Joining 2: Mismatched Values, One-to-Many & Multi-Key Joins

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, countrycode)
```

Pre-join data cleaning: addressing data inconsistencies

A toy example

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 inconsistencies 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
```

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

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

```
## # A tibble: 3 × 3
##   name    test_date  result
##   <chr>   <chr>      <chr>
## 1 alice   2023-06-05 Negative
## 2 Bob     2023-08-10 Positive
## 3 charlie 2023-05-02 Negative
```

Now let's try a `left_join()` and `inner_join()` on our two datasets.

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

```
## # A tibble: 3 × 4
##   name      age test_date  result
##   <chr>   <dbl> <chr>      <chr>
## 1 Alice      25 <NA>       <NA>
## 2 Bob        32 2023-08-10 Positive
## 3 Charlie    45 <NA>       <NA>
```

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

```
## # A tibble: 1 × 4
##   name    age test_date  result
##   <chr> <dbl> <chr>      <chr>
## 1 Bob      32 2023-08-10 Positive
```

As we can see, R didn't recognize `Alice` and `alice` as the same person, and it also could not match `Charlie` and `charlie`. So in the `left_join()`, `Alice` and `Charlie` are left with NAs, and in the `inner_join()`, they are dropped.

How can we fix this? We need to ensure that the names in both datasets are in title case, with a capitalized first letter. For this we can use `str_to_title()`. Let's try it:

```r
test_info_title <- test_info %>%
  mutate(name = str_to_title(name)) # convert to title case
test_info_title
```

```
## # A tibble: 3 × 3
##   name    test_date  result
##   <chr>   <chr>      <chr>
## 1 Alice   2023-06-05 Negative
## 2 Bob     2023-08-10 Positive
## 3 Charlie 2023-05-02 Negative
```

```r
left_join(demographic, test_info_title, by = "name")
```

```
## # A tibble: 3 × 4
##   name      age test_date  result
##   <chr>   <dbl> <chr>      <chr>
## 1 Alice      25 2023-06-05 Negative
## 2 Bob        32 2023-08-10 Positive
## 3 Charlie    45 2023-05-02 Negative
```

```r
inner_join(demographic, test_info_title, by = "name")
```

```
## # A tibble: 3 × 4
##   name      age test_date  result
##   <chr>   <dbl> <chr>      <chr>
## 1 Alice      25 2023-06-05 Negative
## 2 Bob        32 2023-08-10 Positive
## 3 Charlie    45 2023-05-02 Negative
```

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

PRACTICE
(in RMD)

Q: Inner Join countries

The following two datasets contain data for India, Indonesia, and the Philippines. However an `inner_join()` of these datasets produces no output. What are the differences between the values in the key columns that would have to be changed before joining the datasets?

```r
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
)

inner_join(df1, df2)
```

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

```
## # A tibble: 0 × 4
## # ℹ 4 variables: Country <chr>, Capital <chr>,
## #   Population <dbl>, Life_Expectancy <dbl>
```

Real Data Example 1: Key Typos

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 TB notifications (TB cases or relapses) in 2022 for all 36 Indian states and Union Territories, taken from the Government of India Tubrculosis Report.

```
        tb_notifs <- read_csv(here("data/notif_TB_india_modified.csv"))

        tb_notifs_public <- tb_notifs %>%
          filter(hc_type == "public") %>% # we want only public systems for
now
          select(-hc_type)

        tb_notifs_public
```

```
## # A tibble: 5 × 2
##   state                      tb_notif_count
##   <chr>                               <dbl>
## 1 Andaman & Nicobar Islands             510
## 2 Andhra Pradesh                      62075
## 3 Arunachal Pradesh                    2722
## 4 Assam                               36801
## 5 Bihar                               79008
```

The variables are the state/Union Territory name and the number of TB notifications from 2022.

Our second dataset is on COVID screening among TB cases for 36 Indian states taken from the same TB Report:

```
        covid_screening <- read_csv(here("data/COVID_india_modified.csv"))

        covid_screening_public <- covid_screening %>%
          filter(hc_type == "public") %>% # we want only public systems for
now
          select(-hc_type)

        covid_screening_public
```

```
## # A tibble: 5 × 2
##   state                      tb_covid_pos
##   <chr>                             <dbl>
## 1 Andaman & Nicobar Islands             0
## 2 Andhra Pradesh                       97
## 3 ArunachalPradesh                      0
## 4 Assam                              31
## 5 Bihar                              78
```

It contains the state/Union Territory name and the number of TB patients who tested positive for COVID-19, `tb_covid_pos`. Note that there are some missing values in this dataset.

Now, we'd like to join these two datasets, to allow us to calculate the percentage of TB patients in each state who tested positive for COVID-19.

Let's give it a go using `inner_join()`:

```
tb_notifs_and_covid_screening <-
  inner_join(tb_notifs_public, covid_screening_public)
```

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

```
tb_notifs_and_covid_screening
```

```
## # A tibble: 5 × 3
##   state                     tb_notif_count tb_covid_pos
##   <chr>                              <dbl>        <dbl>
## 1 Andaman & Nicobar Islands            510            0
## 2 Andhra Pradesh                     62075           97
## 3 Assam                              36801           31
## 4 Bihar                              79008           78
## 5 Chandigarh                          5664            8
```

We can now perform the percentage calculation:

```
tb_notifs_and_covid_screening %>%
  mutate(pct_covid_pos = 100 * tb_covid_pos/tb_notif_count)
```

```
## # A tibble: 5 × 4
##   state            tb_notif_count tb_covid_pos pct_covid_pos
##   <chr>                     <dbl>        <dbl>         <dbl>
## 1 Andaman & Nicob…            510            0        0
## 2 Andhra Pradesh            62075           97        0.156
## 3 Assam                     36801           31        0.0842
## 4 Bihar                     79008           78        0.0987
## 5 Chandigarh                 5664            8        0.141
```

Seems alright!

However, notice that we now only have 32 rows in the output dataset, even though both initial datasets had 36 rows. Whenever you lose data in this way, it is worth investigating.

In this case, it turns out that there are several regions spelled differently in the two datasets. Because of these "key typos" could not be joined and were therefore dropped. How can we identify these to avoid information loss?

VOCAB

Recall that a "key" refers to the column(s) used to match rows across datasets in a join. The join matches rows that have identical values in the key columns.

Key typos are spelling or formatting inconsistencies in the values of key columns across datasets. For example, one dataset may list "New Delhi" while the other lists "Delhi". Because of these inconsistencies, rows that should match can't be joined properly and are dropped.

Identifying unmatched values with `setdiff()`

To identify these key typos, we can compare which values are in one data frame but not the other using the `setdiff()` function.

Let's start by comparing the `state` values from `tb_notifs_public` dataframe to the `state` values from the `covid_screening_public` dataframe.

```
setdiff(tb_notifs_public$state, covid_screening_public$state)
```

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

So what does the list above tell us? By putting the `tb_notifs_public` dataset first, we are asking R "which values are in `tb_notifs_public` but *not* in `covid_screening_public`?"

We can (and should!) also switch the order of the datasets to check the reverse, asking "which values are in `covid_screening_public` but not in `tb_notifs_public`?" Let's do this and compare the two lists.

```
setdiff(covid_screening_public$state, tb_notifs_public$state)
```

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

As we can see, there are four values in `covid_screening_public` that have spelling errors or are written differently than in `tb_notifs_public`. In this case, the easiest thing would be to clean our `covid_screening_public` data using the `case_when()` function to have our two dataframes match. Let's clean this up and then compare our datasets again.

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

        setdiff(tb_notifs_public$state, covid_screening_public_clean$state)
```

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

```
        setdiff(covid_screening_public_clean$state, tb_notifs_public$state)
```

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

Great! Now we have no more differences in the region's names.

We can now perform our join without unnecessary data loss:

```
        inner_join(tb_notifs_public, covid_screening_public_clean)
```

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

```
## # A tibble: 5 × 3
##   state                     tb_notif_count tb_covid_pos
##   <chr>                              <dbl>        <dbl>
## 1 Andaman & Nicobar Islands            510            0
## 2 Andhra Pradesh                     62075           97
## 3 Arunachal Pradesh                   2722            0
## 4 Assam                              36801           31
## 5 Bihar                              79008           78
```

All 36 rows are retained!

Identifying unmatched values with antijoin()

The anti_join() function in {dplyr} is a handy alternative to setdiff() for identifying discrepancies in key columns before joining two dataframes. It returns all rows from the first dataframe where the key values don't match the second dataframe.

For example, to find unmatched state values in tb_notifs_public when compared to covid_screening_public, you can use:

```
anti_join(tb_notifs_public, covid_screening_public)
```

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

```
## # A tibble: 4 × 2
##   state                                     tb_notif_count
##   <chr>                                              <dbl>
## 1 Arunachal Pradesh                                   2722
## 2 Dadra and Nagar Haveli and Daman and Diu            1294
## 3 Tamil Nadu                                         71896
## 4 Tripura                                             2865
```

And vice versa, to find the state values that are in covid_screening_public but not in tb_notifs_public, you can run:

```
anti_join(covid_screening_public, tb_notifs_public)
```

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

```
## # A tibble: 4 × 2
##   state                                   tb_covid_pos
##   <chr>                                          <dbl>
## 1 ArunachalPradesh                                   0
## 2 Dadra & Nagar Haveli and Daman & Diu               1
## 3 tamil nadu                                       178
## 4 Tri pura                                           2
```

This method returns the entire rows where discrepancies occur, providing more context and potentially making it easier to diagnose and fix the issues.

After identifying these, we can again fix the errors with mutate() and proceed with the join.

PRACTICE

Q: Check and fix typos before join

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_modified.csv"))

child_public <- child %>%
  filter(hc_type == "public") %>%
  select(-hc_type)

child_public
```

```
## # A tibble: 5 × 2
##   state                       tb_child_notifs
##   <chr>                                 <dbl>
## 1 Andaman & Nicobar Islands                18
## 2 Andhra Pradesh                         1347
## 3 ArunachalPradesh                        256
## 4 Assam                                   992
## 5 Bihar                                  4434
```

1. Using `set_diff()` or `anti_join()` compare the key values from the `child_public` dataframe with those from the `tb_notifs_public` dataframe, which was defined previously
2. Make any necessary changes to the `child_public` dataframe to ensure that the values match.
3. Join the two datasets.
4. Identify which two regions have the highest proportion of TB cases in children.

Real Data Example 2: Key Typos and Data Gaps

In a previous example, we saw how key typos—spelling and formatting inconsistencies—can prevent a successful join between datasets. Now, let's delve into a slightly more complex scenario.

We start with the `covid_screening_public` dataset that has 36 entries:

```
covid_screening_public
```

```
## # A tibble: 5 × 2
##   state                     tb_covid_pos
##   <chr>                            <dbl>
## 1 Andaman & Nicobar Islands            0
## 2 Andhra Pradesh                      97
## 3 ArunachalPradesh                     0
## 4 Assam                               31
## 5 Bihar                               78
```

We aim to enhance this dataset with zoning information available in another dataset, `regions`, which contains 32 entries:

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

```
## # A tibble: 5 × 3
##   zonal_council          subdivision_categ…¹ state_UT
##   <chr>                  <chr>               <chr>
## 1 No Zonal Council       Union Territory     Andaman & Nico…
## 2 North Eastern Council  State               Arunachal Prad…
## 3 North Eastern Council  State               Assam
## 4 Eastern Zonal Council  State               Bihar
## 5 Northern Zonal Council Union Territory     Chandigarh
## # ℹ abbreviated name: ¹subdivision_category
```

The `regions` dataset columns include `zonal_council`, `subdivision_category`, and `state_UT`, which correspond to the zonal council designations, category of subdivision, and the names of states or Union Territories, respectively.

To merge this zoning information without losing rows from our original `covid_screening_public` data, we opt for a left join:

```
covid_regions <- left_join(covid_screening_public,
                           regions,
                           by = c("state" = "state_UT"))
covid_regions
```

```
## # A tibble: 5 × 4
##   state                     tb_covid_pos zonal_council
##   <chr>                            <dbl> <chr>
## 1 Andaman & Nicobar Islands            0 No Zonal Council
## 2 Andhra Pradesh                      97 <NA>
## 3 ArunachalPradesh                     0 <NA>
## 4 Assam                               31 North Eastern Coun…
## 5 Bihar                               78 Eastern Zonal Coun…
## # ℹ 1 more variable: subdivision_category <chr>
```

After performing the left join, we can observe that 7 entries lack zoning information:

```
        covid_regions %>%
          filter(is.na(zonal_council))
```

```
## # A tibble: 5 × 4
##   state                             tb_covid_pos zonal_council
##   <chr>                                    <dbl> <chr>
## 1 Andhra Pradesh                              97 <NA>
## 2 ArunachalPradesh                             0 <NA>
## 3 Chhattisgarh                                57 <NA>
## 4 Dadra & Nagar Haveli and Daman…              1 <NA>
## 5 Ladakh                                       0 <NA>
## # ℹ 1 more variable: subdivision_category <chr>
```

To understand why, we investigate the discrepancies using the `anti_join()` function.

First, we check which states are present in the `regions` dataset but not in `covid_screening_public`:

```
        anti_join(regions, covid_screening_public, by = c("state_UT" =
"state"))
```

```
## # A tibble: 3 × 3
##   zonal_council          subdivision_category state_UT
##   <chr>                  <chr>                <chr>
## 1 North Eastern Council State                Arunachal Prad…
## 2 Western Zonal Council Union Territory      Dadra and Naga…
## 3 North Eastern Council State                Tripura
```

This operation reveals 3 states present in `regions` but not in `covid_screening_public`:

1. Arunachal Pradesh
2. Dadra and Nagar Haveli and Daman and Diu
3. Tripura

Then, we check which states are present in the `covid_screening_public` but not in `regions`:

```
        anti_join(covid_screening_public, regions, by = c("state" =
"state_UT"))
```

```
## # A tibble: 5 × 2
##   state                                  tb_covid_pos
##   <chr>                                         <dbl>
## 1 Andhra Pradesh                                   97
## 2 ArunachalPradesh                                  0
## 3 Chhattisgarh                                       57
## 4 Dadra & Nagar Haveli and Daman & Diu                1
## 5 Ladakh                                              0
```

There are 7 states in `covid_screening_public` that are not matched in the `regions` dataset. Upon closer inspection, we can see that only three of these mismatches are due to key typos:...

The remaining four states—Andhra Pradesh, Chhattisgarh, Ladakh, and Tamil Nadu—are simply absent from the `regions` dataset.

To address the typos, we apply corrections similar to those in the previous example:

```
        covid_screening_public_fixed <- covid_screening_public %>%
          mutate(state =
                   case_when(state == "ArunachalPradesh" ~ "Arunachal
Pradesh",
                             state == "Tri pura" ~ "Tripura",
                             state == "Dadra & Nagar Haveli and Daman & Diu" ~
"Dadra and Nagar Haveli and Daman and Diu",
                             TRUE ~ state))
```

After applying the fixes, we perform another left join:

```
        covid_regions_joined_fixed <- left_join(covid_screening_public_fixed,
                                        regions,
                                        by = c("state" = "state_UT"))
        covid_regions_joined_fixed
```

```
## # A tibble: 5 × 4
##   state                      tb_covid_pos zonal_council
##   <chr>                             <dbl> <chr>
## 1 Andaman & Nicobar Islands             0 No Zonal Council
## 2 Andhra Pradesh                       97 <NA>
## 3 Arunachal Pradesh                     0 North Eastern Coun…
## 4 Assam                                31 North Eastern Coun…
## 5 Bihar                                78 Eastern Zonal Coun…
## # ℹ 1 more variable: subdivision_category <chr>
```

A subsequent check confirms that only four entries remain without zoning information:

```
        covid_regions_joined_fixed %>%
          filter(is.na(zonal_council))
```

```
## # A tibble: 4 × 4
##   state          tb_covid_pos zonal_council
##   <chr>                 <dbl> <chr>
## 1 Andhra Pradesh           97 <NA>
## 2 Chhattisgarh             57 <NA>
## 3 Ladakh                    0 <NA>
## 4 tamil nadu                178 <NA>
## # ℹ 1 more variable: subdivision_category <chr>
```

These four regions were not included in the `regions` dataset, so there is no further action we can take at this point.

Through this example, we see the challenge of ensuring that no data is lost during joins, which becomes increasingly complex with larger datasets. To handle such issues, we may employ strategies such as manual data inspection and correction, or fuzzy matching for imperfect string comparisons, using tools like the `{fuzzyjoin}` package in R.

SIDE NOTE

Identifying and correcting typographical errors in large datasets to be joined is a non-trivial task. There is no fully automated method for cleaning such discrepancies, and often, fuzzy matching—joining datasets based on non-exact string matches—may be the practical solution. You can look into the {fuzzyjoin} R package for information on this.

PRACTICE

(in RMD)

Q: Merging TB Cases with Geographic Data

Run the code bellow to define two datasets.

The first, `top_tb_cases_kids` records the top 20 countries with the highest incidence of tuberculosis (TB) in children for the year 2012:

```
        top_tb_cases_kids <- tidyr::who %>%
          filter(year == 2012) %>%
          transmute(country, iso3, tb_cases_smear_0_14 =
new_sp_m014 + new_sp_f014) %>%
          arrange(desc(tb_cases_smear_0_14)) %>%
          head(20)

        top_tb_cases_kids
```

```
## # A tibble: 5 × 3
##   country                          iso3
tb_cases_smear_0_14
##   <chr>                            <chr>
<dbl>
## 1 India                            IND
12957
## 2 Pakistan                         PAK
3947
## 3 Democratic Republic of the Congo COD
```

```
3138
## 4 South Africa                          ZAF
2677
## 5 Indonesia                             IDN
1703
```

And `country_regions` lists countries along with their respective regions and continents:

```
country_regions <- countrycode::codelist %>%
  select(country_name = iso.name.en, iso3c, region) %>%
  filter(complete.cases(country_name, region))

country_regions
```

```
## # A tibble: 5 × 3
##   country_name   iso3c region
##   <chr>          <chr> <chr>
## 1 Afghanistan    AFG   South Asia
## 2 Albania        ALB   Europe & Central Asia
## 3 Algeria        DZA   Middle East & North Africa
## 4 American Samoa ASM   East Asia & Pacific
## 5 Andorra        AND   Europe & Central Asia
```

PRACTICE (in RMD)

Your task is to augment the TB cases data with the region and continent information without losing any relevant data.

1. Perform a `left_join` of `top_tb_cases_kids` with `country_regions` with the country names as the key. Identify which five countries fail to match correctly.

```
left_join(top_tb_cases_kids, __________________________,
by = ___________)
```

2. Using the code below, amend the country names in `top_tb_cases_kids` using `case_when` to rectify mismatches:

PRACTICE (in RMD)

```r
        top_tb_cases_kids_fixed <- top_tb_cases_kids %>%
          mutate(country = case_when(
            country == "Democratic Republic of the Congo" ~
"Congo, Democratic Republic of the",
            country == "Philippines" ~ "Philippines, The",
            country == "Democratic People's Republic of Korea"
~ "Korea, Democratic People's Republic of",
            country == "United Republic of Tanzania" ~
"Tanzania, United Republic of",
            country == "Cote d'Ivoire" ~ "Côte d'Ivoire",
            TRUE ~ country
          ))

        top_tb_cases_kids_fixed
```

```
## # A tibble: 20 × 3
##    country                           iso3  tb_cases_smear_…
¹
##    <chr>                             <chr>
<dbl>
##  1 India                             IND
12957
##  2 Pakistan                          PAK
3947
##  3 Congo, Democratic Republic of the COD
3138
##  4 South Africa                      ZAF
2677
##  5 Indonesia                         IDN
1703
##  6 Nigeria                           NGA
1187
##  7 China                             CHN
1091
##  8 Philippines, The                  PHL
1049
##  9 Kenya                             KEN
996
## 10 Angola                            AGO
982
## 11 Bangladesh                        BGD
966
## 12 Uganda                            UGA
636
## 13 Afghanistan                       AFG
588
## 14 Brazil                            BRA
580
## 15 Korea, Democratic People's Repub… PRK
520
## 16 Tanzania, United Republic of      TZA
493
```

```
## 18 Madagascar                               MDG
419
## 19 Côte d'Ivoire                            CIV
367
## 20 Myanmar                                  MMR
338
## # ℹ abbreviated name: ¹tb_cases_smear_0_14
```

Now attempt the join again using the revised dataset.

PRACTICE

(in RMD)

```
        left_join(top_tb_cases_kids_fixed,
_______________________, by = __________)
```

3. Try another `left_join`, but this time use the three-letter ISO code as the key. Do those initial five countries now align properly?

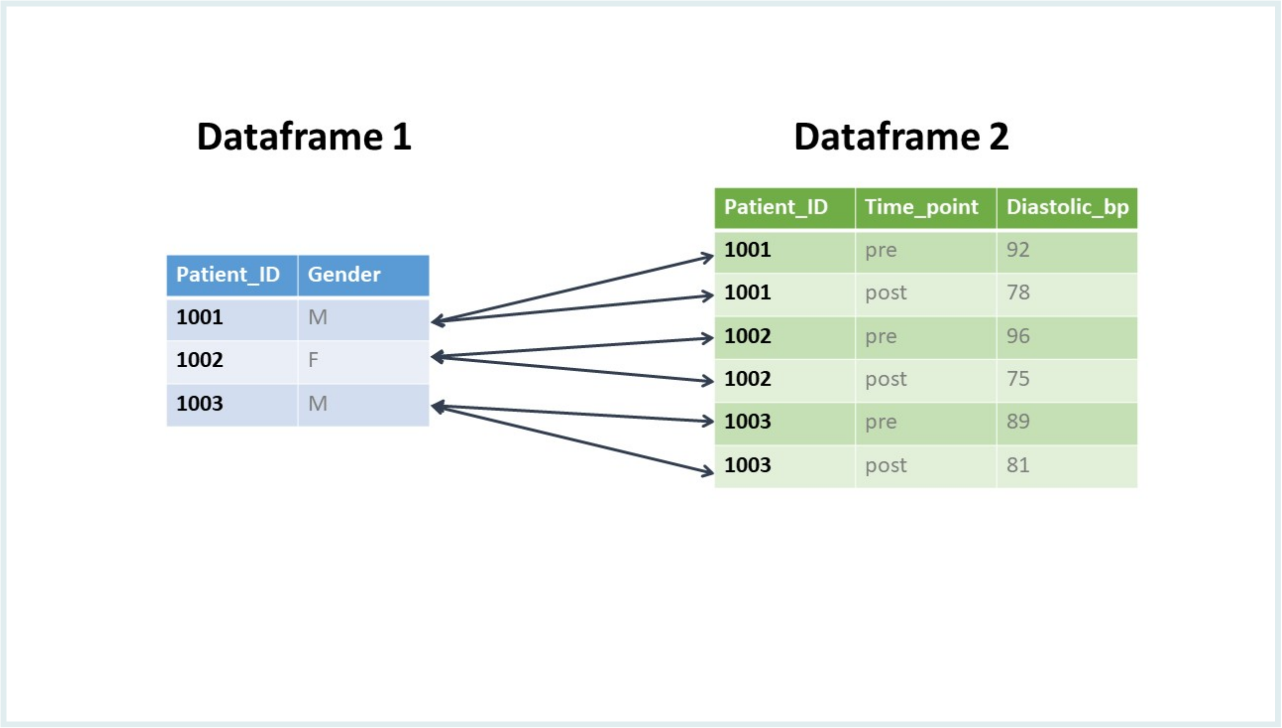
```
        left_join(top_tb_cases_kids, _______________________,
by = __________)
```

4. What is the advantage of utilizing ISO codes when recording and storing country information?

One-to-many relationships

So far, we have primarily looked at one-to-one joins, where an observation in one dataframe corresponded to only 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:



Patient_ID	Gender
1001	M
1002	F
1003	M

Patient_ID	Time_point	Diastolic_bp
1001	pre	92
1001	post	78
1002	pre	96
1002	post	75
1003	pre	89
1003	post	81

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()` with `demographic` as the dataset to the left of the call:

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

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

```
## # A tibble: 4 × 4
##   name      age test_date  result
##   <chr>   <dbl> <chr>      <chr>
## 1 Alice      25 2023-06-05 Negative
## 2 Alice      25 2023-06-10 Positive
## 3 Bob        32 2023-08-10 Positive
## 4 Charlie    45 <NA>       <NA>
```

What´s happened above? 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:

PRACTICE

(in RMD)

Q: Merging TB Cases with Geographic Data

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

PRACTICE (in RMD)

see if you were right!

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

```r
tb_notifs
```

```
## # A tibble: 5 × 3
##   state                     hc_type tb_notif_count
##   <chr>                     <chr>            <dbl>
## 1 Andaman & Nicobar Islands public             510
## 2 Andaman & Nicobar Islands private             24
## 3 Andhra Pradesh            public           62075
## 4 Andhra Pradesh            private          30112
## 5 Arunachal Pradesh         public            2722
```

Note that this dataset is a long dataset, with two rows per state, one for notifications from public health facilities in the state and one for private health facilities.

The second dataset is an Indian regions dataset:

```r
full_regions <- read_csv(here("data/region_data_india_full.csv"))
```

```
## Rows: 36 Columns: 3
## ── Column specification
────────────────────────────────────────────────────────────────────────────────────
## Delimiter: ","
## chr (3): zonal_council, subdivision_category, state_UT
##
## 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.
```

```r
full_regions
```

```
## # A tibble: 5 × 3
##   zonal_council          subdivision_categ…¹ state_UT
##   <chr>                  <chr>               <chr>
## 1 No Zonal Council       Union Territory     Andaman & Nico…
## 2 North Eastern Council  State               Arunachal Prad…
## 3 North Eastern Council  State               Assam
## 4 Eastern Zonal Council  State               Bihar
## 5 Northern Zonal Council Union Territory     Chandigarh
## # ℹ abbreviated name: ¹subdivision_category
```

Let's try joining the datasets:

```
notif_regions <- tb_notifs %>%
  left_join(regions, by = c("state" = "state_UT"))
notif_regions
```

```
## # A tibble: 5 × 5
##   state                hc_type tb_notif_count zonal_council
##   <chr>                <chr>            <dbl> <chr>
## 1 Andaman & Nicobar … public              510 No Zonal Counc…
## 2 Andaman & Nicobar … private              24 No Zonal Counc…
## 3 Andhra Pradesh      public            62075 <NA>
## 4 Andhra Pradesh      private           30112 <NA>
## 5 Arunachal Pradesh   public             2722 North Eastern …
## # ℹ 1 more variable: subdivision_category <chr>
```

As expected, the data from the `regions` dataframe was duplicated for every matching value of the `tb_notifs` dataframe.

PRACTICE

(in RMD)

Q: Joining `child` with `regions`

Join child with region

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

```
joined_dat <- left_join(____________, _______________,
by = ____________)
```

Then work out which Zonal council has the highest number of states/Union territories

```
joined_dat %>%
  count(_______________)
```

Multiple key columns

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

```
blood_pressure <- tribble(
  ~name,    ~time_point,  ~systolic,
  "Alice",   "pre",         139,
  "Alice",   "post",        121,
  "Bob",     "pre",         137,
  "Bob",     "post",        128,
  "Charlie", "pre",         137,
  "Charlie", "post",        130 )
```

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. High creatinine levels may be a side effect of the drug being tested.

```
kidney <- tribble(
  ~name,    ~time_point,  ~creatinine,
  "Alice",   "pre",         0.9,
  "Alice",   "post",        1.3,
  "Bob",     "pre",         0.7,
  "Bob",     "post",        0.8,
  "Charlie", "pre",         0.6,
  "Charlie", "post",        1.4
)
```

We want to join the two datasets so that each patient has two rows, one row for their levels before the drug and one row for their levels after. 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
```

```
## # A tibble: 5 × 5
##   name  time_point.x systolic time_point.y creatinine
##   <chr> <chr>           <dbl> <chr>             <dbl>
## 1 Alice pre               139 pre                 0.9
## 2 Alice pre               139 post                1.3
## 3 Alice post              121 pre                 0.9
## 4 Alice post              121 post                1.3
## 5 Bob   pre               137 pre                 0.7
```

As we can see, this isn't what we wanted at all! We end up with duplicated rows. Now we have FOUR rows for each person. And 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. 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
```

```
## # A tibble: 5 × 4
##   name    time_point systolic creatinine
##   <chr>   <chr>         <dbl>      <dbl>
## 1 Alice   pre             139        0.9
## 2 Alice   post            121        1.3
## 3 Bob     pre             137        0.7
## 4 Bob     post            128        0.8
## 5 Charlie pre             137        0.6
```

SIDE NOTE

Note that specifying `by = c("name", "time_point")` is different from using a named vector in the form of `by = c("keya" = "keyb")`. The former matches the columns by name across both datasets, while the latter is used to match columns with different names between two datasets.

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

```
tb_notifs
```

```
## # A tibble: 5 × 3
##   state                       hc_type tb_notif_count
##   <chr>                       <chr>            <dbl>
## 1 Andaman & Nicobar Islands public             510
## 2 Andaman & Nicobar Islands private             24
## 3 Andhra Pradesh            public           62075
## 4 Andhra Pradesh            private          30112
## 5 Arunachal Pradesh         public            2722
```

```
covid_screening
```

```
## # A tibble: 5 × 3
##   state                       hc_type tb_covid_pos
##   <chr>                       <chr>          <dbl>
## 1 Andaman & Nicobar Islands public             0
## 2 Andaman & Nicobar Islands private            0
## 3 Andhra Pradesh            public            97
## 4 Andhra Pradesh            private           17
## 5 ArunachalPradesh          public             0
```

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` 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 <- tb_notifs %>%
  left_join(covid_screening, by=c("state", "hc_type"))
notif_covid
```

```
## # A tibble: 5 × 4
##   state                  hc_type tb_notif_count tb_covid_pos
##   <chr>                  <chr>            <dbl>        <dbl>
## 1 Andaman & Nicobar Isl… public             510            0
## 2 Andaman & Nicobar Isl… private             24            0
## 3 Andhra Pradesh         public           62075           97
## 4 Andhra Pradesh         private          30112           17
## 5 Arunachal Pradesh      public            2722           NA
```

Great, that's exactly what we wanted!

PRACTICE

(in RMD)

PRACTICE (in RMD)

Q: Joining three datasets, including one-to-many

Join the following three datasets: `notif_covid`, `child` and `regions`, ensuring that no data is lost from any of the datasets.

Wrap-up

In this lesson, we delved into the intricacies of data cleaning before a join, focusing on how to detect and correct mismatches or inconsistencies in key columns. We also highlighted the impact of one-to-many relationships in joining dataframes, showing how data from the "one" side is duplicated for each matching row of the "many" side. Finally, we demonstrated how to join dataframes using multiple key columns.

As we conclude this lesson, we hope that you have gained a deeper understanding of the importance and utility of joining dataframes in R.

Solutions

Contributors

The following team members contributed to this lesson:

AMANDA MCKINLEY

R Developer and Instructor, the GRAPH Network

KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement

(make sure to update the contributor list accordingly!)