Manipulating and Joining Data in R with dplyr

Lubov McKone & Chen Chiu
Johns Hopkins Libraries Data Services

2024-04-11

This webinar will be recorded! 🎥

Your continued participation indicates your consent to be recorded. This recording may be shared with the JHU community.

Any questions you ask verbally or in chat will be edited to protect your identity.

JHU Data Services

We help faculty, researchers, and students find, use, manage, visualize, and share data.

- Find out more at dataservices.library.jhu.edu
- Email us for a consultation at dataservices@jhu.edu
- Share your research data at archive.data.jhu.edu

What you will learn today

- How to reshape data using the powerful `dplyr` package
- How to use the pipe `|>` to simplify code
- How to join two datasets together using different approaches and conditions
- Additional resources for manipulating and joining data using `dplyr`

You should have:

- A template R script that we will fill out today called `class_script_blank.R`
- dplyr cheatsheet
- Basic knowledge of R
  - Installing and loading packages
  - Basic terminology of R or programming in general

Why reshape data?

- Calculate new variables to analyze
- Summarize data differently to suit your unit of analysis
- Rearrange or sort data to make it easier to visualize

Libraries

Today we'll be using the tidyverse library, which includes dplyr.

```
library(tidyverse)
```

```
Warning: package 'tidyverse' was built under R version 4.3.2
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ dplyr     1.1.2     ✔ readr     2.1.4
✔ forcats   1.0.0     ✔ stringr   1.5.0
✔ ggplot2   3.4.2     ✔ tibble    3.2.1
✔ lubridate 1.9.2     ✔ tidyr     1.3.0
✔ purrr     1.0.1
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

Review: reading and viewing data

```
# we'll be looking at data on Groundhog predictions
groundhogs <- readr::read_csv('https://raw.githubusercontent.com/rfordatasc
predictions <- readr::read_csv('https://raw.githubusercontent.com/rfordatas
```

You can view a dataframe in R using `View()` or by clicking the object in the environment pane.

Let's take a look at our groundhog predictions dataset:

```
head(predictions)
```

id <dbl>	year <dbl>	shadow <lgl>
1	1886	*NA*
1	1887	TRUE
1	1888	TRUE
1	1889	*NA*
1	1890	FALSE

id	year	shadow
<dbl>	<dbl>	<lgl>
1	1891	*NA*

6 rows | 1-3 of 4 columns

Our task today

- We are groundhog mythbusters and our goal is to collect some summary statistics about the groundhog prediction phenomenon.
- Our main question is whether different groundhogs are more or less likely to predict an early spring.
- Over the course of this workshop, we'll be creating summary tables that will set us up for further visualization and analysis.

The dplyr package

- dplyr is a "grammar" of data manipulation
- dplyr is a set of R functions that work together to help you solve the most common data manipulation challenges, like:
  - Filtering out certain rows and sorting your data
  - Calculating a new column based on existing columns
  - Grouping and summarizing data
  - Joining data from different sources

dplyr grammar

- dplyr’s core is a set functions that can be divided into 4 groups based on what they operate across:
    - rows
    - columns
    - groups
    - tables
- We’ll call these the dplyr *verbs*
- dplyr also contains a number of useful *helper functions* that operate on single values or arrays. We’ll introduce those along the way.

anatomy of a dplyr verb

In every dplyr verb:

- the first argument is always dataframe
- the output is always a new dataframe
- arguments with a . in front of them are settings for the function, not column names

the pipe 💉

- Each verb does one thing very well, so you'll typically chain together multiple verbs. The **pipe** helps you do this by passing the result of every action onto the next action.
- The pipe is represented in R as `|>`. Everything to the left of the pipe is passed as the first argument to the thing immediately to the right of the pipe.
- `x |> f(y)` is equivalent to `f(x, y)`
- `x |> f(y) |> g(x)` is equivalent to`g(f(x,y), z)`
- ⚠️ The pipe does not save new variables automatically

row verbs: `filter()`

- `filter()` : keep rows based on the value of one or more columns
- You can compose conditions using `==`, `>`, `<`, `>=`, `<=`, `!=`, and include multiple conditions using `&` or `|`
- The `%in%` operator can serve as a combination of `|` and `==`

row verbs: `filter()`

```r
# find groundhog predictions from 2020
filter(predictions, year == 2020)
```

id <dbl>	year <dbl>	shadow <lgl>
1	2020	FALSE
2	2020	FALSE
3	2020	TRUE
4	2020	TRUE
5	2020	TRUE
6	2020	FALSE
7	2020	TRUE
8	2020	FALSE
9	2020	TRUE
10	2020	FALSE

1-10 of 67 rows | 1-3 of 4 colu… Previous **1** 2 3 4 5 6 7 Next

```r
# find groundhog predictions from 2020 and 2021
filter(predictions, year == 2020 | year == 2021)
```

id <dbl>	year <dbl>	shadow <lgl>
1	2020	FALSE
1	2021	TRUE
2	2020	FALSE
2	2021	*NA*
3	2020	TRUE
3	2021	FALSE
4	2020	TRUE
4	2021	FALSE
5	2020	TRUE
5	2021	FALSE

1-10 of 136 rows | 1-3 of 4 co… Previous **1** 2 3 4 5 6 ... 14 Next

```r
filter(predictions, year %in% c(2020, 2021))
```

id <dbl>	year <dbl>	shadow <lgl>

id <dbl>	year <dbl>	shadow <lgl>
1	2020	FALSE
1	2021	TRUE
2	2020	FALSE
2	2021	*NA*
3	2020	TRUE
3	2021	FALSE
4	2020	TRUE
4	2021	FALSE
5	2020	TRUE
5	2021	FALSE

1-10 of 136 rows | 1-3 of 4 co… Previous **1** 2 3 4 5 6 ... 14 Next

`filter()`: your turn!

Find groundhog predictions between 1900 and 2000.

Bonus: Use the pipe in your answer!

`filter()`: your turn!

```
# find predictions between 1900 and 2000
predictions |>
  filter(year >= 1900 & year <= 2000)
```

id <dbl>	year <dbl>	shadow <lgl>
1	1900	TRUE
1	1901	TRUE
1	1902	FALSE
1	1903	TRUE
1	1904	TRUE
1	1905	TRUE
1	1906	TRUE
1	1907	TRUE
1	1908	TRUE
1	1909	TRUE

1-10 of 421 rows | 1-3 of 4 co... Previous **1** 2 3 4 5 6 ... 43 Next

`filter()`: useful helper functions

- `between()` tests if a variable falls between two values (inclusive)
- `near()` tests if a variable is within a certain range of a given number (you can set the tolerance)
- `is.na()` tests whether the variable is NA. Use is conjunction with `!` to filter for non-NA values.

row verbs: `arrange()`

`arrange()`: changes the row order based on one or more columns

You can wrap the columns with `desc()` to sort in descending order

```r
# sort our predictions by year
arrange(predictions, year)
```

id <dbl>	year <dbl>	shadow <lgl>
1	1886	NA
1	1887	TRUE
1	1888	TRUE
1	1889	NA
1	1890	FALSE
1	1891	NA
1	1892	NA
1	1893	NA
1	1894	NA
1	1895	NA

Previous 1 2 3 4 5 6 ... 14 Next

```r
# sort our predictions by year
arrange(predictions, desc(year))
```

id <dbl>	year <dbl>	shadow <lgl>
1	2023	TRUE
2	2023	FALSE
3	2023	FALSE
4	2023	TRUE
5	2023	FALSE
6	2023	TRUE
7	2023	FALSE
8	2023	FALSE
9	2023	TRUE
10	2023	FALSE

Previous 1 2 3 4 5 6 ... 14 Next

row verbs: `distinct()`

`distinct()`: finds all the unique rows based on the values of one or more columns

- Without any additional inputs, `distinct()` finds and keeps the first occurence of all unique rows
- You can optionally supply one or more columns to check for distinct combinations of

```
predictions |>
  distinct(year)
```

year <dbl>
1886
1887

year
<dbl>
1888
1889
1890
1891
1892
1893
1894
1895

1-10 of 138 rows

Previous **1** 2 3 4 5 6 ... 14 Next

🚩 checkpoint: row verbs

Let's put it all together!

- Remove rows with no prediction record
- Remove duplicate predictions
- Sort the result by year, descending
- Assign the result to predictions, overwriting the previous dataframe

🚩 checkpoint: row verbs

```
# create a subset of your data where "shadow" has a value of either TRUE or
predictions <- predictions |>
  filter(shadow %in% c(TRUE, FALSE)) |>
  distinct(year, id, .keep_all = TRUE) |>
  arrange(desc(year))
```

group verbs: `group_by()`

`group_by()` groups your dataframe

On it's own, it doesn't change your data. But you can feed the "grouped" output into other special functions to apply different transformations to each group in your data.

```
# group predictions by year
predictions |>
  group_by(year)
```

id <dbl>	year <dbl>	shadow <lgl>
1	2023	TRUE
2	2023	FALSE
3	2023	FALSE
4	2023	TRUE
5	2023	FALSE

id <dbl>	year <dbl>	shadow <lgl>
6	2023	TRUE
7	2023	FALSE
8	2023	FALSE
9	2023	TRUE
10	2023	FALSE

1-10 of 1,317 rows | 1-3 of 4 … Previous **1** 2 3 4 5 6 ... 132 Next

`n()` within `summarize()`

- `summarize()` reduces the dataframe to a summary table with one row for each group and one or more calculations by group
- One of the most important summaries is `n()`, which counts the observations in each group.
- Let's try it together: How many predictions were made in each year?

n() within summarize()

```r
# How many predictions were made in each year?
predictions |>
  group_by(year) |>
  summarize(n_predictions = n()) |>
  arrange(desc(year))
```

year <dbl>	n_predictions <int>
2023	70
2022	71
2021	60
2020	65
2019	60
2018	61
2017	57
2016	51
2015	51

year	n_predictions
<dbl>	<int>
2014	48

1-10 of 128 rows Previous **1** 2 3 4 5 6 ... 13 Next

`summarize()` helper functions

- Other powerful summary functions include:
  - `n_distinct()`: counts the number of distinct values of a given column within a group
  - `max()` and `min()`: finds the max and min value of a given column within a group
- Exercises:
  - How many different groundhogs made predictions each year?
  - What is the first year each groundhog made a prediction?

`summarize()` helper functions

```
# How many different groundhogs made predictions each year?
predictions |>
  group_by(year) |>
  summarize(n_groundhogs = n_distinct(id)) |>
  arrange(desc(n_groundhogs))
```

year <dbl>	n_groundhogs <int>
2022	71
2023	70
2020	65
2018	61
2019	60
2021	60
2017	57
2015	51
2016	51

year	n_groundhogs
<dbl>	<int>
2014	48

1-10 of 128 rows Previous **1** 2 3 4 5 6 ... 13 Next

summarize() helper functions

```r
# What is the first year each groundhog made a prediction?
predictions |>
  group_by(id) |>
  summarize(first_prediction = min(year))
```

id <dbl>	first_prediction <dbl>
1	1887
2	1926
3	1955
4	1969
5	1979
6	1980
7	1982
8	1980
9	1993
10	1983

1-10 of 75 rows

Previous **1** 2 3 4 5 6 ... 8 Next

`sum()` within `summarize()`

- `sum()`: finds the sum of a given column within a group. You can also specify conditions within `sum()` to calculate the number of records within a group that meet a certain condition.
- Exercise: Let's return to our dataframe with the number of predictions in each year. How would we add a column for the number of shadows seen in each year?

sum() within summarize()

```r
# Let's return to our dataframe with the number of predictions in each year
# How would we add a column for the number of shadows seen in each year?
predictions |>
  group_by(year) |>
  summarize(n_predictions = n(),
            n_shadows = sum(shadow == TRUE))
```

year <dbl>	n_predictions <int>	n_shadows <int>
1887	1	1
1888	1	1
1890	1	0
1898	1	1
1900	1	1
1901	1	1
1902	1	0
1903	1	1
1904	1	1

year	n_predictions	n_shadows
<dbl>	<int>	<int>
1905	1	1

1-10 of 128 rows Previous **1** 2 3 4 5 6 ... 13 Next

🚩 checkpoint: group verbs

Your turn! Create a dataframe with three variables:

- groundhog id
- the number of total predictions each groundhog has made
- the number of times each groundhog has seen it's shadow.

checkpoint: group verbs

```
# Create a dataframe with 3 variables:
# groundhog id
# the number of total predictions each groundhog has made
# the number of times each groundhog has seen its shadow
predictions |>
  group_by(id) |>
  summarize(n_predictions = n(),
            n_shadows = sum(shadow == TRUE))
```

id <dbl>	n_predictions <int>	n_shadows <int>
1	128	108
2	91	72
3	60	25
4	55	23
5	45	18
6	40	12
7	40	4

id	n_predictions	n_shadows
<dbl>	<int>	<int>
8	35	12
9	30	10
10	28	9

1-10 of 75 rows Previous **1** 2 3 4 5 6 ... 8 Next

column verbs

Now that we've calculated some summary variables within the groups that interest us (groundhog and year), we might want to use those summary variables to calculate more new variables.

column verbs: `mutate()`

`mutate()` adds new columns calculated from existing columns

- By default, columns are added on the left side of the dataframe. You can use the `.before` or `.after` to specify where the new variable should fall

```
# calculate how many characters are in the details field and put the variab
predictions |>
  mutate(details_length = nchar(details), .after = id)
```

id <dbl>	details_length <int>	year <dbl>	shadow <lgl>
1	158	2023	TRUE
2	*NA*	2023	FALSE
3	24	2023	FALSE

id <dbl>	details_length <int>	year <dbl>	shadow <lgl>
4	*NA*	2023	TRUE
5	*NA*	2023	FALSE
6	*NA*	2023	TRUE
7	*NA*	2023	FALSE
8	*NA*	2023	FALSE
9	*NA*	2023	TRUE
10	*NA*	2023	FALSE

1-10 of 1,317 rows | 1-4 of 5 … Previous 1 2 3 4 5 6 ... 132 Next

re-coding data with `mutate()`

`if_else()` tests for a condition and returns one value if true and another if false.

```r
# create a column that indicates whether the prediction was made by Punxata
predictions |>
  mutate(phil = if_else(id == 1, 'TRUE', 'FALSE'))
```

id <dbl>	year <dbl>	shadow <lgl>
1	2023	TRUE
2	2023	FALSE
3	2023	FALSE
4	2023	TRUE
5	2023	FALSE
6	2023	TRUE
7	2023	FALSE
8	2023	FALSE

id <dbl>	year <dbl>	shadow <lgl>
9	2023	TRUE
10	2023	FALSE

1-10 of 1,317 rows | 1-3 of 5 … Previous **1** 2 3 4 5 6 ... 132 Next

re-coding data with `mutate()`

`case_when()` tests for multiple conditions and maps them to values accordingly.

```
# create a column that indicates the century of the predictions
predictions |>
  mutate(century = case_when(year < 1900 ~ 19,
                             year < 2000 & year >= 1900 ~ 20,
                             year >= 2000 ~ 21))
```

id <dbl>	year <dbl>	shadow <lgl>
1	2023	TRUE
2	2023	FALSE
3	2023	FALSE
4	2023	TRUE
5	2023	FALSE
6	2023	TRUE
7	2023	FALSE

id	year	shadow
<dbl>	<dbl>	<lgl>
8	2023	FALSE
9	2023	TRUE
10	2023	FALSE

1-10 of 1,317 rows | 1-3 of 5 … Previous **1** 2 3 4 5 6 … 132 Next

column verbs: `select()` and `rename()`

- `select()` keeps a subset of columns
  - You can select by name, series, test for data type (`select(where(is.character()))`) or use other helper functions such as `starts_with()`, `ends_with()`, or `contains()`
  - You can rename variables as you select them with `=` , with the new name on the left and old on the right
- `rename()` works the same way as renaming in select with `=`

🚩 checkpoint: put it all together!

Let's return to our original research question: Are certain groundhogs more likely to see their shadow than others?

Working off of our table with the number of predictions and number of shadows seen per groundhog, lets:

- Add a column called `shadow_percent` that gives the percentage of time each groundhog sees its shadow
- Filter for groundhogs with more than 5 predictions
- Keep only the variables `id` and `shadow_percent`, and rename id to `groundhog_id`
- Assign the result to a variable `groundhog_predictions`

🚩 checkpoint: put it all together!

```
groundhog_predictions <- predictions |>
  group_by(id) |>
  summarize(n_predictions = n(),
            n_shadows = sum(shadow == TRUE)) |>
  mutate(shadow_percent = n_shadows/n_predictions) |>
  filter(n_predictions > 5) |>
  select(id, shadow_percent) |>
  rename(groundhog_id = id)
```

table verbs: joining data

We've done a lot with the mere 4 variables in our predictions table!

What if we wanted to enhance our data with more information about each groundhog from the `groundhogs` table?

```
head(groundhogs)
```

id <dbl>	slug <chr>	shortname <chr>	name <chr>
1	punxsutawney-phil	Phil	Punxsutawney Pl
2	octoraro-orphie	Orphie	Octoraro Orphie
3	wiarton-willie	Willie	Wiarton Willie
4	jimmy-the-groundhog	Jimmy	Jimmy the Groun
5	concord-charlie	Charlie	Concord Charlie

id	slug	shortname	name
<dbl>	<chr>	<chr>	<chr>
6	buckeye-chuck	Chuck	Buckeye Chuck

6 rows | 1-5 of 17 columns

join terminology

There are two main types of join:

- **mutating joins** add variables from one dataframe to another based on matching characteristics between the two
- **filtering joins** subset one dataframe based on matching characteristics with another dataframe

join terminology

- Every join involves a **primary key** and a **foreign key**
  - A primary key is a variable or set of variables that uniquely identifies an observation
  - A foreign key is just another table's primary key that matches your tables' primary key. It might have a different name or be spread across more or less variables.
- The first step when joining data is to identify the primary and foreign keys you'll work with
- Always check that your primary & foreign keys are truly unique to each row!

joining predictions & groundhogs

```r
head(groundhog_predictions, 3)
```

groundhog_id <dbl>	shadow_percent <dbl>
1	0.8437500
2	0.7912088
3	0.4166667

3 rows

```r
head(groundhogs, 3)
```

id <dbl>	slug <chr>	shortname <chr>	name <chr>
1	punxsutawney-phil	Phil	Punxsutawney Phil
2	octoraro-orphie	Orphie	Octoraro Orphie
3	wiarton-willie	Willie	Wiarton Willie

3 rows | 1-5 of 17 columns

joining predictions & groundhogs

- How would we determine if there is a difference between the average shadow prediction rate of different types of groundhogs?
- primary key: `groundhog_id` in `groundhog_predictions`
- foreign key: `id` in `groundhogs`

joining predictions & groundhogs

- We want to add the variables from `groundhogs` to our `groundhog_predictions` table
- We'll need a **mutating join**, specifically a **left join**.
- A **left join** retains all rows in the left dataframe, and adds additional data in from the right dataframe if the keys match.
- `left_join(x, y, join_by(x.key == y.key))`

joining predictions & groundhogs

```r
left_join(groundhog_predictions, groundhogs, join_by(groundhog_id == id))
```

groundhog_id <dbl>	shadow_percent <dbl>	slug <chr>
1	0.84375000	punxsutawney-phil
2	0.79120879	octoraro-orphie
3	0.41666667	wiarton-willie
4	0.41818182	jimmy-the-groundhog
5	0.40000000	concord-charlie
6	0.30000000	buckeye-chuck
7	0.10000000	general-beauregard-lee
8	0.34285714	french-creek-freddie
9	0.33333333	gertie-the-groundhog
10	0.32142857	dunkirk-dave

1-10 of 60 rows | 1-4 of 18 colu… Previous 1 2 3 4 5 6 Next

more mutating joins

`right_join()` keeps everything in the right dataframe and adds in data from the left

more mutating joins

`inner_join()` keeps rows with keys that appear in both dataframes

`full_join()` keeps all rows from both dataframes

filtering joins

filtering joins subset one dataframe based on matching characteristics with another dataframe. In filtering

`semi_join(x, y)` keeps all rows in x with a match in y

`anti_join(x, y)` returns all rows in x without a match in y

join exercises

- `groundhog_predictions` contains one row per 50 unique groundhogs
- `groundhogs` contains one row per 65 unique groundhogs
- Every groundhog in `groundhog_predictions` appears in `groundhogs`
  - How many rows would each of the following joins have: right join with `groundhogs` on the right, inner join, full join, semi_join, anti_join?

more complex join conditions

- Within `join_by()`, we can use more complex conditions than whether `key == key`
- You can use other numeric operations like `>`, `<`, etc.
- The `closest()` function matches the closest key to another key based on some criteria (closest value at all, closest value that is larger, etc.)
- `between()` and `within()` can test whether a value falls between two other values. This is useful if you want to join events that happened within a given time span.

other table verbs

- `bind_rows()` pastes rows onto the bottom of a dataframe
- `bind_cols()` pastes columns onto the right of a dataframe.
- There is no matching logic in these functions, you can think of them as copy-and-paste.

🏁 finish line

Let's put everything we've learned together!

Let's create a summary table that gives the rate at which each type of groundhog sees its' shadow

🏁 finish line

```r
predictions |>
  left_join(groundhogs, join_by(id == id)) |>
  group_by(type) |>
  summarize(n_groundhogs = n_distinct(id),
            n_predictions = n(),
            n_shadows = sum(shadow == TRUE)) |>
  mutate(percent_shadow = n_shadows/n_predictions)
```

type <chr>	**n_g**
Ameraucana chicken	
Animatronic groundhog	
Armadillo	
Atlantic lobster	
Beaver	
Bullfrog	
Cat	
Groundhog	

type <chr>	**n_g**
Groundhog golf club cover	
Groundhog puppet	

1-10 of 26 rows | 1-4 of 5 columns Previous **1** 2 3 Next

🏆 bonus exercises

- write code to calculate the column `predictions_count` in `groundhogs`
- Write code to calculate the column `is_groundhog` in `groundhogs`

summary: verbs & helper functions

Verbs:

- `filter()`, `arrange()`, `distinct()`
- `group_by()`, `summarize()`
- `mutate()`
- `left_`, `right_`, `inner_`, `full_`, `semi_`, `anti_` joins
- `bind_rows` and `_cols`

Helper functions:

- `desc()`
- `n()`, `n_distinct()`, `min()`, `max()`, `sum()`
- `if_else()` and `case_when()`
- `between()` and `within()`

resources

- R for Data Science 2e, Chapters 3 & 19
- dplyr documentation

thank you! 🙏

other trainings

survey link