R avancé

Manipulation de données

Nicolas Klutchnikoff

January 2023

Introduction

Présentation

Le `tidyverse` est une collection de packages partageant les mêmes principes de fonctionnement et une syntaxe largement commune. L'objet central avec lequel travaillent les packages du `tidyverse` sont les **tibbles**.

Principaux packages du `tidyverse` :

- `tibble` : définition, création, etc. des tibbles ;
- `dplyr` : manipulation avancée des tibbles ;
- `readr` : lecture de fichiers plats ;
- `ggplot2` : pour faire de beaux graphiques ;
- `tidyr` : réorganisation des données.

Il en existe d'autres (`forcats`, `stringr` et `purrr`) et certains packages, sans faire partie du `tidyverse` sont compatibles avec son fonctionnement. Citons `readxl`, `DBI`, `jsonlite` ou encore `rvest`.

Chargement du meta-package

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   1.0.0
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Quelques rappels

Classes d'un objet

Les data-frames sont des **objets** au sens de R : ils possèdent une classe.

```
df <- data.frame(x = 1:10, y = 11:20)
class(df)
```

```
## [1] "data.frame"
```

La fonction `print()` adapte son affichage à l'objet qu'elle reçoit en fonction de sa classe.

```
print(df)
```

```
##     x  y
## 1   1 11
## 2   2 12
## 3   3 13
## 4   4 14
## 5   5 15
## 6   6 16
## 7   7 17
## 8   8 18
## 9   9 19
## 10 10 20
```

Fonctions génériques

C’est le principe des fonctions **génériques** :

```
print.default(df)
```

```
## $x
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $y
##  [1] 11 12 13 14 15 16 17 18 19 20
##
## attr(,"class")
## [1] "data.frame"
```

Les tibbles

Si un objet possède plusieurs classes (c'est possible), celles-ci sont ordonnées

```
class(as_tibble(df))
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
print(as_tibble(df))
```

```
## # A tibble: 10 x 2
##        x     y
##    <int> <int>
##  1     1    11
##  2     2    12
##  3     3    13
##  4     4    14
##  5     5    15
##  6     6    16
##  7     7    17
##  8     8    18
##  9     9    19
## 10    10    20
```

Principes

Lors de l'appel de la fonction `print()` :

- Si elle existe, on appelle la **méthode** `print.tbl_df()`;
- Sinon, on appelle la **méthode** `print.tbl()`;
- Sinon, on appelle la **méthode** `print.data.frame()`;
- En dernier recours, on appelle la **méthode** `print.default()`.

Lecture des données

Les fonctions de lecture

Le `tidyverse` propose ses propres fonctions d'importation des données via l'utilisation du package `readr` :

- `read_csv()` pour lire les données au format comma separated values ;
- `read_tsv()` pour les données au format tabulation separated values ;
- `read_delim()` dont les deux fonctions précédentes sont des cas particuliers.

Ces fonctions retournent des tibbles.

Focus sur `read_csv()` (1)

```
tbl <- read_csv("data/piscines.csv")
```

```
## Rows: 20 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): Name, Address
## dbl (2): Latitude, Longitude
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message
```

Un message d'alerte nous informe que quelque chose s'est passé. La fonction a reconnu que les variables `Name` et `Address` sont des chaînes de caratères tandis que `Latitude` et `Longitude` sont des doubles.

Pas de magie ici : les premières lignes sont lues pour deviner le type de chaque colonne. En cas d'erreur il faudra les définir manuellement!

Focus sur `read_csv()` (2)

```
class(tbl)
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

```
head(tbl, 5) # head() est une fonction générique
```

```
## # A tibble: 5 x 4
##   Name                        Address                              Latit~1 Longi~2
##   <chr>                       <chr>                                  <dbl>   <dbl>
## 1 Acacia Ridge Leisure Centre 1391 Beaudesert Road, Acacia Ridge     -27.6    153.
## 2 Bellbowrie Pool             Sugarwood Street, Bellbowrie           -27.6    153.
## 3 Carole Park                 Cnr Boundary Road and Waterford R~     -27.6    153.
## 4 Centenary Pool (inner City) 400 Gregory Terrace, Spring Hill       -27.5    153.
## 5 Chermside Pool              375 Hamilton Road, Chermside           -27.4    153.
## # ... with abbreviated variable names 1: Latitude, 2: Longitude
```

Manipulations élémentaires

Principes

Le package `dplyr` propose une grammaire de manipulation des données.

Cette grammaire est centrée autour de verbes. Il en existe principalement 5, même si ce n'est pas complètement vrai :

- `filter()` : permet de filtrer des individus ;
- `select()` : permet de sélectionner des variables ;
- `mutate()` : permet l'ajout de variables à un tibble ;
- `arrange()` permet permuter les individus pour présenter un tibble d'une manière différente.
- `summarise()` permet d'extraire des informations contenues dans un tibble.

Filtrer des individus (1)

On utilise essentiellement le verbe `filter()` dont la syntaxe générale est donnée par :

```
filter(tbl, TEST)
```

- `tbl` est un tibble ;
- `TEST` est un vecteur de booléens.

Une ligne est retenue dans le tibble créé si `TEST` vaut `TRUE` pour cette ligne.

```
p1 <- filter(tbl, Longitude>153.02)
p1$Longitude
```

```
##  [1] 153.0264 153.0251 153.0351 153.0789 153.0215 153.0368 153.0735 153.1874
##  [9] 153.0943 153.0764 153.0691 153.0487
```

Filtrer des individus (2)

On peut donner un exemple un peu plus complexe en utilisant des **expressions régulières** pour filtrer les piscines dont le nom ne contient pas “Pool” :

```
p2 <- filter(tbl, str_detect(Name, "Pool"))
p2$Name
```

```
##  [1] "Bellbowrie Pool"
##  [2] "Centenary Pool (inner City)"
##  [3] "Chermside Pool"
##  [4] "Colmslie Pool (Morningside)"
##  [5] "Dunlop Park Pool (Corinda)"
##  [6] "Fortitude Valley Pool"
##  [7] "Ithaca Pool ( Paddington)"
##  [8] "Jindalee Pool"
##  [9] "Manly Pool"
## [10] "Musgrave Park Pool (South Brisbane)"
## [11] "Newmarket Pool"
## [12] "Runcorn Pool"
## [13] "Sandgate Pool"
## [14] "Langlands Parks Pool (Stones Corner)"
## [15] "Yeronga Park Pool"
```

Filtrer des individus (3)

On peut combiner les test avec :

- & : pour **et**;
- | : pour **ou**.

```
p3 <- filter(tbl, Longitude > 153.02 | Latitude < -27.488)
p3 <- select(p3, Longitude, Latitude)
```

```
head(p3, 10)
```

```
## # A tibble: 10 x 2
##    Longitude Latitude
##        <dbl>    <dbl>
##  1      153.    -27.6
##  2      153.    -27.6
##  3      153.    -27.6
##  4      153.    -27.5
##  5      153.    -27.4
##  6      153.    -27.5
##  7      153.    -27.5
##  8      153.    -27.5
##  9      153.    -27.5
```

Filtrer des individus (4)

Le verbe `slice()` permet de filtrer des individus à l'aide du numéro de ligne. On passe un vecteur d'indices :

```
slice(tbl, c(2, 3, 5, 7, 11, 13))
```

```
## # A tibble: 6 x 4
##   Name                          Address                          Latit~1 Longi~2
##   <chr>                         <chr>                              <dbl>   <dbl>
## 1 Bellbowrie Pool               Sugarwood Street, Bellbowrie       -27.6    153.
## 2 Carole Park                   Cnr Boundary Road and Waterfor~    -27.6    153.
## 3 Chermside Pool                375 Hamilton Road, Chermside       -27.4    153.
## 4 Spring Hill Baths (inner City) 14 Torrington Street, Springhi~   -27.5    153.
## 5 Ithaca Pool ( Paddington)     131 Caxton Street, Paddington      -27.5    153.
## 6 Manly Pool                    1 Fairlead Crescent, Manly         -27.5    153.
## # ... with abbreviated variable names 1: Latitude, 2: Longitude
```

Sélectionner des variables (1)

On utilise le verbe `select()` dont la syntaxe générale est donnée par :

```
select(tbl, VAR1, VAR2, ...)
```

- `tbl` est un tibble ;
- `VAR1`, `VAR2`, etc. sont les variables à sélectionner.

Note : comme pour les verbes `filter()` et `slice()` :

- le premier élément est un tibble ;
- la valeur de retour est également un tibble ;
- Le tibble `tbl` n'est pas modifié par l'opération.

Sélectionner des variables (2)

```
coord1 <- select(tbl, Latitude, Longitude)
head(tbl, 2)
```

```
## # A tibble: 2 x 4
##   Name                        Address                             Latit~1 Longi~2
##   <chr>                       <chr>                                 <dbl>   <dbl>
## 1 Acacia Ridge Leisure Centre 1391 Beaudesert Road, Acacia Ridge   -27.6    153.
## 2 Bellbowrie Pool             Sugarwood Street, Bellbowrie         -27.6    153.
## # ... with abbreviated variable names 1: Latitude, 2: Longitude
```

```
head(coord1, 2)
```

```
## # A tibble: 2 x 2
##   Latitude Longitude
##      <dbl>     <dbl>
## 1    -27.6      153.
## 2    -27.6      153.
```

On remarque que le nom des variables est écrit directement :

- pas de quotes `" "` ;
- inutile d'écrire `tbl$Longitude` non plus.

Sélectionner des variables (3)

On peut utiliser des **helper functions** définies dans `dplyr` pour construire des sélections plus complexes basées sur le nom des variables. Par exemple `starts_with()`ou `ends_with()` permet de sélectionner des variables dont le nom début ou finit par une chaîne de caractère donnée.

```
coord2 <- select(tbl, ends_with("tude"))
head(coord2, 2)
```

```
## # A tibble: 2 x 2
##   Latitude Longitude
##      <dbl>     <dbl>
## 1    -27.6      153.
## 2    -27.6      153.
```

Sélectionner des variables (4)

Voici deux **helper functions** bien utiles :

- `contains()` : contient la chaîne de cartère passée en paramètre.
- `matches()` : correspond exactement à la chaîne de caractère passée en paramètre (éventuellement une expression régulière).

```
coord3 <- select(tbl, matches("L.*tude"))
head(coord3, 2)
```

```
## # A tibble: 2 x 2
##   Latitude Longitude
##      <dbl>     <dbl>
## 1    -27.6      153.
## 2    -27.6      153.
```

Créer des variables (1)

On utilise le verbe `mutate()` dont la syntaxe générale est donnée par :

```
mutate(tbl, NEW.VAR = expression(VAR1, VAR2, ...))
```

- `tbl` est un tibble ;
- `NEW.VAR` est le nom de la nouvelle variable ;
- `expression(VAR1, VAR2, ...)` est une expression faisant intervenir certaines variables de `tbl`.

Ici encore les variables sont utilisées directement (pas de chaînes de caractères ni de `tbl$`).

Créer des variables (2)

```
mutate(tbl, phrase=paste("La piscine", Name, "est située à l'adresse", Address))
```

```
## # A tibble: 20 x 5
##    Name                                      Address      Latit~1 Longi~2 phrase
##    <chr>                                     <chr>          <dbl>   <dbl> <chr>
##  1 Acacia Ridge Leisure Centre               1391 Beaude~   -27.6    153. La pi~
##  2 Bellbowrie Pool                           Sugarwood S~   -27.6    153. La pi~
##  3 Carole Park                               Cnr Boundar~   -27.6    153. La pi~
##  4 Centenary Pool (inner City)               400 Gregory~   -27.5    153. La pi~
##  5 Chermside Pool                            375 Hamilto~   -27.4    153. La pi~
##  6 Colmslie Pool (Morningside)               400 Lytton ~   -27.5    153. La pi~
##  7 Spring Hill Baths (inner City)            14 Torringt~   -27.5    153. La pi~
##  8 Dunlop Park Pool (Corinda)                794 Oxley R~   -27.5    153. La pi~
##  9 Fortitude Valley Pool                     432 Wickham~   -27.5    153. La pi~
## 10 Hibiscus Sports Complex (upper MtGravatt) 90 Klumpp R~   -27.6    153. La pi~
## 11 Ithaca Pool ( Paddington)                 131 Caxton ~   -27.5    153. La pi~
## 12 Jindalee Pool                             11 Yallambe~   -27.5    153. La pi~
## 13 Manly Pool                                1 Fairlead ~   -27.5    153. La pi~
## 14 Mt Gravatt East Aquatic Centre            Cnr wecker ~   -27.5    153. La pi~
## 15 Musgrave Park Pool (South Brisbane)       100 Edmonst~   -27.5    153. La pi~
## 16 Newmarket Pool                            71 Alderson~   -27.4    153. La pi~
## 17 Runcorn Pool                              37 Bonemill~   -27.6    153. La pi~
## 18 Sandgate Pool                             231 Flinder~   -27.3    153. La pi~
## 19 Langlands Parks Pool (Stones Corner)      5 Panitya S~   -27.5    153. La pi~
## 20 Yeronga Park Pool                         81 School R~   -27.5    153. La pi~
## # ... with abbreviated variable names 1: Latitude, 2: Longitude
```

Créer des variables (3)

On peut ajouter plusieurs variables en même temps :

```
mutate(tbl,
       phrase = paste("La piscine", Name, "est située à l'adresse", Address),
       inutile = Longitude + Latitude
)
```

```
## # A tibble: 20 x 6
##    Name                                  Address Latit~1 Longi~2 phrase inutile
##    <chr>                                 <chr>     <dbl>   <dbl> <chr>    <dbl>
##  1 Acacia Ridge Leisure Centre           1391 B~   -27.6    153. La pi~    125.
##  2 Bellbowrie Pool                       Sugarw~   -27.6    153. La pi~    125.
##  3 Carole Park                           Cnr Bo~   -27.6    153. La pi~    125.
##  4 Centenary Pool (inner City)           400 Gr~   -27.5    153. La pi~    126.
##  5 Chermside Pool                        375 Ha~   -27.4    153. La pi~    126.
##  6 Colmslie Pool (Morningside)           400 Ly~   -27.5    153. La pi~    126.
##  7 Spring Hill Baths (inner City)        14 Tor~   -27.5    153. La pi~    126.
##  8 Dunlop Park Pool (Corinda)            794 Ox~   -27.5    153. La pi~    125.
##  9 Fortitude Valley Pool                 432 Wi~   -27.5    153. La pi~    126.
## 10 Hibiscus Sports Complex (upper MtGrav~ 90 Klu~   -27.6    153. La pi~    126.
## 11 Ithaca Pool ( Paddington)             131 Ca~   -27.5    153. La pi~    126.
## 12 Jindalee Pool                         11 Yal~   -27.5    153. La pi~    125.
## 13 Manly Pool                            1 Fair~   -27.5    153. La pi~    126.
## 14 Mt Gravatt East Aquatic Centre        Cnr we~   -27.5    153. La pi~    126.
## 15 Musgrave Park Pool (South Brisbane)   100 Ed~   -27.5    153. La pi~    126.
## 16 Newmarket Pool                        71 Ald~   -27.4    153. La pi~    126.
## 17 Runcorn Pool                          37 Bon~   -27.6    153. La pi~    125.
## 18 Sandgate Pool                         231 Fl~   -27.3    153. La pi~    126.
## 19 Langlands Parks Pool (Stones Corner)  5 Pani~   -27.5    153. La pi~    126.
## 20 Yeronga Park Pool                     81 Sch~   -27.5    153. La pi~    125.
```

Créer des variables (4)

Les variables créent peuvent dépendre les unes des autres :

```
tbl_temp <- mutate(tbl,
       phrase1 = paste("La piscine", Name, "est située à l'adresse", Address),
       phrase2 = paste("Attention !", phrase1)
)
tbl_temp$phrase2
```

```
##  [1] "Attention ! La piscine Acacia Ridge Leisure Centre est située à l'adresse 1391 Beaudesert Road, Acacia
##  [2] "Attention ! La piscine Bellbowrie Pool est située à l'adresse Sugarwood Street, Bellbowrie"
##  [3] "Attention ! La piscine Carole Park est située à l'adresse Cnr Boundary Road and Waterford Road Wacol"
##  [4] "Attention ! La piscine Centenary Pool (inner City) est située à l'adresse 400 Gregory Terrace, Spring
##  [5] "Attention ! La piscine Chermside Pool est située à l'adresse 375 Hamilton Road, Chermside"
##  [6] "Attention ! La piscine Colmslie Pool (Morningside) est située à l'adresse 400 Lytton Road, Morningside
##  [7] "Attention ! La piscine Spring Hill Baths (inner City) est située à l'adresse 14 Torrington Street, Spr
##  [8] "Attention ! La piscine Dunlop Park Pool (Corinda) est située à l'adresse 794 Oxley Road, Corinda"
##  [9] "Attention ! La piscine Fortitude Valley Pool est située à l'adresse 432 Wickham Street, Fortitude Vall
## [10] "Attention ! La piscine Hibiscus Sports Complex (upper MtGravatt) est située à l'adresse 90 Klumpp Road
## [11] "Attention ! La piscine Ithaca Pool ( Paddington) est située à l'adresse 131 Caxton Street, Paddington"
## [12] "Attention ! La piscine Jindalee Pool est située à l'adresse 11 Yallambee Road, Jindalee"
## [13] "Attention ! La piscine Manly Pool est située à l'adresse 1 Fairlead Crescent, Manly"
## [14] "Attention ! La piscine Mt Gravatt East Aquatic Centre est située à l'adresse Cnr wecker Road and Newnh
## [15] "Attention ! La piscine Musgrave Park Pool (South Brisbane) est située à l'adresse 100 Edmonstone Stree
## [16] "Attention ! La piscine Newmarket Pool est située à l'adresse 71 Alderson Stret, Newmarket"
## [17] "Attention ! La piscine Runcorn Pool est située à l'adresse 37 Bonemill Road, Runcorn"
## [18] "Attention ! La piscine Sandgate Pool est située à l'adresse 231 Flinders Parade, Sandgate"
## [19] "Attention ! La piscine Langlands Parks Pool (Stones Corner) est située à l'adresse 5 Panitya Street, S
## [20] "Attention ! La piscine Yeronga Park Pool est située à l'adresse 81 School Road, Yeronga"
```

Créer des variables (5)

Attention toutefois de veiller à l'ordre d'apparition des variables : le code suivant produit une erreur !

```
mutate(tbl,
       phrase2 = paste("Attention !", phrase1),
       phrase1 = paste("La piscine", Name, "est située à l'adresse", Address)
)
```

Réordonner un tibble (1)

On utilise le verbe `arrange()`.

```
head(arrange(tbl, Longitude), 2)
```

```
## # A tibble: 2 x 4
##   Name            Address                                  Latitude Longitude
##   <chr>           <chr>                                       <dbl>     <dbl>
## 1 Bellbowrie Pool Sugarwood Street, Bellbowrie                -27.6      153.
## 2 Carole Park     Cnr Boundary Road and Waterford Road Wacol  -27.6      153.
```

```
head(arrange(tbl, desc(Longitude)), 2)
```

```
## # A tibble: 2 x 4
##   Name                          Address                          Latit~1 Longi~2
##   <chr>                         <chr>                              <dbl>   <dbl>
## 1 Manly Pool                    1 Fairlead Crescent, Manly         -27.5    153.
## 2 Mt Gravatt East Aquatic Centre Cnr wecker Road and Newnham Ro~   -27.5    153.
## # ... with abbreviated variable names 1: Latitude, 2: Longitude
```

Réordonner un tibble (2)

Il arrive parfois qu'on se retrouve avec des ex-aequo. Une situation classique est la suivante :

```
lucky_luke <- tribble(
  ~nom, ~prenom, ~taille,
  "Dalton", "Joe", 1.40,
  "Dalton", "William", 1.67,
  "Dalton", "Jack", 1.93,
  "Lincoln", "Abraham", 1.93,
  "Dalton", "Averell", 2.13
)
```

Noter au passage l'utilisation de `tribble()`.

```
arrange(lucky_luke, nom, prenom)
```

```
## # A tibble: 5 x 3
##   nom     prenom  taille
##   <chr>   <chr>    <dbl>
## 1 Dalton  Averell   2.13
## 2 Dalton  Jack      1.93
## 3 Dalton  Joe       1.4
## 4 Dalton  William   1.67
## 5 Lincoln Abraham   1.93
```

Résumer un tibble (1)

À l'aide du verbe `summarise()` dont `summarize()` est un alias.

Il permet de construire de nouveaux tibbles qui contiennent des statistiques sur le tibble originel en appliquant des **fonctions d'agrégat** (associent un nombre à un vecteur) :

1. `mean()`;
2. `median()`;
3. `IQR()`;
4. `var()`.

Résumer un tibble (2)

```
summarise(tbl,
          mean_long = mean(Longitude),
          med_lat = median(Latitude),
          min_lat = min(Latitude),
          sum_long = sum(Longitude)
)
```

```
## # A tibble: 1 x 4
##   mean_long med_lat min_lat sum_long
##       <dbl>   <dbl>   <dbl>    <dbl>
## 1      153.   -27.5   -27.6    3061.
```

On applique plusieurs fonctions d'agrégat sur diverses colonnes du tibble.

Le tibble final contient une seule ligne.

Résumer un tibble (3)

Le package `dplyr` définit certaines nouvelles fonctions à utiliser avec `summarise()` :

1. `n()` : permet de calculer le nombre de lignes du tibble.
2. `n_distinct()` : nombre d'éléments distincts dans un vecteur : version efficiente de `length(unique())`.
3. `fisrt()` et `last()` retournent respectivement le premier élément et le dernier élément d'un vecteur.

Résumer un tibble (4)

```
summarise(lucky_luke,
          n = n(),
          n_nom = n_distinct(nom),
          n_prenom = n_distinct(prenom),
          size_of_joe = first(taille)
)
```

```
## # A tibble: 1 x 4
##       n n_nom n_prenom size_of_joe
##   <int> <int>    <int>       <dbl>
## 1     5     2        5         1.4
```

Chaînage des verbes (1)

Pour tous les verbes :

- Le premier argument est un tibble ;
- La valeur retournée est aussi un tibble.

```
p1 <- filter(tbl, !grepl("Pool", Name))
p2 <- summarise(p1,
                min_lat = min(Latitude),
                max_lat = min(Latitude),
                min_lon = min(Longitude),
                max_lon = min(Longitude)
)
p2
```

```
## # A tibble: 1 x 4
##   min_lat max_lat min_lon max_lon
##     <dbl>   <dbl>   <dbl>   <dbl>
## 1   -27.6   -27.6    153.    153.
```

Chaînage des verbes (2)

Pour régler ce problème, il faut **composer** (au sens mathématique) les opérations.

```
summarise(
  filter(tbl, !grepl("Pool", Name)),
  min_lat = min(Latitude),
  max_lat = min(Latitude),
  min_lon = min(Longitude),
  max_lon = min(Longitude)
)
```

```
## # A tibble: 1 x 4
##   min_lat max_lat min_lon max_lon
##     <dbl>   <dbl>   <dbl>   <dbl>
## 1   -27.6   -27.6    153.    153.
```

Chaînage des verbes (3)

```
tbl %>%
  filter(!grepl("Pool", Name)) %>%
  summarise(min_lat = min(Latitude),
            max_lat = min(Latitude),
            min_lon = min(Longitude),
            max_lon = min(Longitude))
```

```
## # A tibble: 1 x 4
##   min_lat max_lat min_lon max_lon
##     <dbl>   <dbl>   <dbl>   <dbl>
## 1   -27.6   -27.6    153.    153.
```

Fonctionnement basé sur l'équivalence entre :

- `f(x, ...)`
- `x %>% f(...)`.

Chaînage des verbes (4)

Intérêt du chaînage à l'aide de l'opérateur **pipe %>%** :

- lisibilité du code ;
- ajout/supression d'une instruction au sein du flux.

Manipulations avancées

Les données utilisées (1)

```
library(hflights)
hflights <- as_tibble(hflights)
hflights %>% glimpse()
```

```
## Rows: 227,496
## Columns: 21
## $ Year              <int> 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011~
## $ Month             <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ DayofMonth        <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1~
## $ DayOfWeek         <int> 6, 7, 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7, 1, 2~
## $ DepTime           <int> 1400, 1401, 1352, 1403, 1405, 1359, 1359, 1355, 1443~
## $ ArrTime           <int> 1500, 1501, 1502, 1513, 1507, 1503, 1509, 1454, 1554~
## $ UniqueCarrier     <chr> "AA", "AA", "AA", "AA", "AA", "AA", "AA", "AA", "AA"~
## $ FlightNum         <int> 428, 428, 428, 428, 428, 428, 428, 428, 428, 428, 42~
## $ TailNum           <chr> "N576AA", "N557AA", "N541AA", "N403AA", "N492AA", "N~
## $ ActualElapsedTime <int> 60, 60, 70, 70, 62, 64, 70, 59, 71, 70, 70, 56, 63, ~
## $ AirTime           <int> 40, 45, 48, 39, 44, 45, 43, 40, 41, 45, 42, 41, 44, ~
## $ ArrDelay          <int> -10, -9, -8, 3, -3, -7, -1, -16, 44, 43, 29, 5, -9, ~
## $ DepDelay          <int> 0, 1, -8, 3, 5, -1, -1, -5, 43, 43, 29, 19, -2, -3, ~
## $ Origin            <chr> "IAH", "IAH", "IAH", "IAH", "IAH", "IAH", "IAH", "IA~
## $ Dest              <chr> "DFW", "DFW", "DFW", "DFW", "DFW", "DFW", "DFW", "DF~
## $ Distance          <int> 224, 224, 224, 224, 224, 224, 224, 224, 224, 224, 22~
## $ TaxiIn            <int> 7, 6, 5, 9, 9, 6, 12, 7, 8, 6, 8, 4, 6, 5, 6, 12, 8,~
## $ TaxiOut           <int> 13, 9, 17, 22, 9, 13, 15, 12, 22, 19, 20, 11, 13, 15~
## $ Cancelled         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ CancellationCode  <chr> "", "", "", "", "", "", "", "", "", "", "", "", "", ~
## $ Diverted          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
```

Les données utilisées (2)

```
HF <- hflights %>%
  mutate(UniqueCarrier=factor(UniqueCarrier)) %>%
  mutate(UniqueCarrier=fct_recode(UniqueCarrier,
           "American"          = "AA"   ,
           "Alaska"            = "AS"   ,
           "JetBlue"           = "B6"   ,
           "Continental"       = "CO"   ,
           "Delta"             = "DL"   ,
           "SkyWest"           = "OO"   ,
           "United"            = "UA"   ,
           "US_Airways"        = "US"   ,
           "Southwest"         = "WN"   ,
           "Atlantic_Southeast"= "EV"   ,
           "Frontier"          = "F9"   ,
           "AirTran"           = "FL"   ,
           "American_Eagle"    = "MQ"   ,
           "ExpressJet"        = "XE"   ,
           "Mesa"              = "YV"))
```

Les données utilisées (3)

```
HF <- HF %>%
  mutate(CancellationCode = factor(CancellationCode)) %>%
  mutate(CancellationCode = fct_recode(CancellationCode,
         "carrier"            = "A",
         "weather"            = "B",
         "national air system"= "C",
         "security"           = "D",
         "not cancelled"      = "")) %>%
  select(UniqueCarrier, CancellationCode, DepDelay)
HF %>% glimpse()
```

```
## Rows: 227,496
## Columns: 3
## $ UniqueCarrier    <fct> American, American, American, American, American, Ame~
## $ CancellationCode <fct> not cancelled, not cancelled, not cancelled, not canc~
## $ DepDelay         <int> 0, 1, -8, 3, 5, -1, -1, -5, 43, 43, 29, 19, -
2, -3, -~
```

Groupement de données (1)

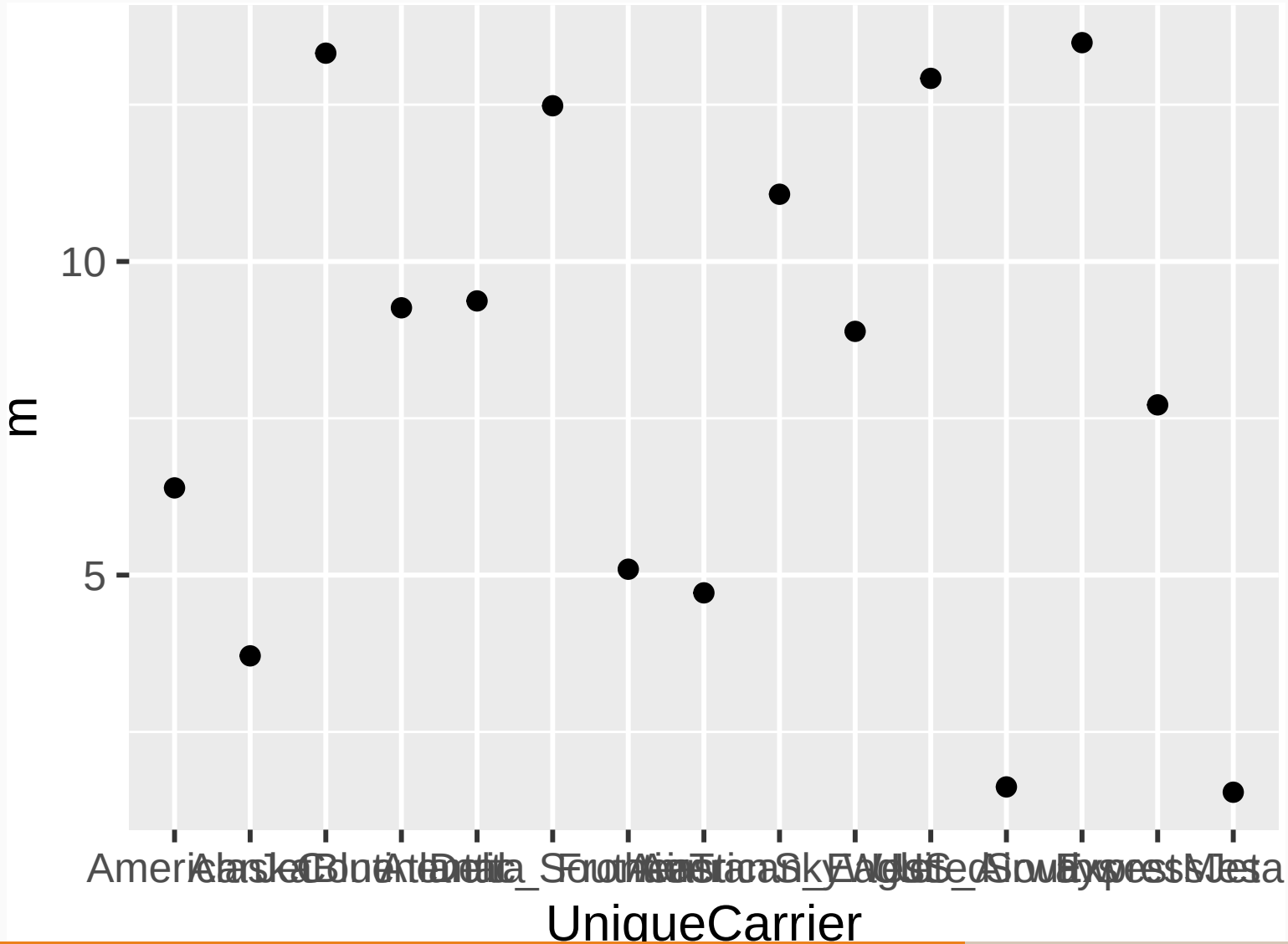
Les données peuvent être groupées. C'est très utile avec `summarise()`.

```
HF %>%
  group_by(CancellationCode) %>%
  summarise(n = n(),
            m = mean(DepDelay, na.rm = TRUE),
            sd = sd(DepDelay, na.rm = TRUE))
```

```
## # A tibble: 5 x 4
##   CancellationCode          n      m    sd
##   <fct>                 <int>  <dbl> <dbl>
## 1 not cancelled        224523   9.43  28.8
## 2 carrier                1202  45.2  106.
## 3 weather                1652  53.6   58.1
## 4 national air system     118  53.5   41.7
## 5 security                  1 NaN     NA
```

Groupement de données (2)

```
HF %>%
  group_by(UniqueCarrier) %>%
  summarise(m = mean(DepDelay, na.rm = TRUE), .groups='drop') %>%
  ggplot() + aes(x = UniqueCarrier, y = m) + geom_point()
```



Groupement de données (3)

```
HF %>%
  filter(is.finite(DepDelay) & DepDelay>0) %>% # ajouté pour supprimer des incohérences
  group_by(CancellationCode, UniqueCarrier) %>%
  summarise(n = n(),
            min = min(DepDelay, na.rm = TRUE),
            max = max(DepDelay, na.rm = TRUE),
            med = median(DepDelay, na.rm = TRUE)) %>%
  filter(CancellationCode != "not cancelled") %>%
  arrange(-n) %>%
  head(2)
```

```
## `summarise()` has grouped output by 'CancellationCode'. You can override using
## the `.groups` argument.

## # A tibble: 2 x 6
## # Groups:   CancellationCode [2]
##   CancellationCode UniqueCarrier     n   min   max   med
##   <fct>            <fct>         <int> <int> <int> <dbl>
## 1 weather          ExpressJet       13     1   173    64
## 2 carrier          ExpressJet        4     5   271    91
```

Différents formats de données

Certaines analyses statistiques nécessitent un format particulier pour les données. Construisons un exemple jouet :

```
df <- iris %>%
  group_by(Species) %>%
  summarise_all(list(mean))
head(df)
```

```
## # A tibble: 3 x 5
##   Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <fct>             <dbl>       <dbl>        <dbl>       <dbl>
## 1 setosa             5.01        3.43         1.46       0.246
## 2 versicolor         5.94        2.77         4.26       1.33
## 3 virginica          6.59        2.97         5.55       2.03
```

Cette représenttion est parfois appelée représentation compacte

Passer au format long avec `pivot_longer()`

```r
df1 <- df %>% pivot_longer(names_to = "variable", values_to = "value", -Species)
head(df1)
```

```
## # A tibble: 6 x 3
##   Species    variable     value
##   <fct>      <chr>        <dbl>
## 1 setosa     Sepal.Length 5.01
## 2 setosa     Sepal.Width  3.43
## 3 setosa     Petal.Length 1.46
## 4 setosa     Petal.Width  0.246
## 5 versicolor Sepal.Length 5.94
## 6 versicolor Sepal.Width  2.77
```

Passer au format compact avec `pivot_wider()`

```
df1 %>%
  pivot_wider(names_from = variable, values_from = value) %>%
  head()
```

```
## # A tibble: 3 x 5
##   Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <fct>             <dbl>       <dbl>        <dbl>       <dbl>
## 1 setosa             5.01        3.43         1.46       0.246
## 2 versicolor         5.94        2.77         4.26       1.33
## 3 virginica          6.59        2.97         5.55       2.03
```

Remarque : on dispose des mêmes informations présentées dans deux formats différents.

Scinder et regrouper des colonnes (1)

```
df <- tibble(
  date=as.Date(c("01/03/2015","05/18/2017", "09/14/2018"),
               "%m/%d/%Y"),
  temp=c(18, 21, 15))
head(df)
```

```
## # A tibble: 3 x 2
##   date        temp
##   <date>     <dbl>
## 1 2015-01-03    18
## 2 2017-05-18    21
## 3 2018-09-14    15
```

```
df1 <- df %>% separate(date, into = c("year", "month", "day"))
df1
```

```
## # A tibble: 3 x 4
##   year  month day    temp
##   <chr> <chr> <chr> <dbl>
## 1 2015  01    03       18
## 2 2017  05    18       21
## 3 2018  09    14       15
```

Scinder et regrouper des colonnes (2)

```
df1 %>% unite(date, year, month, day, sep="/")
```

```
## # A tibble: 3 x 2
##   date        temp
##   <chr>      <dbl>
## 1 2015/01/03    18
## 2 2017/05/18    21
## 3 2018/09/14    15
```

Jointure

Il arrive souvent que l'information qu'on recherche soit répartie entre différentes tables qu'il faut joindre astucieusement. Avec `dplyr`, plusieurs fonctions permettent de réaliser ces opérations. Elles se répartissent en deux groupes :

- Les **mutating joins** servent à augmenter la table primaire à l'aide de variables de la table secondaire ;
- Les **filtering joins** servent à filtrer les individus de la table primaire à l'aide de la table secondaire.

Mutating joins

Il y a quatre fonctions de ce type :

- `left_join()`;
- `right_join()`;
- `inner_join()`;
- `full_join()`.

Clé primaire

Pour faire correspondre les lignes de deux tables, il faut trouver une combinaison de variables :

- présente dans les deux tables ;
- qui permet d'indentifier de façon unique chaque ligne de la table primaire.

```
personnages <- tibble::tribble(
  ~prenom, ~nom, ~hasard,
  "Émile", "Zola", 1.1234,
  "Émile", "Basly", 0.9876,
  "Étienne", "Lantier", 0.8765,
  "Toussaint", "Maheu", 1.2345,
  "Zacharie", "Maheu", 0.9786
)
```

Le couple `c("prenom", "nom")` permet l'identification unique de chaque ligne. Il en va de même pour `c("hasard")`.

Les données (1)

```
fp <- file.path("data", "chanson-francaise.xlsx")
readxl::excel_sheets(fp)
```

```
## [1] "chanteurs" "albums"
```

```
chanteurs <- readxl::read_excel(fp, sheet="chanteurs")
albums <- readxl::read_excel(fp, sheet="albums")
```

Le package **readxl** est adjacent au **tidyverse** mais n'en fais pas partie. Il permet de :

- lister les différentes feuilles d'un classeur;
- lire les feuille et importer les données sous forme de tibble.

Les données (2)

```
chanteurs
```

```
## # A tibble: 4 x 4
##   prenom  nom      naissance  mort
##   <chr>   <chr>        <dbl> <dbl>
## 1 Georges Brassens      1921  1981
## 2 Léo     Ferré         1916  1993
## 3 Jacques Brel          1929  1978
## 4 Renaud  Séchan        1952    NA
```

```
albums
```

```
## # A tibble: 76 x 4
##    titre                            annee prenom  nom
##    <chr>                            <dbl> <chr>   <chr>
##  1 La Mauvaise Réputation            1952 Georges Brassens
##  2 Le Vent                           1953 Georges Brassens
##  3 Les Sabots d'Hélène               1954 Georges Brassens
##  4 Je me suis fait tout petit        1956 Georges Brassens
##  5 Oncle Archibald                   1957 Georges Brassens
##  6 Le Pornographe                    1958 Georges Brassens
##  7 Les Funérailles d'antan           1960 Georges Brassens
##  8 Le temps ne fait rien à l'affaire 1961 Georges Brassens
##  9 Les Trompettes de la renommée     1962 Georges Brassens
## 10 Les Copains d'abord               1964 Georges Brassens
## # ... with 66 more rows
```

left_join()

La fonction `left_join()` s'utilise pour ajouter à une table **primaire** (premier argument de la fonction) des colonnes provenant d'une table **secondaire** ou **étrangère** (deuxième argument de la fonction).

```
chanteurs %>% left_join(albums, by=c("prenom", "nom"))
```

```
## # A tibble: 65 x 6
##    prenom  nom      naissance  mort titre                          annee
##    <chr>   <chr>        <dbl> <dbl> <chr>                          <dbl>
##  1 Georges Brassens      1921  1981 La Mauvaise Réputation          1952
##  2 Georges Brassens      1921  1981 Le Vent                         1953
##  3 Georges Brassens      1921  1981 Les Sabots d'Hélène             1954
##  4 Georges Brassens      1921  1981 Je me suis fait tout petit      1956
##  5 Georges Brassens      1921  1981 Oncle Archibald                 1957
##  6 Georges Brassens      1921  1981 Le Pornographe                  1958
##  7 Georges Brassens      1921  1981 Les Funérailles d'antan         1960
##  8 Georges Brassens      1921  1981 Le temps ne fait rien à l'affaire 1961
##  9 Georges Brassens      1921  1981 Les Trompettes de la renommée   1962
## 10 Georges Brassens      1921  1981 Les Copains d'abord             1964
## # ... with 55 more rows
```

left_join()

```
albums %>%
  filter(annee>1968) %>%
  group_by(prenom, nom) %>%
  summarise(post_soixante_huit=n()) %>%
  left_join(chanteurs, by=c("prenom", "nom")) %>%
  select(prenom, nom, naissance, mort, post_soixante_huit)
```

```
## `summarise()` has grouped output by 'prenom'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 4 x 5
## # Groups:   prenom [4]
##   prenom   nom        naissance  mort post_soixante_huit
##   <chr>    <chr>          <dbl> <dbl>              <int>
## 1 Georges  Brassens        1921  1981                  3
## 2 Jacques  Brel            1929  1978                  2
## 3 Juliette Noureddine       NA    NA                  12
## 4 Léo      Ferré            1916  1993                 21
```

Les autres fonctions

Elles fonctionnent sur le même principe mais diffèrent sur quelques points :

- La fonction `right_join()` est identique à la fonction `left_join()` sauf que c'est le deuxième argument qui sert de table primaire. On ajoute donc des variables du premier argument (table secondaire ou étrangère) au second argument.
- La fonction `inner_join()` est identique à `left_join()` (en particulier le premier argument est la table primaire) mais seules les lignes communes aux deux `data.frame` sont sélectionnées.
- La fonction `full_join()` fait le contraire : toutes les lignes des deux tableaux sont sélectionnées.

inner_join()

Reprenons l'exemple ci-dessus mais remplaçons `left_join()` par `inner_join()` ou par `full_join()` pour mieux comprendre les effets.

```
albums %>%
  filter(annee>1968) %>%
  group_by(prenom, nom) %>%
  summarise(post_soixante_huit=n()) %>%
  inner_join(chanteurs, by=c("prenom", "nom")) %>%
  select(prenom, nom, naissance, mort, post_soixante_huit)
```

```
## `summarise()` has grouped output by 'prenom'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 3 x 5
## # Groups:   prenom [3]
##   prenom  nom      naissance  mort post_soixante_huit
##   <chr>   <chr>        <dbl> <dbl>              <int>
## 1 Georges Brassens      1921  1981                  3
## 2 Jacques Brel          1929  1978                  2
## 3 Léo     Ferré         1916  1993                 21
```

Exemple `full_join()`

```
albums %>%
  filter(annee>1968) %>%
  group_by(prenom, nom) %>%
  summarise(post_soixante_huit=n()) %>%
  full_join(chanteurs, by=c("prenom", "nom")) %>%
  select(prenom, nom, naissance, mort, post_soixante_huit)
```

```
## `summarise()` has grouped output by 'prenom'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 5 x 5
## # Groups:   prenom [5]
##   prenom   nom        naissance  mort post_soixante_huit
##   <chr>    <chr>          <dbl> <dbl>              <int>
## 1 Georges  Brassens         1921  1981                  3
## 2 Jacques  Brel            1929  1978                  2
## 3 Juliette Noureddine         NA    NA                 12
## 4 Léo      Ferré            1916  1993                 21
## 5 Renaud   Séchan           1952    NA                 NA
```

Filtering joins

Il y a deux fonctions de ce type :

- `semi_join()` permet de filtrer des lignes de la table primaire : une ligne est conservée si sa clé primaire se retrouve dans la table secondaire ;
- `anti_join()` fait le contraire. Elle est surtout utile dans une phase exploratoire des données pour vérifier par exemple qu'il n'y a pas de problème dans la façon dont les variables sont orthographiées entre deux tables (on peut penser aux problèmes des majuscules dans les titres des chansons).

semi_join()

Par exemple pour ne conserver que les albums des artistes qui apparaissent dans la table `chanteurs` on peut coder :

```
albums %>% semi_join(chanteurs, by="nom")
```

```
## # A tibble: 64 x 4
##    titre                              annee prenom  nom
##    <chr>                              <dbl> <chr>   <chr>
##  1 La Mauvaise Réputation              1952 Georges Brassens
##  2 Le Vent                             1953 Georges Brassens
##  3 Les Sabots d'Hélène                 1954 Georges Brassens
##  4 Je me suis fait tout petit          1956 Georges Brassens
##  5 Oncle Archibald                     1957 Georges Brassens
##  6 Le Pornographe                      1958 Georges Brassens
##  7 Les Funérailles d'antan             1960 Georges Brassens
##  8 Le temps ne fait rien à l'affaire   1961 Georges Brassens
##  9 Les Trompettes de la renommée       1962 Georges Brassens
## 10 Les Copains d'abord                 1964 Georges Brassens
## # ... with 54 more rows
```

Et tous les albums de Juliette ont disparu ! Ici seules les variables de la table `albums` sont conservées.

anti_join()

Donnons juste un exemple illustratif en considérant l'extrait de table suivant :

```
detail_albums <- tibble::tribble(
  ~titre, ~commentaire,
  "La Mauvaise Réputation", "Aucun commentaire",
  "Le vent", "Pas plus de commentaire"
)
```

```
detail_albums %>% anti_join(albums, by="titre")
```

```
## # A tibble: 1 x 2
##   titre   commentaire
##   <chr>   <chr>
## 1 Le vent Pas plus de commentaire
```

Opérations ensemblistes

Elles sont utilisées pour faires des opérations sur des `tibble` qui ont exactement la même structure (même variables). Il y en a trois :

- `union(df1, df2)` pour la réunion des deux tibbles ;
- `intersect(df1, df2)` pour l'intersection des deux tibbles ;
- `setdiff(df1, df2)` pour sélectionner l'ensembles des individus qui sont dans `df1` mais pas dans `df2`.

```
gaston <- tibble(nom = c("Gaston", "Spirou",  "Fantasio", "Longtarin",
                         "Prunelle", "Lebrac", "Mlle Jeanne", "De Mesmaeker"))
spirou_fantasio <- tibble(nom = c("Spirou", "Fantasio", "Marsupilami", "Seccotine","Spip", "Zorglub"))
```

Personnages qui apparaissent dans les BD de Gaston mais pas dans Spirou et Fantasio, et inversement :

```
setdiff(gaston, spirou_fantasio)
```

```
## # A tibble: 6 x 1
##   nom
##   <chr>
## 1 Gaston
## 2 Longtarin
## 3 Prunelle
## 4 Lebrac
## 5 Mlle Jeanne
## 6 De Mesmaeker
```

```
setdiff(spirou_fantasio, gaston)
```

```
## # A tibble: 4 x 1
##   nom
##   <chr>
## 1 Marsupilami
## 2 Seccotine
## 3 Spip
## 4 Zorglub
```