Joining 2: Joining Real-World Datasets

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

Introduction

Now that we have a solid grasp on the different types of joins and how they work, we can look at how to manage messier and more complex datasets. Joining real-world data from different sources often requires a bit of thought and cleaning ahead of time.

Learning Objectives

- You know how to check for mismatched values between dataframes
- You understand how to join using a one-to-many match
- You know how to join on multiple key columns

Packages

Please load the packages needed for this lesson with the code below:

```
if(!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse)
```

Pre-cleaning data

Often you will need to pre-clean your data when you draw it from different sources before you're able to join it. This is because there can be discrepancies in ways that values are recorded in different tables such as spelling errors, differences in capitalization, and extra

spaces. In order to join values, we need them to match perfectly. If there are any differences, R considers them to be different values.

To illustrate this, let's return to our mock patient data from the first lesson. If you recall, we had two dataframes, one called `demographic` and the other called `test_info`. We can recreate these datasets but change `Alice` to `alice` in the `demographic` dataset and keep all other values the same.

```
demographic <- tribble(
  ~name,     ~age,
  "alice",    25,
  "Bob",      32,
  "Charlie",  45,
)
demographic
```

```
test_info <- tribble(
  ~name,  ~test_date,    ~result,
  "Alice",        "2023-06-05",  "Negative",
  "Bob",          "2023-08-10",  "Positive",
  "Xavier",       "2023-05-02",  "Negative",
)
test_info
```

Now let's try an `inner_join()` on our two datasets.

```
inner_join(demographic, test_info, by="name")
```

As we can see, R didn't recognize `Alice` and `alice` as the same person, so the only common value between the datasets was `Bob`. How do we deal with this? Well, there are multiple functions we can use to modify our string values. In this case, using `str_to_title()` would work to ensure that all values are the same. If we apply this to our `name` column in our `demographic` dataset, we'll be able to join the tables appropriately.

```
demographic <- demographic %>%
  mutate(name=str_to_title(name))
demographic
```

```
inner_join(demographic, test_info, by="name")
```

That worked perfectly! We won't go into detail about all the different functions we can use to modify strings, since they're covered extensively in the strings lesson. The important part of this lesson is that we will learn how to identify mismatched values between dataframes.

The following two datasets contain data for India, Indonesia, and the Philippines. What are the differences between the values in the key columns that would have to be changed before joining the datasets?

```
df1 <- tribble(
  ~Country,      ~Capital,
  "India",       "New Delhi",
  "Indonesia",   "Jakarta",
  "Philippines", "Manila"
)

df2 <- tribble(
  ~Country,      ~Population,    ~Life_Expectancy,
  "India ",       1393000000,    69.7,
  "indonesia",    273500000,     71.7,
  "Philipines",   113000000,     72.7
)
```

In small datasets such as our mock data above, it's quite easy to notice the differences between values in our key columns. But what if we have much bigger datasets? To illustrate this, let's take a look at two real-world datasets on TB in India.

Our first dataset contains data on notification of TB cases in 2022 for all Indian states and Union Territories, taken from the Government of India Tubrculosis Report. Our variables include the state/Union Territory name, the type of healthcare system the patients were detected in (pulic or private), the target number of patients to be notified of their TB status, and the actual number of TB patients notified.

```
notification <- read_csv(here("data/notif_TB_india.csv"))
```

```
notification
```

```
## # A tibble: 5 × 4
##   state_UT                  hc_type target_notify
##   <chr>                     <chr>           <dbl>
## 1 Andaman & Nicobar Islands public            520
## 2 Andaman & Nicobar Islands private            10
## 3 Andhra Pradesh            public           85000
## 4 Andhra Pradesh            private          30000
## 5 Arunachal Pradesh         public            3450
##   actual_notified
##             <dbl>
## 1             510
## 2              24
## 3           62075
## 4           30112
## 5            2722
```

Our second dataset, taken from the same TB Report, contains the state/Union Territory name, the type of healthcare system, the number of TB patients screened for COVID, and the number of TB patients diagnosed with COVID.

```
covid <- read_csv(here("data/COVID_TB_india.csv"))
```

```
covid
```

```
## # A tibble: 5 × 4
##   state_UT                  hc_type covid_screened
##   <chr>                     <chr>            <dbl>
## 1 Andaman & Nicobar Islands public             322
## 2 Andaman & Nicobar Islands private              1
## 3 Andhra Pradesh            public           63319
## 4 Andhra Pradesh            private          26410
## 5 ArunachalPradesh          public            1761
##   covid_dx
##      <dbl>
## 1        0
## 2        0
## 3       97
## 4       17
## 5        0
```

For the sake of this lesson, we've modified some of the state/Union Territory names in the `covid` dataset. Our goal is to have them match the names from our `notification` dataset so to do this we need to compare values between them. For large dataframes, if we want to compare which values are in one but not the other, we can use the `setdiff()` function and state which dataframes & columns we want to compare. Let's start by comparing the `state_UT` values from `notification` dataframe to the `state_UT` values from the `covid` dataframe.

```
setdiff(notification$state_UT, covid$state_UT)
```

```
## [1] "Arunachal Pradesh"
## [2] "Dadra and Nagar Haveli and Daman and Diu"
## [3] "Tamil Nadu"
## [4] "Tripura"
```

So what does the list above tell us? Well by putting the `notification` dataset first, we are asking R "which values are in `notification` but *not* in `covid`?" We can (and should!) also switch the order of the datasets to check the reverse, asking "which values are in `covid` but not in `notification`?" Let's do this and compare the two lists.

```
setdiff(covid$state_UT, notification$state_UT)
```

```
## [1] "ArunachalPradesh"
## [2] "Dadra & Nagar Haveli and Daman & Diu"
## [3] "tamil nadu"
## [4] "Tri pura"
```

`covid` data using the `case_when()` function to have our two dataframes match. Let's clean this up and then compare our datasets again.

```
covid <- covid %>%
  mutate(state_UT =
           case_when(state_UT == "ArunachalPradesh" ~ "Arunachal Pradesh",
                     state_UT == "tamil nadu" ~ "Tamil Nadu",
                     state_UT == "Tri pura" ~ "Tripura",
                     state_UT == "Dadra & Nagar Haveli and Daman & Diu" ~
         "Dadra and Nagar Haveli and Daman and Diu",
                     TRUE ~ state_UT))

setdiff(notification$state_UT, covid$state_UT)
```

```
## character(0)
```

```
setdiff(covid$state_UT, notification$state_UT)
```

```
## character(0)
```

REMINDER

For the sake of illustration purposes, we've re-written the original values in our `covid` dataframe. However, in practice whenever you're transforming your variables, it's always best to create a new clean variable and drop old ones later if you no longer need them!

Great! As we can see, there are no longer differences in values between our dataframes. Now that we've ensured that our data is clean, we can move on to joining! Since we understand joining basics from our first lesson, we can move onto more complex topics.

The following dataframe, also taken from the TB Report, contains information on the number of pediatric TB cases and the number of pediatric patients initiated on treatment.

```
child <- read_csv(here("data/child_TB_india.csv"))
```

```
child
```

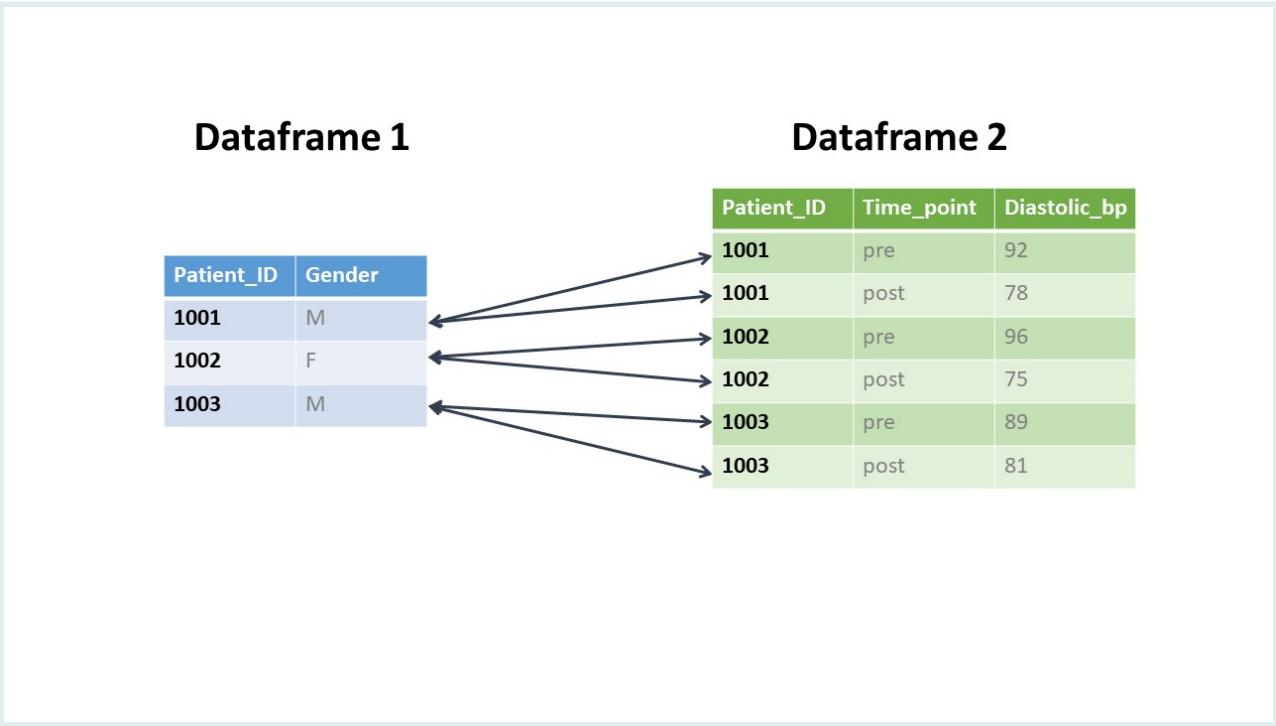
```
## # A tibble: 5 × 4
##   state_UT                  hc_type child_notify
##   <chr>                     <chr>          <dbl>
## 1 Andaman & Nicobar Islands public            18
## 2 Andaman & Nicobar Islands private            1
## 3 Andhra Pradesh            public          1347
## 4 Andhra Pradesh            private         1333
## 5 Arunachal Pradesh         public           256
```

```
##   child_treatment
##              <dbl>
## 1              19
## 2               0
## 3            1684
## 4             993
## 5             282
```

Using `set_diff()` compare the merging values from the `child` dataframe with those from the `notification` dataframe and make any necessary changes to the `child` dataframe to ensure that the values match.

One-to-many relationships

In the previous lesson, we looked at one-to-one joins, where an observation in one dataframe corresponded to no more than one observation in the other dataframe. In a one-to-many join, an observation one dataframe corresponds to multiple observations in the other dataframe. The image below illustrates this concept:



Let's take a look at a one-to-many join with a `left_join()` first!

`left_join()`

To illustrate a one-to-many join, let's return to our patients and their COVID test data. Let's imagine that in our dataset, `Alice` and `Xavier` got tested multiple times for COVID.

We can add two more rows to our `test_info` dataframe with their new test information:

```
test_info_many <- tribble(
  ~name,    ~test_date, ~result,
  "Alice",  "2023-06-05", "Negative",
  "Alice",  "2023-06-10", "Positive",
  "Bob",    "2023-08-10", "Positive",
  "Xavier", "2023-05-02", "Negative",
  "Xavier", "2023-05-12", "Negative",
)
```

Next, let's take a look at what happens when we use a `left_join()`, first with `demographic` as the dataset to the left of the call:

```
left_join(demographic, test_info_many)
```

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

What´s happened above? Well as we know, `Alice` was in our left dataframe so that row was retained. But she featured twice in the right dataset, so her demographic information was duplicated in the final dataset.`Xavier` wasn't in the left dataframe, so he was dropped entirely. Basically, when you perform a one-to-many join, the data from the "one" side are duplicated for each matching row of the "many" side. The graphic below illustrates this process:

We can see the same thing happen when the order of the datasets are switched, putting `test_info_many` to the left of the call..

```
left_join(test_info_many, demographic)
```

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

Once again, the demographic data for `Alice` has been duplicated! `Xavier` was in the left dataframe `test_info_many` so his rows were kept, but since he wasn't in the `demographic` dataframe, those cells are set to `NA`.

Copy the code below to create two small dataframes:

```
patient_info <- tribble(
  ~patient_id, ~name,      ~age,
  1,           "Liam",     32,
  2,           "Manny",    28,
  3,           "Nico",     40
)

conditions <- tribble(
  ~patient_id, ~disease,
  1,           "Diabetes",
  1,           "Hypertension",
  2,           "Asthma",
  3,           "High Cholesterol",
  3,           "Arthritis"
)
```

If you use a `left_join()` to join these datasets, how many rows will be in the final dataframe? Try to figure it out and then perform the join to see if you were right!

Let's apply this to our real-world datasets. The first dataset that we'll work with is the `notification` dataset. As reminder, here is what it looks like:

```
notification
```

```
## # A tibble: 5 × 4
##   state_UT                  hc_type target_notify
##   <chr>                     <chr>           <dbl>
## 1 Andaman & Nicobar Islands public            520
## 2 Andaman & Nicobar Islands private            10
## 3 Andhra Pradesh            public          85000
## 4 Andhra Pradesh            private         30000
## 5 Arunachal Pradesh         public           3450
##   actual_notified
##             <dbl>
## 1             510
## 2              24
## 3           62075
## 4           30112
## 5            2722
```

Our second dataset contains 32 of the 36 Indian state and Union Territories, as well as their subdivision category, and the zonal council they're situated in.

```
regions <- read_csv(here("data/region_data_india.csv"))
```

```
regions
```

```
## # A tibble: 5 × 3
##   zonal_council          subdivision_category
##   <chr>                  <chr>
## 1 No Zonal Council       Union Territory
## 2 North Eastern Council  State
## 3 North Eastern Council  State
## 4 Eastern Zonal Council  State
## 5 Northern Zonal Council Union Territory
##   state_UT
##   <chr>
## 1 Andaman & Nicobar Islands
## 2 Arunachal Pradesh
## 3 Assam
## 4 Bihar
## 5 Chandigarh
```

First, we can check if their are any differences between the datasets.

```
setdiff(notification$state_UT, regions$state_UT)
```

```
## [1] "Andhra Pradesh" "Chhattisgarh"   "Ladakh"
## [4] "Tamil Nadu"
```

```
setdiff(regions$state_UT, notification$state_UT)
```

```
## character(0)
```

As we can see, there are four states in the `notification` dataset that are not in the `regions` dataset. These aren't errors that need cleaning, rather we just don't have the complete information in our `regions` dataset. If we want to keep all the `notification` cases, we will put it in the left position of our join. Let's try it out now!

```
notif_regions <- notification %>%
  left_join(regions)
```

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

As expected, the data from the `regions` dataframe was duplicated for every matching value of the `notification` dataframe. For states that aren't in the `regions` dataset, such as `Andhra Pradesh`, the corresponding cells are set to `NA`.

Great! Now we know how to use a `left_join()` when joining datasets with a one-to-many match. Let's take a look at the differences and similarities with an `inner_join()`.

Using a `left_join()`, join the cleaned `child` TB dataset with the `regions` dataset whilst keeping all of the values from the `child` dataframe.

`inner_join()`

When using an `inner_join()` with a one-to-many relationship, the same principles apply as with a `left_join()`. To illustrate this, let's take a look at our COVID patient data and their test info.

```
demographic
```

```
test_info_many
```

Now, let's take a look at what happens when we use an `inner_join()` to join these two datasets.

```
inner_join(demographic, test_info_many)
```

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

With an `inner_join()`, the values that are common between the datasets are kept and those from the "one" side are duplicated for every row of the "many" side. Since `Alice` and `Bob` are common between the two datasets, those are the only ones that are kept. And since `Alice` is featured twice in the `test_info`, her row from the `demographic` dataset is duplicated!

Let's try it out with our `covid` TB dataset, and our `regions` dataset. As a reminder, here are our datasets.

```
covid
```

```
regions
```

As we saw previously, the regions dataset is missing 4 states/Union Territories, so we can expect them to be excluded from our final dataframe with an `inner_join()`. Let's create a new dataframe called `inner_covid_regions`.

```
inner_covid_regions <- covid %>%
  inner_join(regions)
```

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

```
inner_covid_regions
```

Great, that's exactly what we wanted!

Use `set_diff()` to compare values between the `child` and `regions` datasets. Then, use an `inner_join()` to join the two. How many observations are left?

Multiple key columns

Sometimes we have more than one column that uniquely identify the observations that we want to match on. For example, let's imagine that we have systolic and diastolic blood pressure measures for three patients before (pre) and after (post) taking a new blood pressure drug.

```
blood_pressure <- tribble(
  ~name,    ~time_point,  ~systolic,  ~diastolic,
  "David",  "pre",         139,        87,
  "David",  "post",        121,        82,
  "Eamon",  "pre",         137,        86,
  "Eamon",  "post",        128,        79,
  "Flavio", "pre",         137,        81,
  "Flavio", "post",        130,        73
)
blood_pressure
```

Now, let's imagine we have another dataset with the same 3 patients and their serum creatinine levels before and after taking the drug. Creatinine is a waste product that is normally processed by the kidneys. If creatinine levels in the blood increase, it can signify that the kidneys aren't functioning properly, which can be a side effect of blood pressure drugs.

```
kidney <- tribble(
  ~name,    ~time_point,  ~creatinine,
  "David",  "pre",         0.9,
  "David",  "post",        1.3,
  "Eamon",  "pre",         0.7,
  "Eamon",  "post",        0.8,
  "Flavio", "pre",         0.6,
  "Flavio", "post",        1.4
)
kidney
```

We want to join the two datasets so that each patient has two rows, one row for their blood pressure and creatinine levels before taking the drug, and one row for their blood pressure and creatinine levels after the drug. To do this, our first instinct may be to join on the patients name. Let's try it out and see what happens:

```
bp_kidney_dups <- blood_pressure %>%
  left_join(kidney, by="name")
```

```
## Warning in left_join(., kidney, by = "name"): Detected an unexpected many-
to-many relationship between
## `x` and `y`.
## ℹ Row 1 of `x` matches multiple rows in `y`.
## ℹ Row 1 of `y` matches multiple rows in `x`.
## ℹ If a many-to-many relationship is expected, set
##   `relationship = "many-to-many"` to silence this
##   warning.
```

```
bp_kidney_dups
```

As we can see, this isn't what we wanted at all! We can join on the patient names, but R gives a warning message that this is considered a "many-to-many" relationship because multiple rows in one dataframe correspond to multiple rows in the other dataframe, meaning we end up with 4 rows per patient. As a general rule,you should avoid many-to-many joins whenever possible! Also note that since we have two columns called `time_point` (one from each dataframe), these columns in the new dataframe are differentiated by `.x` and `.y`.

What we want to do is match on both `name` and `time_point`. To do this we have to specify to R that there are two columns to match on. In reality, this is very simple! All we have to do is use the `c()` function and specify both column names.

```
bp_kidney <- blood_pressure %>%
  left_join(kidney, by = c("name", "time_point"))
bp_kidney
```

That looks great! Now let's apply this to our real-world `notification` and `covid` datasets.

```
notification
```

```
covid
```

Let's think about how we want our final dataframe to look. We want to have two rows for each state, one with the TB notification and COVID data for the public sector, and one with the TB notification and COVID data for the private sector. That means we have to match on both `state_UT` and `hc_type`. Just as we did for the patient data, we have to specify both key values in the `by=` statement using `c()`. Let's try it out!

```
notif_covid <- notification %>%
  left_join(covid, by=c("state_UT", "hc_type"))
notif_covid
```

Great, that's exactly what we wanted!

Create a new dataframe called `all_tb_data` together the `notif_covid` dataset with the `child` dataset. Then, join this dataframe with the `regions` data for a final combined dataframe, ensuring that no TB data is lost.

Contributors

The following team members contributed to this lesson:

KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement

(make sure to update the contributor list accordingly!)