Day 3, Part 1: Manipulating Data using Tidyverse

Brennan Terhune-Cotter and Matt Dye

https://github.com/brennangitsit/2023_IAM3_R

Agenda

1. Pseudocode!
2. Welcome to the Tidyverse!
3. The pipe
4. Main tidyverse functions:
    1. Arrange and filter rows
    2. Select and mutate columns
5. Joining dataframes

Pseudocode

- What do you do if you're not sure how to write something in code?
- You write *pseudocode* first, and gradually change it into code.
- Very helpful for thinking about how to convert something from your language into the language of computers!

Writing Pseudocode

Write down your goal in steps

```
With a dataframe called 'people'
that has names and ages of people in my study
I need to calculate the age of people older t
```

Write pseudocode

```
For data frame 'people',
Find rows where 'age' is greater than 20,
From these rows, calculate the mean of 'age'
```

Slowly turn pseudocode into code...

```
With data frame 'people',
Use filter() to select rows where 'age' > 20,
Use summarise() with mean() to
calculate mean 'age' of these rows.
```

Until you have code!

```
library(dplyr)

people %>%
  filter(age > 20) %>%
  summarise(mean_age = mean(age))
```

Welcome to the Tidyverse!

What is the Tidyverse?

- A *package* is a themed collection of functions and datasets for doing something.
- Tidyverse is a *package of packages*.

R packages for data science

The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

The Pipe

The Pipe

- In the `magrittr` package, which is part of `tidyverse`
- The greatest invention since sliced bread
- The **pipe** (%>% | ctrl shift M) allows you to express a sequence of multiple operations clearly.
- Using the pipe is like writing "For object x, do this, ***then*** do this".

without the pipe:	**with the pipe:**
get a box	get a box
see this box? color it blue	then color it blue
see this blue box? make it bigger	then make it bigger
see this big blue box? open it	then open it
see this big blue open box? throw it away	then throw it away

The Pipe

"Assign 2 to x, **and then** add 3, **and then** subtract 4, **and then** print x":

```
1 library(magrittr) # has pipe
2 x <- 2
3 x <- add(x,3)
4 x <- subtract(x,4)
5 print(x)
```

```
[1] 1
```

```
1 library(magrittr) # has pipe
2 x <- 2 %>%
3   add(3) %>%
4   subtract(4) %>%
5   print()
```

```
[1] 1
```

- The pipe **passes** the object into the first argument of the next function
  - The first argument becomes "invisible" to you.
- This is very useful for data manipulation using tidyverse, because the first argument is almost always the df you are working on.

The Pipe: when NOT to use it

- But pipes aren't always the best way to go about things. For example, when doing arithmetic operations, this is simpler:

```
1  x <- print(2 + 3 - 4)
```

```
[1] 1
```

- You also don't want to use pipes if:
  - You want to look at "intermediate" objects in between steps (assign them new names!)
  - You use functions that don't accept your dataframe as the first argument.

Main Tidyverse Functions

- **group_by()** and **summarise()** data
- **arrange()** and **filter()** rows
- **select()** and **mutate()** columns

eruptions

To explain the main Tidyverse functions, we will be using the `eruptions` dataset:

volcano_number	volcano_name	country	last_known_eruption	year	primary_volcano_ty
210010	West Eifel Volcanic Field	Germany	8300 BCE	-8300	Volcanic field
210020	Chaine des Puys	France	4040 BCE	-4040	Lava dome(s)
210030	Olot Volcanic Field	Spain	Unknown	NA	Volcanic field
210040	Calatrava Volcanic Field	Spain	3600 BCE	-3600	Volcanic field
211004	Colli Albani	Italy	Unknown	NA	Caldera

group_by() and summarise()

- We will learn about these functions on the last day!
- You can get the mean, median, standard deviation, min, max, count, etc.
- You can also calculate your own summary statistics

What summary statistic does this code calculate?

```
eruptions %>%
  group_by(subregion) %>%
  summarise(mean_year_erupted = mean(year, na.rm=T))
```

Filtering Rows

```
1 ?dplyr::arrange
```

```
1 ?dplyr::filter
```

arrange()

- Orders rows by a variable
- Often used with summarise() to make tables more presentable

```
library(tidyverse)
eruptions %>%
  group_by(region) %>%
  summarise(mean_year = mean(year, na.rm=T),
            stdev_year = sd(year, na.rm=T)) %>%
  arrange(mean_year)
```

```
# A tibble: 19 × 3
   region                          mean_year stdev_year
   <chr>                               <dbl>      <dbl>
 1 Canada and Western USA             -1018.      3241.
 2 Mediterranean and Western Asia      -607.      3093.
 3 Kamchatka and Mainland Asia         -359.      2773.
 4 Africa and Red Sea                   245.      3333.
 5 Middle East and Indian Ocean         393.      2551.
 6 Antarctica                           528.      3095.
 7 South America                        554.      2430.
 8 México and Central America           888.      2249.
 9 Japan, Taiwan, Marianas              936.      2286.
10 Philippines and SE Asia              939.      2345.
11 Alaska                              1026.      1871.
12 Iceland and Arctic Ocean            1068.      1519.
13 Hawaii and Pacific Ocean            1101       1631.
14 Atlantic Ocean                      1186.      1568.
15 Melanesia and Australia             1380.      1682.
16 New Zealand to Fiji                 1416.      1578.
17 West Indies                         1497.       694.
18 Kuril Islands                       1648.      1695.
```

filter()

- Include or exclude cases based on values
- You do this by writing **logical statements** in filter()

```
1 kable(people)
```

name	age	humor
Alice	25	Bad
Spew	16	Good
Charlemagne	42	Good
Kay	18	Bad
Mackenzie	3	Terrible
Spirulina	16	Too Good
Jason	20	Good

```
1 # I only want my besties to be adults with a
2 # good (but not too good) sense of humor
3 besties <- people %>%
4   filter(age > 18,
5          humor == "Good")
6 kable(besties)
```

name	age	humor
Charlemagne	42	Good
Jason	20	Good

- Open 14_manipulate.Rmd
- Read **Arranging and Filtering Rows**

Manipulating Columns

```
1 ?dplyr::select
```

```
1 ?dplyr::mutate
```

select()

- Include, exclude, and rearrange columns
- Very important for reducing your cognitive load :)

```
eruptions %>%
  select(volcano_name, year, region, subregion) %>%
  select(-subregion) %>%
  head(6) %>%
  knitr::kable()
```

volcano_name	year	region
West Eifel Volcanic Field	-8300	Mediterranean and Western Asia
Chaine des Puys	-4040	Mediterranean and Western Asia
Olot Volcanic Field	NA	Mediterranean and Western Asia
Calatrava Volcanic Field	-3600	Mediterranean and Western Asia
Colli Albani	NA	Mediterranean and Western Asia
Campi Flegrei	1538	Mediterranean and Western Asia

mutate()

- Often, the raw data isn't analyzable in its current form!
- You will need to transform columns using `mutate()`
- `mutate()` creates new columns from existing columns, **or** changes existing columns
  - **very, very useful for manipulating your data!**

mutate()

- Uses similar syntax as summarise(): `name = function`

```
dataset %>%
  mutate(minutes = seconds / 60,
         hours = minutes / 60,
         days = hours / 24)
```

- You can use this to replace columns, not just add them:

```
# convert morning time to afternoon time
dataset %>%
  mutate(hours = hours+12)
```

Using mutate() to change variable types

- `mutate()` will be used by you for almost *everything*
  - For example - changing variable types:

```
# change a char column in a data frame to a factor column
people_factored <- people %>%
  mutate(humor = as.factor(humor))
```

The dataset will look the same, but there is an important difference:

```
people
```

```
         name age     humor
1       Alice  25       Bad
2        Spew  16      Good
3 Charlemagne  42      Good
4         Kay  18       Bad
5   Mackenzie   3  Terrible
6   Spirulina  16 Too Good
7       Jason  20      Good
```

```
class(people$humor)
```

```
[1] "character"
```

```
people_factored
```

```
         name age     humor
1       Alice  25       Bad
2        Spew  16      Good
3 Charlemagne  42      Good
4         Kay  18       Bad
5   Mackenzie   3  Terrible
6   Spirulina  16 Too Good
7       Jason  20      Good
```

```
class(people_factored$humor)
```

```
[1] "factor"
```

- Open 14_manipulate.Rmd
- Read **Arranging and Filtering Rows**

- Open 14_manipulate.Rmd
- Read **Selecting and Mutating Columns**

Joining Data Frames

When we have multiple data frames...

- Very often, we will have multiple data frames that reflect different data sources.
- For example:

1. A data frame with testing scores
2. A data frame with language background information
3. A data frame with demographic information

- Ideally, it is very easy to combine these data frames!
  - You need a column or set of columns that reflects **unique values** for each subject. This is your `key`.
    - Usually your `key` is your subject ID column!
  - Your `key` must have all unique values within a df. There cannot be two rows in a df with the same key!

Combining data frames using dplyr

```
1 ?dplyr::`mutate-joins`
```

- This family of joins takes two data frames, `x` and `y`, and matches them by a `key`:
  - `dplyr::inner_join()` removes all rows that don't have matching values in **both** dfs
  - `dplyr::left_join()` removes rows that aren't in **x** (most common)
  - `dplyr::right_join()` removes rows that aren't in **y**
  - `dplyr::full_join()` does not remove any rows

Choosing your key

```
combined_df <- left_join(df1,
                         df2,
                         by = "subj_id")
```

- This code adds rows in df2 that have a matching **subj_id** to df1.
- What if you need to match by multiple variables?
  - For example, what if each subject has two testing visits? No problem!

```
combined_df <- left_join(df1,
                         df2,
                         by = c("subj_id","num_visit"))
```

IF HAVE TIME...

- Open 14_manipulate.Rmd
- Read **Recoding Variables**