Week 4: Joins and Binds

Ellen Bledsoe

2024-02-06

Binds and Joins

In today's lesson, we will be talking about how to bring multiple dataframes together into one data frame.

Often times, we have a lot of data for one project that are related but storing all of the data in one file would add unnecessary redundancy (e.g., data in certain rows would need to be repeated too often). Other times, data has been collected separately and needs to be combined before analysis.

Being able to join together data from related tables is a key skill in data science, and for working with larger data structures (databases with their own languages, like SQL).

To understand binds and joins, we will continue using the Portal datasets.

Set-up

Let's go ahead and load our usual packages.

```r
# load packages
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(readr)
```

And load the `surveys`, `species`, and `plots` data.

```r
surveys <- read_csv("surveys.csv")
```

```
## Rows: 35549 Columns: 9
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): species_id, sex
## dbl (7): record_id, month, day, year, plot_id, hindfoot_length, weight
##
## 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.
```

```
species <- read_csv("species.csv")
```

```
## Rows: 54 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): species_id, genus, species, taxa
##
## 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.
```

```
plots <- read_csv("plots.csv")
```

```
## Rows: 24 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): plot_type
## dbl (1): plot_id
##
## 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.
```

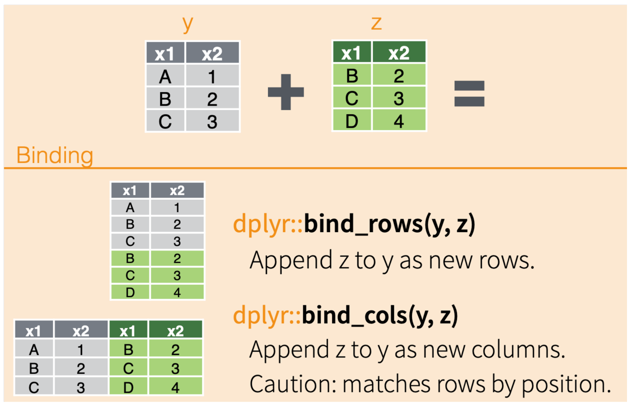
Binds vs. Joins

We have 2 main methods of combining datasets: binds and joins. They work in different ways.

Binds

One way we can combine data sets in through binds. Binds act similarly to gluing datasets together instead of sorting rows to match up.

They don't match up data based on unique identifiers; instead they match up data by column name (`bind_rows`) or row position (`bind_cols`).



dplyr::**bind_rows(y, z)**
Append z to y as new rows.

dplyr::**bind_cols(y, z)**
Append z to y as new columns.
Caution: matches rows by position.

These can be very useful, but you need to be careful using them, especially `bind_cols`, as the function will automatically assume that the rows are in the correct positions.

Let's work with an example.

The `surveys` data only goes through 2002, but we have a lot more data from the site! Let's pull down all of the rodent data since 2002 from the `portalr` package, a package we made to make working with the (actual) Portal data a bit easier. Run the following code chunk.

```
# load our packages
library(portalr)

# load the new rodent data and do a little cleaning
# don't worry if you don't know what the mutate function is doing!
new_rodents <- summarize_individual_rodents() %>%
  select(month, day, year, plot_id = plot, species_id = species, sex, hindfoot_length = hfl, weight = w
  filter(year > 2002) %>%
  mutate(record_id = seq(nrow(surveys) + 1, nrow(surveys) + nrow(.))) %>%
  as_tibble()
```

```
## Loading in data version 3.168.0
```

```
new_rodents
```

```
## # A tibble: 35,315 x 9
##    month   day  year plot_id species_id sex   hindfoot_length weight record_id
##    <int> <int> <int>   <int> <fct>      <chr>           <int>  <dbl>     <int>
##  1     2     1  2003       1 PB         M                  27     52     35550
##  2     2     1  2003       1 PB         M                  26     32     35551
##  3     2     1  2003       1 DO         M                  34     47     35552
##  4     2     1  2003       1 PP         M                  22     17     35553
##  5     2     1  2003       1 PP         F                  22     19     35554
##  6     2     1  2003       1 DO         F                  36     53     35555
##  7     2     1  2003       1 OT         M                  20     27     35556
##  8     2     1  2003       1 DO         F                  35     59     35557
##  9     2     1  2003       1 DO         F                  36     NA     35558
## 10     2     1  2003       1 NA         F                  31    194     35559
## # i 35,305 more rows
```

When we look at the `new_rodents` data frame, we can see the same columns as in `surveys`, though the `record_id` column is at the end.

We want to bind these two dataframes together so that we have *all* of the rodent data in one dataframe. To do so, we would use `bind_rows()`. Even though columns are getting matched up, the things we are binding together are multiple rows, hence `bind_rows()`.

The arguments for both `bind_rows()` and `bind_cols()` are the names of the dataframes you want to bind together.

```r
all_rodents <- bind_rows(surveys, new_rodents)

head(all_rodents)
```

```
## # A tibble: 6 x 9
##   record_id month   day  year plot_id species_id sex   hindfoot_length weight
##       <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
## 1         1     7    16  1977       2 NL         M                  32     NA
## 2         2     7    16  1977       3 NL         M                  33     NA
## 3         3     7    16  1977       2 DM         F                  37     NA
## 4         4     7    16  1977       7 DM         M                  36     NA
## 5         5     7    16  1977       3 DM         M                  35     NA
## 6         6     7    16  1977       1 PF         M                  14     NA
```

```r
tail(all_rodents)
```

```
## # A tibble: 6 x 9
##   record_id month   day  year plot_id species_id sex   hindfoot_length weight
##       <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
## 1     70859     2    27  2022       9 DO         M                  33     53
## 2     70860     2    27  2022       9 DO         M                  34     53
## 3     70861     2    27  2022       9 DM         M                  37     41
## 4     70862     2    27  2022       9 DO         M                  35     42
## 5     70863     2    NA  2022       1 <NA>       <NA>               NA     NA
## 6     70864     2    NA  2022      16 <NA>       <NA>               NA     NA
```

Notice that the data has been successfully rearranged to have the columns match up. This will only happen if the columns have the *exact* same names.

Let's Practice!

Try your hand at Question 6 on the assignment.

Joins

Multiple Data Tables

When we talked about data structure, one of the things we discussed was splitting data into multiple tables. This lets us avoid unnecessary redundant information, like listing the full taxonomy for every individual of a species. This, in turn, makes storage more efficient and allows us to make changes in one place, not hundreds of places.

Our goal is for each table to contain a single kind of information.

Let's look at this in the Portal dataset.

```r
# surveys <- read_csv("surveys.csv")
species <- read_csv("species.csv")
```

```
## Rows: 54 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): species_id, genus, species, taxa
##
## 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
plots <- read_csv("plots.csv")
```

```
## Rows: 24 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): plot_type
## dbl (1): plot_id
##
## 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.
```

In the Portal dataset:

- `surveys`: information about individuals
- `species`: information about species
- `plots`: information about plots

This way, if a species name changes (for example), we only need to change it in the species table rather than tens of thousands of times.

When we need to combine the datasets together to use data from multiple data frames, we use a join function.

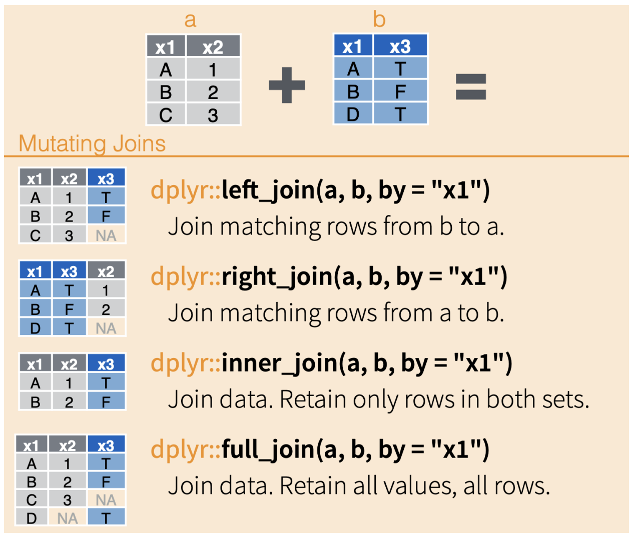
How Joins Work

Joins are arguably the more complicated of the two types of ways to combine data, but they are, therefore, the more flexible and useful of the two.

The magic comes because they *match up columns of data based on unique identifiers in each row of data.*

In the following diagram, the two example data frames have the column `x1` in common, and each of the values in `x1` are unique (no repeats in the same data frame). When combining the datasets, all of the columns are added, and their rows are matched up to their respective values in the `x1` column.

This can happen a couple ways, depending on which data frame is the reference and how much data you want to retain.


a
b
Mutating Joins
dplyr::left_join(a, b, by = "x1")
Join matching rows from b to a.
dplyr::right_join(a, b, by = "x1")
Join matching rows from a to b.
dplyr::inner_join(a, b, by = "x1")
Join data. Retain only rows in both sets.
dplyr::full_join(a, b, by = "x1")
Join data. Retain all values, all rows.

To enable us to make these connections, the tables need one (or more) columns that link them together.

In the case of the Portal data, there is one column that links the `surveys` and `species` tables: `species_id`

There is also one column that links the `surveys` and `plots` tables: `plot_id`

Let's join the surveys and the species tables together using an "inner join."

To do this, we use the inner_join function from `dplyr`. It takes three arguments:

1) The first of the two tables we want to join
2) The second of the two tables we want to join
3) The column, or columns, that provide the linkage between the two tables, specified in a `join_by()` function,

```
combined <- inner_join(surveys, species, join_by(species_id))
combined
```

```
## # A tibble: 34,786 x 12
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##  1         1     7    16  1977       2 NL         M                  32     NA
##  2         2     7    16  1977       3 NL         M                  33     NA
##  3         3     7    16  1977       2 DM         F                  37     NA
##  4         4     7    16  1977       7 DM         M                  36     NA
##  5         5     7    16  1977       3 DM         M                  35     NA
##  6         6     7    16  1977       1 PF         M                  14     NA
##  7         7     7    16  1977       2 PE         F                  NA     NA
##  8         8     7    16  1977       1 DM         M                  37     NA
##  9         9     7    16  1977       1 DM         F                  34     NA
## 10        10     7    16  1977       6 PF         F                  20     NA
## # i 34,776 more rows
## # i 3 more variables: genus <chr>, species <chr>, taxa <chr>
```

Looking at the combined table, we can see that on every row with a particular value for `species_id`, the join has added the matching values on `genus`, `species`, and `taxa`.

One way to think about this join is that it adds the relevant information in the species table to the surveys table. Often for scientific data we can think about there being one main table, the surveys table in our case, and multiple supplementary tables that provide additional details.

Inner joins keep information from both tables when both tables have a matching value in the join column.

Let's go back up and look at the visualization of the inner join. When we use the `inner_join` function to merge together table a and b, the result only has rows for which the common column (`x1`) have the same values in each table (A and B).

Translating this to the rodent data, that means that we dropped any rows for which there is no matching `species_id`.

Scroll to Line 324 in the `surveys` table; these have missing species IDs. If you look in the `combined` table, ou'll notice that `record_id`, 324-326 are missing from the table.

The other join functions might have handled this differently, based on how they work.

For example, left joins keep all rows in the first, or left, table. If we wanted to keep rows with missing species IDs in the `surveys` data frame, we could use `left_join()`.

```
combined <- left_join(surveys, species, join_by(species_id))
```

There are also right joins, which keep all rows in the second (or right) table, and full joins, which keep all rows from both tables, even there isn't a matching row.

For our exercises, we'll focus on using inner joins.

Multi-table Joins

Sometimes, we need to combine more than two tables.

To join more than two tables, we start by joining two tables, then join the resulting table to a third table, and so on.

For Portal, we could start by joining the `surveys` and the `species` tables. We can then combine the resulting table with the `plots` table to get a fully merged data frame.

```
full_data <- surveys %>%
  inner_join(species, join_by(species_id)) %>%
  inner_join(plots, join_by(plot_id))
```

If it helps you to keep track of which data tables are being joined when (especially for left and right joins), you can also use the placeholder for your chosen pipe.

```
full_data <- surveys %>%
  inner_join(., species, join_by(species_id)) %>%
  inner_join(., plots, join_by(plot_id))
```

Let's Practice

You should be able to complete Questions 4, 5, and 7