Introduction to R for Data Analysis in the Health Sciences: Lecture 7

Amy Willis, Biostatistics, UW

15 November, 2019

Today

Advanced data manipulation

- Reordering data
- Joining multiple datasets
  - Adding columns
  - Adding rows
  - Removing duplicates
  - More complicated joining setups
- Reshaping data

As always. . .

```r
library(tidyverse)
```

```
## -- Attaching packages -----------------------------

## v ggplot2 3.1.0     v purrr   0.3.0
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.2.0

## -- Conflicts ------------------------------------ tidy
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Running example: FEV

```
fev <- read_csv("datasets/fev.csv")
```

```
## Parsed with column specification:
## cols(
##   seqnbr = col_integer(),
##   subjid = col_integer(),
##   age = col_integer(),
##   fev = col_double(),
##   height = col_double(),
##   sex = col_integer(),
##   smoke = col_integer()
## )
```

Running example: FEV

```r
head(fev)
```

```
## # A tibble: 6 x 7
##   seqnbr subjid   age   fev height   sex smoke
##    <int>  <int> <int> <dbl>  <dbl> <int> <int>
## 1      1    301     9  1.71   57       0     0
## 2      2    451     8  1.72   67.5     0     0
## 3      3    501     7  1.72   54.5     0     0
## 4      4    642     9  1.56   53       1     0
## 5      5    901     9  1.90   57       1     0
## 6      6   1701     8  2.34   61       0     0
```

Reordering data

Use arrange() to sort the values of a variable in increasing order

```
fev %>% arrange(age)
```

```
## # A tibble: 654 x 7
##    seqnbr subjid   age   fev height   sex smoke
##     <int>  <int> <int> <dbl>  <dbl> <int> <int>
##  1     26   5642     3 1.40    51.5     1     0
##  2    222  50951     3 1.07    46       0     0
##  3     23   5152     4 0.839   48       0     0
##  4     59  13751     4 1.57    50       0     0
##  5     64  14252     4 1.58    49       0     0
##  6    104  23841     4 0.796   47       1     0
##  7    173  40541     4 1.79    52       1     0
##  8    216  49551     4 1.10    48       0     0
##  9    233  54751     4 1.39    48       0     0
## 10    286  75951     4 1.42    49       0     0
## # ... with 644 more rows
```

Reordering data

Use `arrange(desc())` function to sort the values of a variable in decreasing order

```
fev %>% arrange(desc(age)) # sort in decreasing order
```

```
## # A tibble: 654 x 7
##     seqnbr subjid   age   fev height   sex smoke
##      <int>  <int> <int> <dbl>  <dbl> <int> <int>
##  1     609   6144    19  5.10   72       1     0
##  2     610   6252    19  3.52   66       0     1
##  3     618  21351    19  3.34   65.5     0     1
##  4     608   4051    18  2.91   66       0     0
##  5     619  22251    18  3.08   64.5     0     0
##  6     626  30441    18  4.22   68       1     0
##  7     638  48141    18  4.09   67       1     1
##  8     645  59944    18  4.40   70.5     1     1
##  9     652  73751    18  2.85   60       0     0
## 10     612   7142    17  4.43   70       1     0
```

Reordering data

Multiple criteria can be used for reordering:

```r
fev %>% arrange(age, height)
```

```
## # A tibble: 654 x 7
##    seqnbr subjid   age   fev height   sex smoke
##     <int>  <int> <int> <dbl>  <dbl> <int> <int>
##  1    222  50951     3 1.07    46       0     0
##  2     26   5642     3 1.40    51.5     1     0
##  3    104  23841     4 0.796   47       1     0
##  4     23   5152     4 0.839   48       0     0
##  5    216  49551     4 1.10    48       0     0
##  6    233  54751     4 1.39    48       0     0
##  7    299  80841     4 1.00    48       1     0
##  8     64  14252     4 1.58    49       0     0
##  9    286  75951     4 1.42    49       0     0
## 10     59  13751     4 1.57    50       0     0
## # ... with 644 more rows
```

Observations are now sorted first by age, then by height within age.

Joining data from multiple sources

It was very convenient that we received all of this information in one dataset! But it doesn’t always work out this way. . .

Joining data: Binding rows together

Suppose we have two datasets: all of the children under 13 y.o. in `fev_under_13.csv`, and all of the children 13+ y.o. in `fev_13_and_over.csv`.

How do we combine these datasets into a single file?

Joining data: Binding rows together

First, read in the data:

```
fev_younger <- read_csv("datasets/fev_under_13.csv")
```

```
## Parsed with column specification:
## cols(
##   seqnbr = col_integer(),
##   subjid = col_integer(),
##   age = col_integer(),
##   fev = col_double(),
##   height = col_double(),
##   sex = col_integer(),
##   smoke = col_integer()
## )
```

Joining data: Binding rows together

And again for the other dataset:

```r
fev_older <- read_csv("datasets/fev_13_and_over.csv")
```

```
## Parsed with column specification:
## cols(
##   seqnbr = col_integer(),
##   subjid = col_integer(),
##   age = col_integer(),
##   fev = col_double(),
##   height = col_double(),
##   sex = col_integer(),
##   smoke = col_integer()
## )
```

Joining data: Binding rows together

To combine datasets with some overlapping columns, use `bind_rows()`. This combines them end-to-end:

```
fev_combined <- bind_rows(fev_younger, fev_older)
fev_combined
```

```
## # A tibble: 654 x 7
##    seqnbr subjid   age   fev height   sex smoke
##     <int>  <int> <int> <dbl>  <dbl> <int> <int>
##  1      1    301     9  1.71   57       0     0
##  2      2    451     8  1.72   67.5     0     0
##  3      3    501     7  1.72   54.5     0     0
##  4      4    642     9  1.56   53       1     0
##  5      5    901     9  1.90   57       1     0
##  6      6   1701     8  2.34   61       0     0
##  7      7   1752     6  1.92   58       0     0
##  8      8   1753     6  1.42   56       0     0
##  9      9   1901     8  1.99   58.5     0     0
## 10     10   1951     9  1.94   60       0     0
## # ... with 644 more rows
```

Joining data: Binding rows together

If you have used `R` before, you may be familiar with `rbind`. `rbind` is a predecessor of `bind_rows()`.

`bind_rows()` is more flexible than `rbind`. `bind_rows()` makes more sensible assumptions about what to do when the columns don't match. . .

Joining data: Binding rows together

Suppose we didn't have smoking information for the younger children, and we didn't have height for the older children:

```
bind_rows(fev_younger %>% select(-smoke) %>% head(3),
          fev_older %>% select(-height) %>% head(3))
```

```
## # A tibble: 6 x 7
##   seqnbr subjid   age   fev height   sex smoke
##    <int>  <int> <int> <dbl>  <dbl> <int> <int>
## 1      1    301     9  1.71   57       0    NA
## 2      2    451     8  1.72   67.5     0    NA
## 3      3    501     7  1.72   54.5     0    NA
## 4    312    341    14  3.38   NA       1     0
## 5    319   2041    14  3.74   NA       1     0
## 6    320   2042    13  4.34   NA       1     0
```

If the columns don't match, `NA` will be used to fill in the absent column.

Joining data: Binding rows together

`rbind` returns an error when tasked with the same problem:

```
rbind(fev_younger %>% select(-smoke) %>% head(3),
      fev_older %>% select(-height) %>% head(3))
```

```
## Error in match.names(clabs, names(xi)): names do not mat
```

Joining data: removing duplicates

What we do if some of the observations are contained in both datasets?

```
fev_13_plus <- read_csv("datasets/fev_13_and_over.csv")
fev_13_minus <- read_csv("datasets/fev_13_and_under.csv")
all_fev <- bind_rows(fev_13_minus,
                     fev_13_plus)
```

`all_fev` has all of the 13 y.o. children recorded twice!

Joining data: removing duplicates

You can use `distinct()` to keep only the unique rows

```
fev_distinct <- all_fev %>% distinct
fev_distinct
```

```
## # A tibble: 654 x 7
##    seqnbr subjid   age   fev height   sex smoke
##     <int>  <int> <int> <dbl>  <dbl> <int> <int>
##  1      1    301     9  1.71   57       0     0
##  2      2    451     8  1.72   67.5     0     0
##  3      3    501     7  1.72   54.5     0     0
##  4      4    642     9  1.56   53       1     0
##  5      5    901     9  1.90   57       1     0
##  6      6   1701     8  2.34   61       0     0
##  7      7   1752     6  1.92   58       0     0
##  8      8   1753     6  1.42   56       0     0
##  9      9   1901     8  1.99   58.5     0     0
## 10     10   1951     9  1.94   60       0     0
## # ... with 644 more rows
```

You can also use the function `unique`, but `distinct` is ~5x faster

Joining data: removing duplicates

Be careful! If you do not have subject-specific identification numbers, you may end up accidentally removing different observations that have the same values for the variables.

e.g. There are two non-smoking 8 y.o. males children whose FEV is 2.631L and are 59cm tall in this dataset:

```
fev_distinct %>% filter(subjid %in% c(39901, 71401))
```

```
## # A tibble: 2 x 7
##   seqnbr subjid   age   fev height   sex smoke
##    <int>  <int> <int> <dbl>  <dbl> <int> <int>
## 1    169  39901     8  2.63     59     1     0
## 2    265  71401     8  2.63     59     1     0
```

Joining data: Binding columns together

You can also bind datasets together side-to-side with `bind_cols()`:

```
bind_cols(fev[,1:4],
          fev[,c(2,5:7)])
```

```
## # A tibble: 654 x 8
##    seqnbr subjid   age   fev subjid1 height   sex smoke
##     <int>  <int> <int> <dbl>   <int>  <dbl> <int> <int>
##  1      1    301     9  1.71     301   57       0     0
##  2      2    451     8  1.72     451   67.5     0     0
##  3      3    501     7  1.72     501   54.5     0     0
##  4      4    642     9  1.56     642   53       1     0
##  5      5    901     9  1.90     901   57       1     0
##  6      6   1701     8  2.34    1701   61       0     0
##  7      7   1752     6  1.92    1752   58       0     0
##  8      8   1753     6  1.42    1753   56       0     0
##  9      9   1901     8  1.99    1901   58.5     0     0
## 10     10   1951     9  1.94    1951   60       0     0
## # ... with 644 more rows
```

Joining data

Common challenges encountered when joining data (and why I rarely use `bind_cols()`):

- Observations may be in a different order
- Observations may be missing from one dataset
- Observations may have multiple records in one data set and not the other

Fortunately, there exist functions to join multiple datasets that deal with some of these issues!

Joining data

Suppose there is another dataset with IDs and race information. We want to add information on race to the `fev` tibble.

```
fev_race <- read_csv("datasets/fev_race.csv")
```

```
## Parsed with column specification:
## cols(
##   ID = col_integer(),
##   Race = col_character()
## )
```

Joining data

In fev we have subject ID in subjid, but in fev_race we have subject ID in column ID

```r
fev_race %>% arrange(ID)
```

```
## # A tibble: 654 x 2
##        ID Race
##     <int> <chr>
##  1   201 White
##  2   202 White
##  3   301 White
##  4   341 White
##  5   351 Non-white
##  6   401 White
##  7   441 Non-white
##  8   451 White
##  9   501 White
## 10   551 White
## # ... with 644 more rows
```

Joining data

Use `full_join` and specify the columns to be used for joining with the by argument:

```
fev_augmented <- full_join(fev,
                           fev_race,
                           by = c("subjid" = "ID"))
fev_augmented %>% arrange(subjid)
```

```
## # A tibble: 654 x 8
##    seqnbr subjid   age   fev height   sex smoke Race
##     <int>  <int> <int> <dbl>  <dbl> <int> <int> <chr>
##  1    310    201    11  2.88   69       1     0 White
##  2    311    202    10  2.33   64       1     0 White
##  3      1    301     9  1.71   57       0     0 White
##  4    312    341    14  3.38   63       1     0 White
##  5    313    351    11  2.17   58       0     0 Non-white
##  6    314    401    11  3.47   66.5     1     0 White
##  7    606    441    15  4.28   70       1     0 Non-white
##  8      2    451     8  1.72   67.5     0     0 White
##  9      3    501     7  1.72   54.5     0     0 White
## 10    315    551    12  3.06   60.5     0     0 White
## # ... with 644 more rows
```

Joining data

There are different ways to join datasets when there is imperfect overlap:

- Keep rows from the first argument
  - `left_join`: return all rows from `input1` and put `NA` if there is no match in `input2`
- Keep rows only if there is a match in both arguments
  - `inner_join`: return all rows from `input1` where there are matching values in `input2`
- Keep all rows from both arguments
  - `full_join`: return all rows and columns from `input1` and `input2`

I can never remember which does which... `?inner_join` helps!

Joining data

Questions about joining data?

Reshaping datasets

The same data can be organised in multiple ways.

Let's consider the following example: each patient has a ID, and a disease status at each time:

```r
ex_long <- tibble(
  "id" = c(1,1,2,2,3,3,3),
  "time" = c(1,2,1,2,1,2,3),
  "status" = c("healthy", "sick",
               "healthy", "healthy",
               "healthy", "healthy", "sick"))
```

Reshaping datasets

This data is in “long form”: one observation per row

```
ex_long
```

```
## # A tibble: 7 x 3
##      id  time status
##   <dbl> <dbl> <chr>
## 1     1     1 healthy
## 2     1     2 sick
## 3     2     1 healthy
## 4     2     2 healthy
## 5     3     1 healthy
## 6     3     2 healthy
## 7     3     3 sick
```

Reshaping datasets

Here is the *same data* in “wide form”: multiple observations per row

```r
ex_wide <- tibble("id" = 1:3,
                  "t1" = c("healthy", "healthy", "healthy"
                  "t2" = c("sick", "healthy", "healthy"),
                  "t3" = c(NA, NA, "sick"))
ex_wide
```

```
## # A tibble: 3 x 4
##      id t1      t2      t3
##   <int> <chr>   <chr>   <chr>
## 1     1 healthy sick    <NA>
## 2     2 healthy healthy <NA>
## 3     3 healthy healthy sick
```

Reshaping datasets

Different analyses require different shapes of data.

We can `pivot` data from long to wide, and wide to long

- `pivot_longer`: turns a wide dataset into a long dataset
- `pivot_wider`: turns a long dataset into a wide dataset

Reshaping datasets

Remember carefully what `ex_wide` looks like:

```
ex_wide
```

```
## # A tibble: 3 x 4
##      id t1      t2      t3
##   <int> <chr>   <chr>   <chr>
## 1     1 healthy sick    <NA>
## 2     2 healthy healthy <NA>
## 3     3 healthy healthy sick
```

Reshaping datasets

pivot_longer: column names go into a new column ("name"), and the value in the column goes into a new column ("value"):

```r
ex_wide %>%
  pivot_longer(cols = c("t1", "t2", "t3"))
```

```
## # A tibble: 9 x 3
##      id name  value
##   <int> <chr> <chr>
## 1     1 t1    healthy
## 2     1 t2    sick
## 3     1 t3    <NA>
## 4     2 t1    healthy
## 5     2 t2    healthy
## 6     2 t3    <NA>
## 7     3 t1    healthy
## 8     3 t2    healthy
## 9     3 t3    sick
```

We specify the columns to convert to long format with `cols`

Reshaping datasets

We can rename `name` and `value` using `names_to` and `values_to`:

```
ex_wide %>%
  pivot_longer(cols = c("t1", "t2", "t3"),
               names_to="time",
               values_to = "status")
```

```
## # A tibble: 9 x 3
##      id time  status
##   <int> <chr> <chr>
## 1     1 t1    healthy
## 2     1 t2    sick
## 3     1 t3    <NA>
## 4     2 t1    healthy
## 5     2 t2    healthy
## 6     2 t3    <NA>
## 7     3 t1    healthy
## 8     3 t2    healthy
## 9     3 t3    sick
```

What's another way to rename a column?

Reshaping datasets

Lots of columns to pivot? If the columns are all next to each other, you can specify the first and the last:

```r
ex_wide %>%
  pivot_longer(cols = t1:t3)
```

```
## # A tibble: 9 x 3
##      id name  value
##   <int> <chr> <chr>
## 1     1 t1    healthy
## 2     1 t2    sick
## 3     1 t3    <NA>
## 4     2 t1    healthy
## 5     2 t2    healthy
## 6     2 t3    <NA>
## 7     3 t1    healthy
## 8     3 t2    healthy
## 9     3 t3    sick
```

Reshaping datasets: long to wide

Now remember what the long version looks like:

```
ex_long
```

```
## # A tibble: 7 x 3
##      id  time status
##   <dbl> <dbl> <chr>
## 1     1     1 healthy
## 2     1     2 sick
## 3     2     1 healthy
## 4     2     2 healthy
## 5     3     1 healthy
## 6     3     2 healthy
## 7     3     3 sick
```

Reshaping datasets

With `pivot_wider`, we specify *where the column names are from* with `names_from`, and *where the values are from* with `values_from`:

```
ex_long %>%
  pivot_wider(names_from = "time",
              values_from = "status")
```

```
## # A tibble: 3 x 4
##      id `1`     `2`     `3`
##   <dbl> <chr>   <chr>   <chr>
## 1     1 healthy sick    <NA>
## 2     2 healthy healthy <NA>
## 3     3 healthy healthy sick
```

Reshaping datasets

There are lots of options for customisation! Check them out with `?pivot_wider`. e.g., to prefix the new column names, use `names_prefix`:

```
ex_long %>%
  pivot_wider(names_from = "time",
              values_from = "status",
              names_prefix = "t_")
```

```
## # A tibble: 3 x 4
##      id t_1     t_2     t_3
##   <dbl> <chr>   <chr>   <chr>
## 1     1 healthy sick    <NA>
## 2     2 healthy healthy <NA>
## 3     3 healthy healthy sick
```

Summary

- Reorder data with `arrange()`
- Joining multiple datasets
  - Adding columns with `bind_cols()`
  - Adding rows with `bind_rows()`
  - Removing duplicates with `distinct()`
  - Joining different datasets with `full_join()`
- Reshaping data with `pivot_wider()` and `pivot_longer()`

Next week: advanced programming, including writing custom functions and using control structures (e.g. loops)

The plan

- 5 minute break
- In-class exercise available via Canvas
  - Designed to be completed by 3:20 p.m.
  - Due today 6:30 p.m.
    - *Yellow sticky note* = urgent; *blue sticky note* = non-urgent
- Homework due next week by 1 p.m. Friday
- Office hours as always: Tuesday, Wednesday and Thursday