Joindre des tables de données

GRAPH Network & WHO, supported by the Global Fund to fight HIV, TB & Malaria

October 2023

This document is a draft of a lesson made by the GRAPH Network, a non-profit headquartered at the University of Geneva Global Health Institute, in collaboration with the World Health Organization, under a Global Fund 2023 grant to create e-learning modules to build in-country data capacity for epidemiological and impact analysis for National HIV, TB and malaria programs

Prélude

La jointure de bases de données est une compétence cruciale lorsqu'on travaille avec des données relatives à la santé car elle permet de combiner des informations provenant de plusieurs sources, conduisant à des analyses plus complètes et perspicaces. Dans cette leçon, vous apprendrez à utiliser différentes techniques de jointure à l'aide du package `dplyr` de R. Commençons !

Objectifs d'apprentissage

- Vous comprenez comment fonctionnent les différentes jointures de `dplyr`
- Vous êtes capable de choisir la jointure appropriée pour vos données
- Vous pouvez joindre des ensembles de données simples en utilisant des fonctions de `dplyr`

Paquets

Veuillez charger les paquets nécessaires à cette leçon avec le code ci-dessous :

```
# Charger les paquets
if(!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse,
               countrycode)
```

Qu'est-ce qu'une jointure et pourquoi en avons-nous besoin ?

Pour illustrer l'utilité des jointures, commençons par un exemple de jouet. Considérez les deux ensembles de données suivants. Le premier, `demographique`, contient les noms et les âges de trois patients :

```
demographique <-
  tribble(~nom,      ~age,
          "Alice",    25,
          "Bob",      32,
          "Charlie",  45)
demographique
```

```
## # A tibble: 3 × 2
##   nom       age
##   <chr>   <dbl>
## 1 Alice      25
## 2 Bob        32
## 3 Charlie    45
```

Le deuxième, `info_test`, contient les dates et les résultats des tests de tuberculose pour ces patients :

```
info_test <-
  tribble(~nom,      ~date_du_test,    ~resultat,
          "Alice",   "2023-06-05",  "Négatif",
          "Bob",     "2023-08-10",  "Positif",
          "Charlie", "2023-07-15",  "Négatif")
info_test
```

```
## # A tibble: 3 × 3
##   nom     date_du_test resultat
##   <chr>   <chr>        <chr>
## 1 Alice   2023-06-05   Négatif
## 2 Bob     2023-08-10   Positif
## 3 Charlie 2023-07-15   Négatif
```

Nous aimerions analyser ces données ensemble, et nous avons donc besoin d'une façon de les combiner.

Une option que nous pourrions envisager est la fonction `cbind()` de base R (`cbind` est l'abréviation de column bind) :

```
cbind(demographique, info_test)
```

nom	age	nom	date_du_test	resultat
Alice	25	Alice	2023-06-05	Négatif

nom	age	nom	date_du_test	resultat
Bob	32	Bob	2023-08-10	Positif
Charlie	45	Charlie	2023-07-15	Négatif

Cela fusionne avec succès les ensembles de données, mais il ne le fait pas très intelligemment. La fonction "colle" ou "agrafe" essentiellement les deux tables ensemble. Ainsi, comme vous pouvez le remarquer, la colonne "nom" apparaît deux fois. Ce n'est pas idéal et cela posera problème pour l'analyse.

Un autre problème se pose si les lignes des deux ensembles de données ne sont pas déjà alignées. Dans ce cas, les données seront combinées de manière incorrecte avec `cbind()`. Considérez l'ensemble de données `info_test_desordonne` ci-dessous, qui a maintenant Bob dans la première ligne :

```
info_test_desordonne <-
  tribble(~nom,      ~date_du_test,    ~resultat,
          "Bob",     "2023-08-10",  "Positif", # Bob in first row
          "Alice",   "2023-06-05",  "Négatif",
          "Charlie", "2023-07-15",  "Négatif")
```

Qu'arrive-t-il si nous `cbind()` ceci avec l'ensemble de données `demographique` original, où Bob était dans la *deuxième* ligne ?

```
cbind(demographique, info_test_desordonne)
```

nom	age	nom	date_du_test	resultat
Alice	25	Bob	2023-08-10	Positif
Bob	32	Alice	2023-06-05	Négatif
Charlie	45	Charlie	2023-07-15	Négatif

Les détails démographiques d'Alice sont maintenant alignés par erreur avec les informations de test de Bob !

Un troisième problème se pose lorsqu'une entité apparaît plus d'une fois dans un ensemble de données. Peut-être qu'Alice a fait plusieurs tests de TB :

```
info_test_multiple <-
  tribble(~nom,      ~date_du_test,    ~resultat,
          "Alice",   "2023-06-05",  "Négatif",
          "Alice",   "2023-06-06",  "Négatif",
          "Bob",     "2023-08-10",  "Positif",
          "Charlie", "2023-07-15",  "Négatif")
```

Si nous essayons de `cbind()` ceci avec l'ensemble de données `demographique`, nous obtiendrons une erreur, due à une incohérence dans le nombre de lignes :

```
cbind(demographique, info_test_multiple)
```

```
Erreur dans data.frame(..., check.names = FALSE) :
  les arguments impliquent un nombre différent de lignes : 3, 4
```

Ce que nous avons ici est appelé une relation `un-à-plusieurs`—une Alice dans les données démographiques, mais plusieurs lignes Alice dans les données de test. La jointure dans de tels cas sera couverte en détail dans la deuxième leçon de jointure.

Il est évident que nous avons besoin d'une manière plus intelligente de combiner les jeux de données que `cbind()` ; nous devrons nous aventurer dans le monde des jointures.

Commençons par la jointure la plus courante, la `left_join()`, qui résout les problèmes auxquels nous avons été précédemment confrontés.

Elle fonctionne pour le cas simple, et elle ne duplique pas la colonne nom :

```
left_join(demographique, info_test)
```

```
## Joining with `by = join_by(nom)`
```

```
## # A tibble: 3 × 4
##   nom       age date_du_test resultat
##   <chr>   <dbl> <chr>        <chr>
## 1 Alice      25 2023-06-05   Négatif
## 2 Bob        32 2023-08-10   Positif
## 3 Charlie    45 2023-07-15   Négatif
```

Elle fonctionne lorsque les jeux de données ne sont pas ordonnés de la même manière :

```
left_join(demographique, info_test_desordonne)
```

```
## Joining with `by = join_by(nom)`
```

```
## # A tibble: 3 × 4
##   nom       age date_du_test resultat
##   <chr>   <dbl> <chr>        <chr>
## 1 Alice      25 2023-06-05   Négatif
## 2 Bob        32 2023-08-10   Positif
## 3 Charlie    45 2023-07-15   Négatif
```

Et elle fonctionne quand il y a plusieurs lignes de test par patient :

```
left_join(demographique, info_test_multiple)
```

```
## Joining with `by = join_by(nom)`
```

```
## # A tibble: 4 × 4
##   nom       age date_du_test resultat
##   <chr>   <dbl> <chr>        <chr>
## 1 Alice      25 2023-06-05   Négatif
## 2 Alice      25 2023-06-06   Négatif
## 3 Bob        32 2023-08-10   Positif
## 4 Charlie    45 2023-07-15   Négatif
```

Simple et magnifique !

::: note latérale

Nous utiliserons également l'opérateur pipe lors des jointures. Souvenez-vous que ceci :

```
demographique %>% left_join(info_test)
```

```
## Joining with `by = join_by(nom)`
```

est équivalent à ceci :

```
left_join(demographique, info_test)
```

```
## Joining with `by = join_by(nom)`
```

:::

Syntaxe des jointures

Maintenant que nous comprenons *pourquoi* nous avons besoin de jointures, regardons leur syntaxe de base.

Les jointures prennent deux dataframes comme deux premiers arguments : `x` (le dataframe *à gauche*) et `y` (le dataframe *à droite*). Comme pour les autres fonctions de R, vous pouvez fournir ces arguments avec ou sans nom :

```
# les deux sont identiques :
left_join(x = demographique, y = info_test) # nommé
left_join(demographique, info_test) # sans nom
```

Un autre argument crucial est `by`, qui indique la colonne ou **clé** utilisée pour connecter les tables. Nous n'avons pas toujours besoin de fournir cet argument ; il peut être *inféré* à partir des jeux de données. Par exemple, dans nos exemples originaux, "nom" est la seule colonne commune à `demographique` et `info_test`. Ainsi, la fonction de jointure suppose `by = "nom"` :

```
# ces deux sont équivalentes
left_join(x = demographique, y = info_test)
left_join(x = demographique, y = info_test, by = "nom")
```

La colonne utilisée pour connecter les lignes entre les tables est connue sous le nom de "clé". Dans les fonctions de jointure de dplyr, la clé est spécifiée dans l'argument `by`, comme on le voit dans `left_join(x = demographique, y = info_test, by = "nom")`

Que se passe-t-il si les clés sont nommées différemment dans les deux jeux de données ? Considérez le jeu de données `info_test_nom_different` ci-dessous, où la colonne "nom" a été modifiée en "destinataire_test" :

```
info_test_nom_different <-
  tribble(~destinataire_test,  ~date_test,    ~resultat,
          "Alice",           "2023-06-05",  "Négatif",
          "Bob",             "2023-08-10",  "Positif",
          "Charlie",         "2023-07-15",  "Négatif")
info_test_nom_different
```

```
## # A tibble: 3 × 3
##   destinataire_test date_test  resultat
##   <chr>             <chr>      <chr>
## 1 Alice             2023-06-05 Négatif
## 2 Bob               2023-08-10 Positif
## 3 Charlie           2023-07-15 Négatif
```

Si nous essayons de joindre `info_test_nom_different` à notre jeu de données `demographique` original, nous rencontrerons une erreur :

```
left_join(x = demographique, y = info_test_nom_different)
```

```
Erreur dans `left_join()` :
! `by` doit être fourni lorsque `x` et `y` n'ont pas de
```

```
variables communes.
ℹ Utiliser `cross_join()` pour effectuer une jointure croisée.
```

L'erreur indique qu'il n'y a pas de variables communes, donc la jointure n'est pas possible.

Dans des situations comme celle-ci, vous avez deux choix : vous pouvez renommer la colonne dans le deuxième dataframe pour qu'elle corresponde à la première, ou plus simplement, spécifier sur quelles colonnes joindre en utilisant `by = c()`.

Voici comment faire cela :

```
left_join(x = demographique, y = info_test_nom_different,
          by = c("nom" = "destinataire_test"))
```

```
## # A tibble: 3 × 4
##   nom       age date_test  resultat
##   <chr>   <dbl> <chr>      <chr>
## 1 Alice      25 2023-06-05 Négatif
## 2 Bob        32 2023-08-10 Positif
## 3 Charlie    45 2023-07-15 Négatif
```

La syntaxe `c("nom" = "destinataire_test")` est un peu inhabituelle. Elle dit essentiellement, "Connecte `nom` du dataframe x avec `destinataire_test` du dataframe y parce qu'ils représentent les mêmes données."

Considérez les deux ensembles de données ci-dessous, l'un avec les détails des patients et l'autre avec les dates de contrôle médical pour ces patients.

```
patients <- tribble(
  ~id_patient, ~nom,      ~age,
  1,           "Jean",     32,
  2,           "Joie",    28,
  3,           "Khan",      40
)

controles <- tribble(
  ~id_patient, ~date_controle,
  1,           "2023-01-20",
  2,           "2023-02-20",
  3,           "2023-05-15"
)
```

Joignez l'ensemble de données `patients` avec l'ensemble de données `controles` en utilisant `left_join()`

Deux ensembles de données sont définis ci-dessous, l'un avec les détails des patients et l'autre avec les registres de vaccination pour ces patients.

```r
# Détails des patients
details_patient <- tribble(
  ~numero_id,  ~nom_complet,   ~adresse,
  "A001",      "Alice",      "123 Elm St",
  "B002",      "Bob",        "456 Maple Dr",
  "C003",      "Charlie",    "789 Oak Blvd"
)

# Registres de vaccination
registres_vaccination <- tribble(
  ~code_patient, ~type_vaccin,  ~date_vaccination,
  "A001",        "COVID-19",     "2022-05-10",
  "B002",        "Grippe",          "2023-09-01",
  "C003",        "Hépatite B",  "2021-12-15"
)
```

Joignez les ensembles de données `details_patient` et `registres_vaccination`. Vous devrez utiliser l'argument `by` car les colonnes identifiant le patient ont des noms différents.

Types de jointures

Les exemples jouets jusqu'à présent ont impliqué des ensembles de données qui pouvaient être parfaitement correspondants - chaque ligne dans un ensemble de données avait une ligne correspondante dans l'autre ensemble de données.

Les données du monde réel sont généralement plus désordonnées. Souvent, il y aura des entrées dans la première table qui n'ont pas d'entrées correspondantes dans la deuxième table, et vice versa.

Pour gérer ces cas de correspondance imparfaite, il existe différents types de jointures avec des comportements spécifiques : `left_join()`, `right_join()`, `inner_join()` et `full_join()`. Dans les sections à venir, nous examinerons des exemples de la manière dont chaque type de jointure opère sur des ensembles de données avec des correspondances imparfaites.

`left_join()`

Commençons par `left_join()`, que vous avez déjà rencontré. Pour voir comment il gère les lignes non appariées, nous allons essayer de joindre notre ensemble de données `demographique` original avec une version modifiée de l'ensemble de données `infos_test`.

Pour rappel, voici l'ensemble de données `demographique`, avec Alice, Bob et Charlie :

```r
demographique
```

```
## # A tibble: 3 × 2
##   nom       age
##   <chr>   <dbl>
## 1 Alice      25
## 2 Bob        32
## 3 Charlie    45
```

Pour les informations de test, nous allons supprimer `Charlie` et nous allons ajouter un nouveau patient, `Xavier`, et ses données de test :

```
infos_test_xavier <- tribble(
  ~nom,    ~date_test, ~resultat,
  "Alice",  "2023-06-05", "Négatif",
  "Bob",    "2023-08-10", "Positif",
  "Xavier", "2023-05-02", "Négatif")
infos_test_xavier
```

```
## # A tibble: 3 × 3
##   nom    date_test  resultat
##   <chr>  <chr>      <chr>
## 1 Alice  2023-06-05 Négatif
## 2 Bob    2023-08-10 Positif
## 3 Xavier 2023-05-02 Négatif
```

Si nous effectuons un `left_join()` en utilisant `demographique` comme ensemble de données de gauche (`x = demographique`) et `infos_test_xavier` comme ensemble de données de droite (`y = infos_test_xavier`), à quoi devrions-nous nous attendre ? Rappelons que Charlie n'est présent que dans l'ensemble de données de gauche, et Xavier n'est présent que dans celui de droite. Eh bien, voici ce qui se passe :

```
left_join(x = demographique, y = infos_test_xavier, by = "nom")
```

Comme vous pouvez le voir, avec la jointure *LEFT*, tous les enregistrements du dataframe *LEFT* (`demographique`) sont conservés. Donc, même si `Charlie` n'a pas de correspondance dans l'ensemble de données `infos_test_xavier`, il est toujours inclus dans la sortie. (Mais bien sûr, comme ses informations de test ne sont pas disponibles dans `infos_test_xavier`, ces valeurs ont été laissées à `NA`.)

Xavier, en revanche, qui n'était présent que dans l'ensemble de données de droite, est supprimé.

Le graphique ci-dessous montre comment cette jointure a fonctionné :

Dans une fonction de jointure telle que `left_join(x, y)`, l'ensemble de données fourni à l'argument `x` peut être appelé l'ensemble de données "de gauche", tandis que l'ensemble de données attribué à l'argument `y` peut être appelé l'ensemble de données "de droite".

Et si nous inversions les ensembles de données ? Voyons le résultat lorsque `infos_test_xavier` est l'ensemble de données de gauche et `demographique` celui de droite :

```
left_join(x = infos_test_xavier, y = demographique, by = "nom")
```

Encore une fois, `left_join()` conserve toutes les lignes de l'ensemble de données *de gauche* (maintenant `infos_test_xavier`). Cela signifie que les données de Xavier sont incluses cette fois. Charlie, en revanche, est exclu.

Le graphique ci-dessous illustre comment cela fonctionne:

Ensemble de données principal : Dans le contexte des jointures, l'ensemble de données principal désigne l'ensemble de données principal ou priorisé dans une opération. Dans une jointure à gauche, l'ensemble de données de gauche est considéré comme l'ensemble de données principal car toutes ses lignes sont conservées dans le résultat, qu'elles aient ou non une ligne correspondante dans l'autre ensemble de données.

Essayez ce qui suit. Voici deux ensembles de données - l'un avec des diagnostics de maladie (`dx_maladie`) et un autre avec des données démographiques de patients (`demographique_patient`).

```
dx_maladie <- tribble(
  ~id_patient, ~maladie,       ~date_diagnostic,
  1,           "Influenza",    "2023-01-15",
  3,           "COVID-19",     "2023-03-05",
  8,           "Influenza",    "2023-02-20",
)

demographique_patient <- tribble(
  ~id_patient, ~nom,       ~age,  ~genre,
  1,           "Fred",       28,  "Femme",
  2,           "Genevieve",  45,  "Femme",
  3,           "Henry",      32,  "Homme",
  5,           "Irene",      55,  "Femme",
  8,           "Jules",      40,  "Homme"
)
```

Utilisez `left_join()` pour fusionner ces ensembles de données, en ne conservant que les patients pour lesquels nous avons des informations démographiques. Réfléchissez bien à quel ensemble de données mettre à gauche.

Essayons un autre exemple, cette fois avec un ensemble de données plus réaliste.

Premièrement, nous avons des données sur le taux d'incidence de la tuberculose par 100 000 personnes pour 47 pays africains, de l'OMS :

```
tb_2019_afrique <- read_csv(here("data/tb_incidence_2019.csv"))
```

```
## Rows: 47 Columns: 3
## ── Column specification ────────────────────────────────────
## Delimiter: ","
## chr (2): country, conf_int_95
## dbl (1): cases
##
## 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.
```

```
tb_2019_afrique
```

Nous voulons analyser comment l'incidence de la TB dans les pays africains varie avec les dépenses de santé par habitant du gouvernement. Pour cela, nous avons des données sur les dépenses de santé par habitant en USD, également de l'OMS :

```
dep_sante_2019 <- read_csv(here("data/health_expend_per_cap_2019.csv"))
```

```
## Rows: 185 Columns: 2
## ── Column specification ────────────────────────────────────
## Delimiter: ","
## chr (1): country
## dbl (1): expend_usd
##
## 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.
```

```
dep_sante_2019
```

Quel ensemble de données devrions-nous utiliser comme dataframe de gauche pour la jointure ?

Comme notre objectif est d'analyser les pays africains, nous devrions utiliser `tb_2019_afrique` comme dataframe de gauche. Cela garantira que nous gardons tous

les pays africains dans l'ensemble de données joint final.

Faisons la jointure :

```
tb_dep_sante_joint <-
  tb_2019_afrique %>%
  left_join(dep_sante_2019, by = "country")
tb_dep_sante_joint
```

Maintenant, dans l'ensemble de données joint, nous avons juste les 47 lignes pour les pays africains, ce qui est exactement ce que nous voulions !

Toutes les lignes du dataframe de gauche `tb_2019_afrique` ont été conservées, tandis que les pays non africains de `dep_sante_2019` ont été écartés.

Nous pouvons vérifier si certaines lignes de `tb_2019_afrique` n'ont pas eu de correspondance dans `dep_sante_2019` en filtrant pour les valeurs `NA` :

```
tb_dep_sante_joint %>%
  filter(is.na(!expend_usd))
```

```
## # A tibble: 3 × 4
##   country     cases conf_int_95 expend_usd
##   <chr>       <dbl> <chr>            <dbl>
## 1 Mauritius      12 [9 - 15]            NA
## 2 South Sudan   227 [147 - 324]         NA
## 3 Comoros        35 [23 - 50]           NA
```

Cela montre que 3 pays - Maurice, le Soudan du Sud et les Comores - n'avaient pas de données sur les dépenses dans `dep_sante_2019`. Mais comme ils étaient présents dans `tb_2019_afrique`, et que c'était le dataframe de gauche, ils ont quand même été inclus dans les données jointes.

Pour en être sûr, nous pouvons rapidement confirmer que ces pays sont absents de l'ensemble de données sur les dépenses avec une déclaration de filtre :

```
dep_sante_2019 %>%
  filter(country %in% c("Mauritius", "South Sudan", "Comoros"))
```

```
## # A tibble: 0 × 2
## # ℹ 2 variables: country <chr>, expend_usd <dbl>
```

En effet, ces pays ne sont pas présents dans `dep_sante_2019`.

Copiez le code ci-dessous pour définir deux ensembles de données.

Le premier, `cas_tb_enfants` contient le nombre de cas de TB chez les moins de 15 ans en 2012, par pays :

```
cas_tb_enfants <- tidyr::who %>%
  filter(year == 2012) %>%
  transmute(country, cas_tb_smear_0_14 = new_sp_m014 + new_sp_f014)

cas_tb_enfants
```

```
## # A tibble: 5 × 2
##   country        cas_tb_smear_0_14
##   <chr>                      <dbl>
## 1 Afghanistan                  588
## 2 Albania                        0
## 3 Algeria                       89
## 4 American Samoa                NA
## 5 Andorra                        0
```

Et `pays_continents`, du package {countrycode}, liste tous les pays et leur région et continent correspondants :

```
pays_continents <-
  countrycode::codelist %>%
  select(country.name.fr, continent, region)

pays_continents
```

```
## # A tibble: 5 × 3
##   country.name.fr   continent region
##   <chr>             <chr>     <chr>
## 1 Afghanistan       Asia      South Asia
## 2 Albanie           Europe    Europe & Central Asia
## 3 Algérie           Africa    Middle East & North Africa
## 4 Samoa américaines Oceania   East Asia & Pacific
## 5 Andorre           Europe    Europe & Central Asia
```

Votre objectif est d'ajouter les données de continent et de région à l'ensemble de données sur les cas de TB.

Quel ensemble de données devrait être le dataframe de gauche, `x` ? Et lequel devrait être le droit, `y` ? Une fois que vous avez décidé, joignez les ensembles de données de manière appropriée en utilisant `left_join()`.

`right_join()`

Un `right_join()` peut être considéré comme une image miroir d'un `left_join()`. Les mécanismes sont les mêmes, mais maintenant toutes les lignes de l'ensemble de données de *DROITE* sont conservées, tandis que seules les lignes de l'ensemble de données de gauche qui trouvent une correspondance à droite sont conservées.

Regardons un exemple pour comprendre cela. Nous utiliserons nos ensembles de données `demographique` et `infos_test_xavier` originaux :

```
demographique
```

```
infos_test_xavier
```

Essayons maintenant `right_join()`, avec `demographique` comme dataframe de droite :

```
right_join(x = infos_test_xavier, y = demographique)
```

```
## Joining with `by = join_by(nom)`
```

J'espère que vous commencez à comprendre cela, et que vous pourriez prédire cette sortie ! Puisque `demographique` était le dataframe de *droite*, et que nous utilisons *right*-join, toutes les lignes de `demographique` sont conservées—Alice, Bob et Charlie. Mais seulement les enregistrements correspondants dans le dataframe de gauche `infos_test_xavier` !

Le graphique ci-dessous illustre ce processus :

Un point important—le même dataframe final peut être créé avec `left_join()` ou `right_join()`, cela dépend simplement de l'ordre dans lequel vous fournissez les dataframes à ces fonctions :

```
# ici, RIGHT_join privilégie le df de DROITE, demographique
right_join(x = infos_test_xavier, y = demographique)
```

```
## Joining with `by = join_by(nom)`
```

```
# ici, LEFT_join privilégie le df de GAUCHE, encore une fois demographique
left_join(x = demographique, y = infos_test_xavier)
```

```
## Joining with `by = join_by(nom)`
```

La seule différence que vous pourriez remarquer entre left et right-join est que l'ordre final des colonnes est différent. Mais les colonnes peuvent facilement être réarrangées, donc se soucier de l'ordre des colonnes n'en vaut vraiment pas la peine.

Comme nous l'avons mentionné précédemment, les data scientists favorisent généralement `left_join()` par rapport à `right_join()`. Il est plus logique de spécifier votre ensemble de données principal d'abord, dans la position de gauche. Opter pour un `left_join()` est une bonne pratique courante en raison de sa logique plus claire, ce qui le rend moins sujet à l'erreur.

Super, maintenant nous comprenons comment fonctionnent `left_join()` et `right_join()`, passons à `inner_join()` et `full_join()` !

`inner_join()`

Ce qui distingue un `inner_join`, c'est que les lignes ne sont conservées que si les valeurs de jointure sont présentes dans *les deux* dataframes. Revenons à notre exemple de patients et de leurs résultats de test COVID. Pour rappel, voici nos ensembles de données :

```
demographique
```

```
infos_test_xavier
```

Maintenant que nous avons une meilleure compréhension de la façon dont fonctionnent les jointures, nous pouvons déjà imaginer à quoi ressemblerait le dataframe final si nous utilisions un `inner_join()` sur nos deux dataframes ci-dessus. Si seules les lignes avec des valeurs de jointure qui sont dans *les deux* dataframes sont conservées, et que les seuls patients qui sont à la fois dans `demographique` et `infos_test` sont `Alice` et `Bob`, alors ils devraient être les seuls patients dans notre ensemble de données final ! Essayons.

```
inner_join(demographique, infos_test_xavier, by="nom")
```

Parfait, c'est exactement ce à quoi nous nous attendions ! Ici, `Charlie` était seulement dans l'ensemble de données `demographique`, et `Xavier` était seulement dans l'ensemble de données `infos_test`, donc tous deux ont été supprimés. Le graphique ci-dessous montre comment fonctionne cette jointure :

Il est logique que l'ordre dans lequel vous spécifiez vos ensembles de données ne change pas les informations qui sont conservées, étant donné que vous avez besoin de valeurs de jointure dans les deux ensembles de données pour qu'une ligne soit conservée. Pour illustrer cela, essayons de changer l'ordre de nos ensembles de données.

```
inner_join(infos_test_xavier, demographique, by="nom")
```

Comme prévu, la seule différence ici est l'ordre de nos colonnes, sinon les informations conservées sont les mêmes. ::: r-pratique

Les données suivantes concernent les épidémies d'origine alimentaire aux États-Unis en 2019, provenant du CDC. Copiez le code ci-dessous pour créer deux nouveaux dataframes :

```
total_inf <- tribble(
  ~pathogene,          ~total_infections,
  "Campylobacter",     9751,
  "Listeria",          136,
  "Salmonella",        8285,
  "Shigella",          2478,
)

resultats <- tribble(
  ~pathogene,         ~n_hosp,     ~n_deces,
  "Listeria",           128,         30,
  "STEC",               582,         11,
  "Campylobacter",      1938,        42,
  "Yersinia",           200,         5,
)
```

Quels sont les pathogènes communs entre les deux ensembles de données ? Utilisez un `inner_join()` pour joindre les dataframes, afin de ne conserver que les pathogènes qui figurent dans les deux ensembles de données.

:::

Revenons à nos données sur les dépenses de santé et le niveau de revenu et appliquons ce que nous avons appris à ces ensembles de données.

```
tb_2019_afrique
```

```
dep_sante_2019
```

Ici, nous pouvons créer un nouveau dataframe appelé `inner_exp_tb` en utilisant un `inner_join()` pour ne conserver que les pays pour lesquels nous avons à la fois des données sur les dépenses de santé et les taux d'incidence de la tuberculose. Essayons-le maintenant :

```
inner_dep_tb <- tb_2019_afrique %>%
  inner_join(dep_sante_2019)
```

```
## Joining with `by = join_by(country)`
```

```
inner_dep_tb
```

Cela semble très bien ! Avec `left_join()`, le `inner_join()` est l'une des jointures les plus courantes lorsqu'on travaille avec des données, il est donc probable que vous y serez souvent confronté. C'est un outil puissant et souvent utilisé, mais c'est aussi la jointure qui exclut le plus d'informations, alors assurez-vous que vous ne voulez que des enregistrements correspondants dans votre ensemble de données final ou vous risquez de perdre accidentellement beaucoup de données ! En revanche, `full_join()` est la jointure la plus inclusive, regardons-la dans la prochaine section.

Utilisez un `inner_join()` pour joindre les dépenses de santé 2019 et . Quels pays restent dans votre ensemble de données final ?

`full_join()`

La particularité de `full_join()` est qu'il conserve *tous* les enregistrements, qu'il y ait ou non une correspondance entre les deux jeux de données. Lorsqu'il manque des informations dans notre jeu de données final, les cellules sont définies sur `NA` comme nous l'avons vu dans `left_join()` et `right_join()`.

Jetons un coup d'œil à nos jeux de données `Demographique` et `test_info` pour illustrer cela.

Voici un rappel de nos données :

```
demographique
```

```
infos_test_xavier
```

Maintenant, effectuons un `full_join`, avec `Demographique` comme nos données principal.

```
full_join(demographique, infos_test_xavier, by="nom")
```

Comme nous pouvons le voir, toutes les lignes ont été conservées donc il n'y a eu aucune perte d'information ! Le graphique ci-dessous illustre ce processus :

Comme cette jointure n'est pas sélective, tout se retrouve dans l'ensemble de données final, donc changer l'ordre de nos ensembles de données ne changera pas les informations qui sont conservées. Cela ne changera que l'ordre des colonnes dans notre ensemble de données final. Nous pouvons le voir ci-dessous lorsque nous spécifions `test_info` (traduit par `info_test`) comme notre ensemble de données principal et `Demographic` (traduit par `Démographique`) comme notre ensemble de données secondaire.

```
full_join(infos_test_xavier, demographique, by="nom")
```

Comme nous l'avons vu ci-dessus, toutes les données des deux ensembles de données d'origine sont toujours là, avec toute information manquante définie à `NA`. ::: r-pratique Les dataframes suivantes contiennent les taux d'incidence mondiaux du paludisme par 100'000 personnes et les taux de mortalité mondiaux par 100'000 personnes dus au paludisme, provenant de Our World in Data. Copiez le code pour créer deux petits dataframes :

```
inc_paludisme <- tribble(
  ~année, ~inc_100k,
  2010, 69.485344,
  2011, 66.507935,
  2014, 59.831020,
  2016, 58.704540,
  2017, 59.151703,
)

deces_paludisme <- tribble(
  ~année, ~deces_100k,
  2011, 12.92,
  2013, 11.00,
  2015, 10.11,
  2016, 9.40,
  2019, 8.95
)
```

Ensuite, joignez les tables ci-dessus en utilisant un `full_join()` afin de conserver toutes les informations des deux ensembles de données.

:::

Revenons à notre ensemble de données sur la tuberculose et notre ensemble de données sur les dépenses de santé.

```
tb_2019_afrique
```

```
## # A tibble: 5 × 3
##   country               cases conf_int_95
##   <chr>                 <dbl> <chr>
## 1 Burundi                 107 [69 - 153]
## 2 Sao Tome and Principe   114 [45 - 214]
## 3 Senegal                 117 [83 - 156]
## 4 Mauritius                12 [9 - 15]
## 5 Côte d'Ivoire           137 [88 - 197]
```

```
dep_sante_2019
```

```
## # A tibble: 5 × 2
##   country              expend_usd
##   <chr>                     <dbl>
## 1 Nigeria                    11.0
## 2 Bahamas                  1002
## 3 United Arab Emirates     1015
## 4 Nauru                    1038
## 5 Slovakia                 1058
```

Maintenant, créons un nouveau dataframe appelé `full_tb_sante` en utilisant un full_join !

```
inner_dep_tb <- tb_2019_afrique %>%
  inner_join(dep_sante_2019)
```

```
## Joining with `by = join_by(country)`
```

```
inner_dep_tb
```

Comme nous l'avons vu précédemment, toutes les lignes ont été conservées entre les deux ensembles de données avec des valeurs manquantes définies à `NA`.

Comme pour l'exercice précédent, obtenez tous les pays et leur région et continent correspondants à partir du package {countrycode} :

```
pays_continents <-
  countrycode::codelist %>%
  select(country.name.fr, continent, region)
```

Ensuite, utilisez un `full_join()` pour joindre avec l'ensemble de données `tb_2019_afrique`.

Résumé

Bravo, vous comprenez maintenant les bases de la jointure ! Le diagramme de Venn ci-dessous donne un résumé utile des différentes jointures et des informations que chacune conserve. Il peut être utile de sauvegarder cette image pour référence future !