse`**.

```
paste(d$ville, collapse= ",")
```

```
## [1] "Dakar,Thiès,Kaolack,Saint-Louis"
```

ou alors :

```
str_c(d$ville, collapse= ",")
```

```
## [1] "Dakar,Thiès,Kaolack,Saint-Louis"
```

6.3 Convertir en majuscules/minuscules

Les fonctions `str_to_lower()`, `str_to_upper()` et `str_to_title()` du package `stringr` permettent de :

- `str_to_lower()` : mettre **en minuscules** toutes les lettres,
- `str_to_upper()` : mettre **en majuscules** toutes les lettres,
- `str_to_title()` : **capitaliser** chaque mot (mettre une majuscule au début de chaque mot).

```
 str_to_lower(d$nom)
```

```
## [1] "m. mamadou diop"  "mme awa ndiaye"   "m. ibrahima sarr" "mme fatou bâ"
```

```
str_to_upper(d$nom)
```

```
## [1] "M. MAMADOU DIOP"  "MME AWA NDIAYE"   "M. IBRAHIMA SARR" "MME FATOU BÂ"
```

```
str_to_title(d$nom)
```

```
## [1] "M. Mamadou Diop"  "Mme Awa Ndiaye"   "M. Ibrahima Sarr" "Mme Fatou Bâ"
```

6.4 Découper des chaînes

La fonction `str_split()` permet de **"découper" une chaîne de caractères** en fonction d'un **délimiteur**. On passe :

- la **chaîne à découper** en premier argument,
- le **délimiteur** (ou séparateur) en second.

```
 str_split("un-deux-trois","-")
```

```
## [[1]]
## [1] "un"    "deux"  "trois"
```

On peut appliquer la fonction à un vecteur, dans ce cas le résultat sera une liste:

```
str_split(d$nom, " ")
```

```
## [[1]]
## [1] "M."      "Mamadou" "Diop"
##
## [[2]]
## [1] "Mme"    "Awa"    "Ndiaye"
##
## [[3]]
## [1] "M."       "Ibrahima" "Sarr"
##
## [[4]]
## [1] "Mme"   "Fatou" "Bâ"
```

On peut aussi obtenir un **tableau (plus précisément une matrice)** si l'on ajoute l'argument `simplify = TRUE` à la fonction `str_split()`.

```r
str_split(d$nom, " ",simplify= TRUE)
```

```
##      [,1]  [,2]       [,3]
## [1,] "M."  "Mamadou"  "Diop"
## [2,] "Mme" "Awa"      "Ndiaye"
## [3,] "M."  "Ibrahima" "Sarr"
## [4,] "Mme" "Fatou"    "Bâ"
```

Si l'on souhaite **créer de nouvelles colonnes** dans un tableau de données en **découpant une colonne de type texte**,
on peut utiliser la fonction `separate()` de l'extension `tidyr`.

Cette fonction permet de **séparer une chaîne de caractères** en plusieurs variables, en **définissant un séparateur**.

Voici un exemple d'utilisation :

```r
d%>%separate(nom,c("genre","prenom","nom"))
```

```
## Warning: Expected 3 pieces. Additional pieces discarded in 1 rows [4].
```

```
## # A tibble: 4 x 5
##   genre prenom   nom    adresse                     ville
##   <chr> <chr>    <chr>  <chr>                       <chr>
## 1 M     Mamadou  Diop   10 rue des Manguiers        Dakar
## 2 Mme   Awa      Ndiaye 47 avenue Blaise Diagne     Thiès
## 3 M     Ibrahima Sarr   12 rue du Souvenir Africain Kaolack
## 4 Mme   Fatou    B      221 avenue Cheikh Anta Diop Saint-Louis
```

6.5 Extraire des sous-chaînes par position

La fonction `str_sub()` permet d'**extraire une sous-chaîne** dans une chaîne de caractères en indiquant les positions des premiers et derniers caractères.

```r
texte <- "Université Cheikh Anta Diop"

# Extraire les 10 premiers caractères
str_sub(texte, 1, 10)
```

```
## [1] "Université"
```

6.6 Détecter des motifs

La fonction `str_detect()` permet de **détecter la présence d'un motif** (texte, mot, symbole...) parmi les éléments d'un vecteur.
Elle renvoie `TRUE` si le motif est trouvé, et `FALSE` sinon.

Exemple : détecter si l'adresse contient le mot "rue"

```r
str_detect(d$adresse, "rue")
```

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

La fonction `str_detect()` renvoie un **vecteur de valeurs logiques** (`TRUE` ou `FALSE`)
et peut donc être utilisée, par exemple, avec le `filter()` de `dplyr` pour **extraire des sous-populations**.
Par exemple :

```r
# Filtrer uniquement les lignes contenant "Libération"
d %>% filter(str_detect(adresse, "rue"))
```

```
## # A tibble: 2 x 3
##   nom              adresse                     ville
##   <chr>            <chr>                       <chr>
## 1 M. Mamadou Diop  10 rue des Manguiers        Dakar
## 2 M. Ibrahima Sarr 12 rue du Souvenir Africain Kaolack
```

La fonction str_count() permet de compter le nombre d'occurrences d'un motif dans chaque élément d'un vecteur.

```r
str_count(d$adresse, "rue")
```

```
## [1] 1 0 1 0
```

On peut aussi utiliser la fonction `str_subset()` pour **ne garder d'un vecteur que les éléments qui correspondent à un motif**.

```r
str_subset(d$adresse, "rue")
```

```
## [1] "10 rue des Manguiers"        "12 rue du Souvenir Africain"
```

6.7 Extraire des motifs

La fonction `str_extract()` permet d'**extraire les valeurs qui correspondent à un motif donné**.

Si l'on lui passe simplement une chaîne fixe (comme `"Dakar"`), cela aura **peu d'intérêt**,
car cela revient à détecter un mot exact.

Mais l'intérêt de `str_extract()` se manifeste lorsqu'on l'utilise avec **des expressions régulières**,
car elle permet alors d'**extraire dynamiquement** des portions spécifiques d'une chaîne (comme un mot, un chiffre, un code, etc.).

Par exemple récupérer le premier numéro de l'adresse.

```r
str_extract(d$adresse,"^\\d+")
```

```
## [1] "10"  "47"  "12"  "221"
```

La fonction `str_extract()` ne récupère **que la première occurrence** du motif spécifié.
Si l'on souhaite **extraire toutes les correspondances** d'un motif dans une chaîne, il faut utiliser la fonction `str_extract_all()`.

6.8 Remplacer des motifs

La fonction `str_replace()` permet de **remplacer une chaîne ou un motif spécifique** dans un texte par une **autre valeur** (mot, expression, symbole, etc.).

Par exemple, on peut remplace les occurrences de “Mr” par “M.” dans les noms de notre tableau :

```
str_replace(d$nom, "Mr", "M.")
```

```
## [1] "M. Mamadou Diop"  "Mme Awa Ndiaye"   "M. Ibrahima Sarr" "Mme Fatou Bâ"
```

La fonction `str_replace_all()` est une variante de `str_replace()`
qui permet de **spécifier plusieurs remplacements à la fois** à l’aide d’une **liste**.

```
str_replace_all(d$adresse, c("avenue"="Avenue",  "rue"="Rue"))
```

```
## [1] "10 Rue des Manguiers"        "47 Avenue Blaise Diagne"
## [3] "12 Rue du Souvenir Africain" "221 Avenue Cheikh Anta Diop"
```

6.9 Modificateurs de motifs

Par défaut, les motifs passés aux fonctions comme `str_detect()`, `str_extract()` ou `str_replace()` sont interprétés comme des **expressions régulières**.

Or certains caractères ont une **signification spéciale** en expression régulière :
par exemple, le point `.` signifie **“n’importe quel caractère”**,
et non un point littéral (comme dans une abréviation : “M.”).

```
str_count(d$nom, ".")
```

```
## [1] 15 14 16 12
```

Il faut donc **spécifier que notre point est bien un caractère littéral** (et non un symbole d’expression régulière)
en l’entourant de la fonction `fixed()`.

```
str_count(d$nom, fixed("."))
```

```
## [1] 1 0 1 0
```

On peut également modifier le **comportement des expressions régulières** en utilisant la fonction `regex()`.
Cela permet, par exemple, de rendre une recherche **insensible à la casse** (majuscule/minuscule)
grâce à l’argument `ignore_case = TRUE`.

```
str_detect(d$nom, "mme")
```

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

```r
str_detect(d$nom, regex("mme", ignore_case = TRUE))
```

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