R: Tidy Data & Data Import 3

Sakol Suethanapornkul

11 January 2021

Preamble

So far, we have worked with various packages in `tidyverse`, from `dplyr`—through `ggplot2`—to `readr`. Through these packages, we have learned how to import into R, subset and manipulate, and then visualize data.

Let's go through the exercise we saw at the end of the previous class once again to refresh our memory. . .

```
library(tidyverse)
```

Exercise

Let's return to Income.csv. We will work through this exercise slowly.

[1] Import Income.csv and name the data frame as `income`.

Exercise

Let's return to Income.csv. We will work through this exercise slowly.

[1] Import Income.csv and name the data frame as `income`.

```
income <- read_csv(file = "./Data/Income.csv")
```

Exercise

Let's return to Income.csv. We will work through this exercise slowly.

[1] Import Income.csv and name the data frame as `income`.
[2] Create a new data frame consisting of country names and years since 1980.
[3] Select China, Japan, Malaysia, Singapore, and Thailand.

Exercise

Let's return to Income.csv. We will work through this exercise slowly.

[1] Import Income.csv and name the data frame as `income`.
[2] Create a new data frame consisting of country names and years since 1980.
[3] Select China, Japan, Malaysia, Singapore, and Thailand.

```
income_sm <- income %>%
  select(!c(`1800`:`1979`)
         ) %>%
  filter(country %in% c("China", "Japan", "Malaysia",
                        "Singapore", "Thailand")
         )
```

NOTE: Since our column names do not begin with letters, we need back ticks (` `).

Exercise

Let's return to Income.csv. We will work through this exercise slowly.

[1] Import Income.csv and name the data frame as `income`.
[2] Create a new data frame consisting of country names and years since 1980.
[3] Select China, Japan, Malaysia, Singapore, and Thailand.
[4] Pivot columns of years to a new variable "`year`" and values of GDP to "`gdp`"

Exercise

Let's return to Income.csv. We will work through this exercise slowly.

[1] Import Income.csv and name the data frame as `income`.
[2] Create a new data frame consisting of country names and years since 1980.
[3] Select China, Japan, Malaysia, Singapore, and Thailand.
[4] Pivot columns of years to a new variable "`year`" and values of GDP to "`gdp`"

```
income_ln <- income_sm %>%
  pivot_longer(cols = !country,
               names_to = "year", values_to = "gdp")
```

Quick review

Recall that `pivot_longer` and `pivot_wider` are asymmetrical. Note how you select columns with these two functions.

1) `pivot_longer`:

```
income_sm %>%
  pivot_longer(cols = !country,
               names_to = "year", values_to = "gdp")
```

2) `pivot_wider`:

```
income_ln %>%
  pivot_wider(names_from = year, values_from = gdp)
```

Part I: `bind_rows()` and `bind_cols()`

One important aspect we have not yet discussed is how to combine several data frames together!

Before we do that, let's create two tibbles, one for each "class". We will have three columns: `id`, `class` and `height`. Can you remember how to create tibbles?

Part I: bind_rows() and bind_cols()

Here is our tibble for class A:

```
cl_a <- tibble(id     = 1:5,
               class  = rep("A", times = 5),
               height = c(173, 176, 177, 180, 182)
               )
cl_a
```

```
## # A tibble: 5 x 3
##      id class height
##   <int> <chr>  <dbl>
## 1     1 A        173
## 2     2 A        176
## 3     3 A        177
## 4     4 A        180
## 5     5 A        182
```

Part I: bind_rows() and bind_cols()

And this is for class B:

```
cl_b <- tibble(id     = 10:14,
               class  = rep("B", times = 5),
               height = c(167, 171, 172, 176, 178)
               )
cl_b
```

```
## # A tibble: 5 x 3
##      id class height
##   <int> <chr>  <dbl>
## 1    10 B        167
## 2    11 B        171
## 3    12 B        172
## 4    13 B        176
## 5    14 B        178
```

Part I: `bind_rows()` and `bind_cols()`

Our two tibbles have the exact same columns (i.e., same column names). Combining them is easy, with `bind_rows()`:

```
class_tot <-  bind_rows(cl_a, cl_b)
```

Or we could "rewrite" our existing tibbles with:

```
cl_a <-  bind_rows(cl_a, cl_b)
```

Part I: bind_rows() and bind_cols()

Suppose our class A has another column gender which represents the gender of students:

```
cl_a <- cl_a %>%
  mutate(gender = c("M", "F", "F", "F", "M") )

cl_a
```

```
## # A tibble: 5 x 4
##      id class height gender
##   <int> <chr>  <dbl> <chr>
## 1     1 A        173 M
## 2     2 A        176 F
## 3     3 A        177 F
## 4     4 A        180 F
## 5     5 A        182 M
```

Part I: `bind_rows()` and `bind_cols()`

EXERCISE: What will happen if we combine the first class (i.e., `cl_a`) with the second class (i.e., `cl_b`)?

Part I: bind_rows() and bind_cols()

ANSWER: An empty cell in the column gender will be given NA.

```
cl_a <- bind_rows(cl_a, cl_b)

head(cl_a, n = 7)
```

```
## # A tibble: 7 x 4
##      id class height gender
##   <int> <chr>  <dbl> <chr>
## 1     1 A        173 M
## 2     2 A        176 F
## 3     3 A        177 F
## 4     4 A        180 F
## 5     5 A        182 M
## 6    10 B        167 <NA>
## 7    11 B        171 <NA>
```

Part I: `bind_rows()` and `bind_cols()`

But it's very much possible that our original tibbles may not have an "identification." For example:

Class A:

```
cl_a <- tibble(id     = 1:5,
               height = c(173, 176, 177, 180, 182) )
```

Class B:

```
cl_b <- tibble(id     = 10:14,
               height = c(167, 171, 172, 176, 178) )
```

EXERCISE: How can we combine them such that we know which class each part of a new tibble come from?

Part I: bind_rows() and bind_cols()

ANSWER: We can name a new column with .id:

```
cl <- bind_rows(.id = "class", "A" = cl_a, "B" = cl_b)

head(cl, n = 7)
```

```
## # A tibble: 7 x 3
##   class    id height
##   <chr> <int>  <dbl>
## 1 A         1    173
## 2 A         2    176
## 3 A         3    177
## 4 A         4    180
## 5 A         5    182
## 6 B        10    167
## 7 B        11    171
```

Part I: bind_rows() and bind_cols()

While `bind_rows()` binds tibbles by rows, `bind_cols()` does so by columns.

Let's look at an example.

```
cl_a1 <- tibble(id     = 1:5,
                class  = rep("A", times = 5),
                height = c(173, 176, 177, 180, 182)
                )

cl_a2 <- tibble(gender = c("M", "F", "F", "F", "M"),
                grade  = c(3.3, 3.4, 3.5, 3.8, 3.9)
                )
```

Part I: bind_rows() and bind_cols()

In bind_cols(), we provide as arguments tibble names:

```
bind_cols(cl_a1, cl_a2)
```

```
## # A tibble: 5 x 5
##      id class height gender grade
##   <int> <chr>  <dbl> <chr>  <dbl>
## 1     1 A        173 M        3.3
## 2     2 A        176 F        3.4
## 3     3 A        177 F        3.5
## 4     4 A        180 F        3.8
## 5     5 A        182 M        3.9
```

Part I: bind_rows() and bind_cols()

But bind_cols() can behave unexpectedly. Let's say we have the following tibbles:

```
cl_a1 <- tibble(id     = 1:5,
                class  = rep("A", times = 5),
                height = c(173, 176, 177, 180, 182)
                )

cl_a2 <- tibble(id     = 1:5,
                gender = c("M", "F", "F", "F", "M"),
                grade  = c(3.3, 3.4, 3.5, 3.8, 3.9)
                )
```

Part I: `bind_rows()` and `bind_cols()`

Go ahead and combine the two tibbles by columns:

```
bind_cols(cl_a1, cl_a2)
```

Let's apply what we have learned from `bind_rows()`. Do following codes fix the problem?

```
bind_cols(.id = "id_new", cl_a1, cl_a2)

bind_cols(.id = "id_new", "1" = cl_a1, "2" = cl_a2)
```

Part II: `*_join()`

It's time we talked about some other important functions that help us deal with **relational data** in a more sophisticated way.

To start, we look back at the two tibbles:

Part II: *_join()

```
head(cl_a1, n = 3)
```

```
## # A tibble: 3 x 3
##      id class height
##   <int> <chr>  <dbl>
## 1     1 A        173
## 2     2 A        176
## 3     3 A        177
```

```
head(cl_a2, n = 3)
```

```
## # A tibble: 3 x 3
##      id gender grade
##   <int> <chr>  <dbl>
## 1     1 M        3.3
## 2     2 F        3.4
## 3     3 F        3.5
```

Part II: `*_join()`

Even with this simple example, we can see one critical element of **relational data**: `id` links the two tibbles together. This element is referred to as a "key."

Part II: *_join()

`id` serves as a **primary key** that identifies unique observations in `cl_a1`. It also is a **foreign key** to `cl_a2` linking the two tibbles together.

We can use `left_join()` to combine the two tibbles together:

```
left_join(x, y,
          by = " ",
          suffix = c(" ", " "))
```

Part II: *_join()

So, to join the two tibbles together, we use:

```
left_join(cl_a1, cl_a2, by = "id")
```

```
## # A tibble: 5 x 5
##      id class height gender grade
##   <int> <chr>  <dbl> <chr>  <dbl>
## 1     1 A        173 M        3.3
## 2     2 A        176 F        3.4
## 3     3 A        177 F        3.5
## 4     4 A        180 F        3.8
## 5     5 A        182 M        3.9
```

Part II: *_join()

If you want to "rewrite" cl_a1, you can do that with:

```
cl_a1 <- cl_a1 %>%
  left_join(cl_a2, by = "id")

cl_a1
```

```
## # A tibble: 5 x 5
##      id class height gender grade
##   <int> <chr>  <dbl> <chr>  <dbl>
## 1     1 A        173 M        3.3
## 2     2 A        176 F        3.4
## 3     3 A        177 F        3.5
## 4     4 A        180 F        3.8
## 5     5 A        182 M        3.9
```

Part II: *_join()

`left_join()` keeps **all observations in x** (which is `cl_a1` in our case). Let's say our other tibble has information from students no. 4 to 8.

```
cl_a2 <- tibble(id     = 4:8,
                gender = c("M", "F", "F", "F", "M"),
                grade  = c(3.3, 3.4, 3.5, 3.8, 3.9)
                )
```

EXERCISE: What will happen when we run the following code?

```
left_join(cl_a1, cl_a2, by = "id")
```

Part II: *_join()

The opposite of `left_join()` is `right_join()` which keeps **all observations in y**.

```
right_join(cl_a1, cl_a2, by = "id")
```

```
## # A tibble: 5 x 5
##      id class height gender grade
##   <int> <chr>  <dbl> <chr>  <dbl>
## 1     4 A        180 M        3.3
## 2     5 A        182 F        3.4
## 3     6 <NA>      NA F        3.5
## 4     7 <NA>      NA F        3.8
## 5     8 <NA>      NA M        3.9
```

Part II: *_join()

There is another kind of mutating joins, which is `inner_join()`. It keeps only observations that both tibbles share:

```
inner_join(cl_a1, cl_a2, by = "id")
```

```
## # A tibble: 2 x 5
##      id class height gender grade
##   <int> <chr>  <dbl> <chr>  <dbl>
## 1     4 A        180 M        3.3
## 2     5 A        182 F        3.4
```

Part II: *_join()

The opposite of `inner_join()` is `full_join()` which keeps **all observations in x and y**.

```
full_join(cl_a1, cl_a2, by = "id")
```

```
## # A tibble: 8 x 5
##      id class height gender grade
##   <int> <chr>  <dbl> <chr>  <dbl>
## 1     1 A        173 <NA>    NA
## 2     2 A        176 <NA>    NA
## 3     3 A        177 <NA>    NA
## 4     4 A        180 M        3.3
## 5     5 A        182 F        3.4
## 6     6 <NA>      NA F        3.5
## 7     7 <NA>      NA F        3.8
## 8     8 <NA>      NA M        3.9
```

Part II: *_join()

In many cases, you may have duplicate keys. For instance,

```
x <- tribble(
  ~key, ~val_x,
     1, "x1",
     2, "x2",
     2, "x3",
     1, "x4"
  )

y <- tribble(
  ~key, ~val_y,
     1, "y1",
     2, "y2"
  )
```

Part II: *_join()

Don't worry. When you join these tibbles (i.e., x and y) together, one is "expanded":

```
left_join(x, y, by = "key")
```

```
## # A tibble: 4 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     2 x3    y2
## 4     1 x4    y1
```

Q & A