

Jointure de tables avec {dplyr}

Marie Vaugoyeau

25 June 2024

Table of contents

1	Les données	2
1.1	Import des packages	2
1.2	Température quotidienne	2
1.3	Densité de population par départements	2
2	Concaténation de tables	3
2.1	Pourquoi ?	3
2.2	Problème	3
2.3	Différents types de jointures	3
3	Préparation des données en amont	6
3.1	Identification de la clé de jointure	6
3.2	Même type d'objets	7
3.3	Présence de doublons	7
3.4	Présence de valeurs manquantes	8
4	Jointure total avec full_join()	8
4.1	Réalisation de différents types de jointure	8
4.2	Vérification de la table créée	8
5	Jointure interne avec inner_join()	10
5.1	Création de la jointure la plus stricte	10
5.2	Vérification de la jointure interne	11

1 Les données

1.1 Import des packages

```
library(tidyverse)
```

1.2 Température quotidienne

Les données sont [les températures quotidiennes départementales](#)

```
temperature <- read_delim("https://www.data.gouv.fr/fr/datasets/r/dd0df06a-85f2-4621-8b8b-5a")
  mutate(
    mois = month(date_obs),
    annee = year(date_obs)
  ) |>
  group_by(departement, mois, annee) |>
  summarise(
    tmin = min(tmin, na.rm = TRUE),
    tmax = max(tmax, na.rm = TRUE),
    tmoy = mean(tmoy, na.rm = TRUE)
  ) |>
  ungroup()
```

1.3 Densité de population par départements

ainsi que les [données de densités par départements](#).

```
download.file("https://www.insee.fr/fr/statistiques/fichier/6683035/ensemble.zip", destfile = "data_raw/insee.zip")
unzip("data_raw/insee.zip", exdir = "data_raw")

info_departement <- read_delim("data_raw/donnees_departements.csv", delim = ";")
file.remove("data_raw/insee.zip")
```

```
[1] TRUE
```

2 Concaténation de tables

2.1 Pourquoi ?

Pour coupler des informations présentes dans différentes tables pour explorer le lien, par exemple les variations de températures en fonction de la densité de population.

2.2 Problème

Pour que les jointures se passent bien, il faut que les informations présentes dans une table correspondent à celles présente dans l'autre !

2.3 Différents types de jointures

Prenons deux tables :

```
(A <- tibble(  
  id = letters[1:3],  
  w = c(5, 9, 7),  
  x = c(1, 4, 8)  
))
```

```
# A tibble: 3 x 3  
  id      w      x  
  <chr> <dbl> <dbl>  
1 a         5      1  
2 b         9      4  
3 c         7      8
```

```
(B <- tibble(  
  id = letters[c(1, 2, 4)],  
  y = c(4, 7, 6),  
  z = c(2, 8, 6)  
))
```

```
# A tibble: 3 x 3  
  id      y      z  
  <chr> <dbl> <dbl>  
1 a         4      2  
2 b         7      8  
3 d         6      6
```

On veut avoir les informations w, x, y, z pour tous les individus -> jointure totale avec `full_join()` de `{dplyr}`

```
full_join(A, B)
```

```
# A tibble: 4 x 5
  id      w      x      y      z
  <chr> <dbl> <dbl> <dbl> <dbl>
1 a      5      1      4      2
2 b      9      4      7      8
3 c      7      8     NA     NA
4 d     NA     NA      6      6
```

On veut que les lignes des individus présents dans les deux tableaux -> jointure interne avec `inner_join()` de `{dplyr}`

```
inner_join(A, B)
```

```
# A tibble: 2 x 5
  id      w      x      y      z
  <chr> <dbl> <dbl> <dbl> <dbl>
1 a      5      1      4      2
2 b      9      4      7      8
```

On veut toutes les caractéristiques disponibles pour les individus d'une des deux tables -> jointure à gauche ou à droite avec `left_join()` et `right_join()` de `{dplyr}`. Le sens de la jointure a une influence dans l'ordre des tables mais fait la même chose si on inverse l'ordre

```
left_join(A, B)
```

```
# A tibble: 3 x 5
  id      w      x      y      z
  <chr> <dbl> <dbl> <dbl> <dbl>
1 a      5      1      4      2
2 b      9      4      7      8
3 c      7      8     NA     NA
```

```
right_join(B, A) |>
  relocate(id, letters[23:26])
```

```
# A tibble: 3 x 5
  id      w      x      y      z
  <chr> <dbl> <dbl> <dbl> <dbl>
1 a      5      1      4      2
2 b      9      4      7      8
3 c      7      8     NA     NA
```

```
left_join(B, A) |>
  relocate(id, letters[23:26])
```

```
# A tibble: 3 x 5
  id      w      x      y      z
  <chr> <dbl> <dbl> <dbl> <dbl>
1 a      5      1      4      2
2 b      9      4      7      8
3 d     NA     NA      6      6
```

```
right_join(A, B) |>
  relocate(letters[23:26], .after = id)
```

```
# A tibble: 3 x 5
  id      w      x      y      z
  <chr> <dbl> <dbl> <dbl> <dbl>
1 a      5      1      4      2
2 b      9      4      7      8
3 d     NA     NA      6      6
```

On veut les individus qui ne sont pas présent dans l'autre table -> anti-jointure avec `anti_join()` de `{dplyr}`. L'ordre à son importance, ce sont les individus de la première table qui ne sont pas présent dans la deuxième qui sortent.

```
anti_join(A, B)
```

```
# A tibble: 1 x 3
  id      w      x
  <chr> <dbl> <dbl>
1 c      7      8
```

```
anti_join(B, A)
```

```
# A tibble: 1 x 3
  id      y      z
  <chr> <dbl> <dbl>
1 d      6      6
```

3 Préparation des données en amont

3.1 Identification de la clé de jointure

Si pas défaut les fonctions prennent comme clé de jointure les colonnes qui ont le même nom, ce n'est pas nécessairement ce que l'on veut obtenir.

La clé peut être basé sur une ou plusieurs colonnes.

```
glimpse(info_departement)
```

```
Rows: 100
Columns: 9
$ CODREG <chr> "84", "32", "84", "93", "93", "93", "84", "44", "76", "44", "76~
$ REG    <chr> "Auvergne-Rhône-Alpes", "Hauts-de-France", "Auvergne-Rhône-Alpe~
$ CODDEP <chr> "01", "02", "03", "04", "05", "06", "07", "08", "09", "10", "11~
$ DEP    <chr> "Ain", "Aisne", "Allier", "Alpes-de-Haute-Provence", "Hautes-Al~
$ NBARR  <dbl> 4, 5, 3, 4, 2, 2, 3, 4, 3, 3, 3, 3, 4, 4, 3, 3, 5, 3, 3, 3, 4, ~
$ NBCAN  <dbl> 23, 21, 19, 15, 15, 27, 17, 19, 13, 17, 19, 23, 29, 25, 15, 19,~
$ NBCOM  <dbl> 393, 799, 317, 198, 162, 163, 335, 449, 327, 431, 433, 285, 119~
$ PMUN   <dbl> 657856, 529374, 335628, 165451, 140605, 1097410, 329325, 269701~
$ PTOT   <dbl> 673801, 541176, 344455, 170060, 144981, 1111390, 338147, 275801~
```

```
glimpse(temperature)
```

```
Rows: 7,392
Columns: 6
$ departement <chr> "Ain", "Ain", "Ain", "Ain", "Ain", "Ain", "Ain", "Ain", "A~
$ mois        <dbl> 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3~
$ annee       <dbl> 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2018, 2019, 2020~
$ tmin        <dbl> -1.7, -6.8, -5.2, -8.9, -7.7, -5.5, -6.3, -12.1, -5.9, -3.~
$ tmax        <dbl> 15.7, 11.4, 15.7, 13.4, 13.7, 17.4, 16.0, 11.4, 18.7, 19.8~
$ tmoy        <dbl> 7.559677, 2.195161, 4.374194, 2.093548, 1.266129, 4.827419~
```

```
Ici : DEP == departement
```

3.2 Même type d'objets

Les données doivent-être de même type donc vérification de la classe des clés est indispensable.

```
class(info_departement$DEP) == class(temperature$departement)
```

```
[1] TRUE
```

3.3 Présence de doublons

```
info_departement |>  
  count(DEP) |>  
  arrange(n)
```

```
# A tibble: 100 x 2  
  DEP          n  
  <chr>      <int>  
1 Ain          1  
2 Aisne        1  
3 Allier        1  
4 Alpes-Maritimes 1  
5 Alpes-de-Haute-Provence 1  
6 Ardennes     1  
7 Ardèche     1  
8 Ariège       1  
9 Aube         1  
10 Aude        1  
# i 90 more rows
```

```
temperature |>  
  count(departement) |>  
  filter(n != 77)
```

```
# A tibble: 0 x 2  
# i 2 variables: departement <chr>, n <int>
```

3.4 Présence de valeurs manquantes

```
info_departement |>
  filter(is.na(DEP))
```

```
# A tibble: 0 x 9
# i 9 variables: CODREG <chr>, REG <chr>, CODDEP <chr>, DEP <chr>, NBARR <dbl>,
#   NBCAN <dbl>, NBCOM <dbl>, PMUN <dbl>, PTOT <dbl>
```

```
temperature |>
  filter(is.na(departement))
```

```
# A tibble: 0 x 6
# i 6 variables: departement <chr>, mois <dbl>, annee <dbl>, tmin <dbl>,
#   tmax <dbl>, tmoy <dbl>
```

Ce n'est pas nécessairement un problème d'avoir des doublons ou des valeurs manquantes mais il faut que cela correspondent à ce que l'on souhaite faire.

4 Jointure total avec `full_join()`

4.1 Réalisation de différents types de jointure

```
jointure_totale <- full_join(
  info_departement,
  temperature,
  by = join_by(DEP == departement)
)
```

4.2 Vérification de la table créée

Calcul de la taille attendue

```
dim(info_departement)
```

```
[1] 100    9
```



```
dim(temperature)
```

```
[1] 7392    6
```

```
# calcul du nombre de colonnes attendue  
ncol(info_departement) + ncol(temperature) - 1 == ncol(jointure_totale)
```

```
[1] TRUE
```

```
# vérification du nombre de lignes  
nrow(temperature) == nrow(jointure_totale)
```

```
[1] FALSE
```

Recherche des NA

```
jointure_totale |>  
  filter(is.na(CODREG))
```

```
# A tibble: 0 x 14  
#   i 14 variables: CODREG <chr>, REG <chr>, CODDEP <chr>, DEP <chr>,  
#   NBARR <dbl>, NBCAN <dbl>, NBCOM <dbl>, PMUN <dbl>, PTOT <dbl>, mois <dbl>,  
#   annee <dbl>, tmin <dbl>, tmax <dbl>, tmoy <dbl>
```

```
jointure_totale |>  
  filter(is.na(mois)) |>  
  View()
```

Possibilité d'utiliser l'anti-jointure pour identifier les lignes à problème

```
anti_join(  
  info_departement,  
  temperature,  
  by = join_by(DEP == departement)  
)
```

```
# A tibble: 4 x 9
  CODREG REG      CODDEP DEP      NBARR NBCAN NBCOM  PMUN  PTOT
  <chr> <chr>    <chr> <chr>    <dbl> <dbl> <dbl> <dbl> <dbl>
1 01    Guadeloupe 971    Guadeloupe 2     21    32 383559 388727
2 02    Martinique 972    Martinique 4     NA    34 361225 365734
3 03    Guyane    973    Guyane    2     NA    22 285133 287355
4 04    La Réunion 974    La Réunion 4     25    24 863083 872635
```

```
anti_join(
  temperature,
  info_departement,
  by = join_by(departement == DEP)
)
```

```
# A tibble: 0 x 6
# i 6 variables: departement <chr>, mois <dbl>, annee <dbl>, tmin <dbl>,
#   tmax <dbl>, tmoy <dbl>
```

5 Jointure interne avec inner_join()

5.1 Création de la jointure la plus stricte

```
jointure_interne <- inner_join(
  info_departement,
  temperature,
  by = join_by(DEP == departement)
)

# même résultat avec la jointure à droite
jointure_a_droite <- right_join(
  info_departement,
  temperature,
  by = join_by(DEP == departement))

# vérification
identical(jointure_interne, jointure_a_droite)
```

```
[1] TRUE
```

5.2 Vérification de la jointure interne

```
jointure_interne |>  
  count(DEP) |>  
  filter(n != 77)
```

```
# A tibble: 0 x 2  
# i 2 variables: DEP <chr>, n <int>
```