Week 2: Workflow

.R & .Rmd, Projects, Read Data

Joe Nese

University of Oregon

Fall 2021

Housekeeping

- covid
- assignments
- breaks

Workflow

Week 2

Agenda

- Project Oriented Workflow
  - RStudio Projects
  - File paths
  - Reading in data
  - Scripts (.R & .Rmd)
  - Loading packages
- Looking at data
- The pipe %>%

Learning Objectives

- Open an RStudio Project
- Understand file paths
- Learn two ways to read data
- Establish good workflow habits

RStudio Projects

RStudio Projects (.RProj)

An RStudio project - .Rproj - is basically a folder to house all the files for your project

- scripts
- products
- data
- figures

Advantages

- Work with several projects at the same time (several projects open)
    - these are self-contained!
- Can save the history of your commands after quitting
- Can specify version control system (e.g., Git)
- Previously edited code tabs are restored upon opening
- The current working directory is set to the project directory

What's a working directory?

Where you're at on your computer

You *can* change your working directory with `setwd("path/to/files")`...

...**BUT** I strongly urge you to avoid that

Instead, we're going to use RStudio Projects and the `{here}` package

You can `list.files(`*path*`)` to see the contents of your project directory (i.e., what your computer "sees")

- See where you're currently at by looking at the top of the console

What's different with .Rproj?

Your working directory is immediately wherever your project is located

This is true for whoever is accessing the project

Use the `{here}` package to specify **paths**

- You can read/save data (and figures, products, etc.) which we'll talk about more later

`setwd()`

Sets working directly

- `setwd()` ONLY works for you
- Can't share it out without A LOT of code editing
- `setwd()` makes it difficult to work on other projects with different directories
  - You have to remember to change it
  - Or work leaks across projects (objects, data, packages)

Instead: `here()`

`rm(list = ls())`

Removes objects in environment

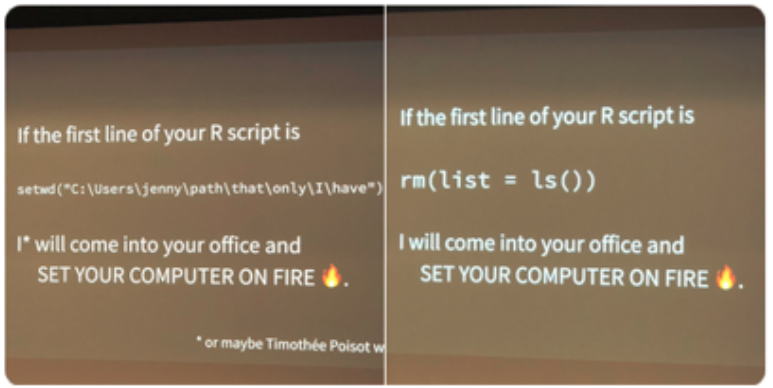
- Does NOT create a fresh R process
- Only deletes user-created objects
- Packages are still loaded!
- Options that have been reset in code still exist
- Your script is not self-contained!
- Not friendly to your collaborators! (deletes their objects)

Instead: **Session > Restart R**
(`Ctrl/Command + Shift + F10`)

Hadley Wickham ✓
@hadleywickham

Follow

The only two things that make @JennyBryan 😤😠🤯. Instead use projects + here::here() #rstats



4:50 PM - 10 Dec 2017

301 Retweets **993** Likes

62 301 993

Hadley Wickham ✓ @hadleywickham · 12 Dec 2017
Replying to @TimKeitt @JeromyAnglim @JennyBryan
I don't understand why you'd remove objects in an Rmd, which is always run in a fresh environment

Rob Knell @RobKnell1 · 10 Dec 2017
Replying to @hadleywickham @JennyBryan
Still haven't seen any explanation of why setwd() is so dreadful. In my work assuming that the person running the script will be using RStudio would cause a lot more trouble.

3 5

Hadley Wickham ✓ @hadleywickham · 10 Dec 2017
Because any use of setwd() presupposes a specific directory structure that is unlikely to exist on another computer

Project Oriented Workflow

Solution: Write every script assuming it will be run in a fresh `R` process

- Do not save `.RData` when you quit `R` and don't load `.RData` when you start `R`
- Daily work habit:
  - Restart `R` very often and re-run your developing script from the top
  - Use the RStudio menu item: **Session > Restart R** (`Ctrl/Command + Shift + F10`)

Workflow

Let’s start by making a new project

1. Make a new .RProj
2. Add folders
    - "data"
    - "scripts"
3. Read in data
4. Create scripts
5. Load packages

RStudio Project

1. Make a new .RProj

Let's name it: "my_first_project"

- notice the naming convention
  - no caps
  - no spaces
  - no special characters (e.g., ?, !, ", #)
  - "_" and "-" are ok

Choose a location for it

- An .RProj will need its own folder, with no other projects in it!
- Projects in Dropbox often lead to unexpected occurences

I'll just save "my_first_project" to my desktop

[demo]

Organize

2. Create folders

Let's make two folder in different ways (either is fine)

data

- where we will store all our project-related data
- let's create that in Rstudio

scripts

- where we will hold all our scripts (.R or .Rmd files)
- let's create that in the folder on our machine

[demo]

Reading Data

Dowload data

Let’s save the following data files into our project "data" folder

go here to download

1. ecls-k_samp.sav
2. Fatality.txt
3. Project_Reads_Scores.csv

Reading data into R

You just need one thing:

1. **where** the data is located

This is most often just a path on you machine

We'll be talking about two data reading packages

1. `{rio}`
2. `{readr}`

Data Location?

We'll use the `{here}` package

`{here}` uses the top-level directory of a project to easily build **paths** to files

- Allows you to worry about paths a little less
- Helps reproducibility
- Your final project!

Think of `here()` simply as a function to print a path

This **path** can be the **location** of your data

path = **location**

Art: Allison Horst

{here}

Think of `here()` simply as a function to print a path

First, install/load the package

```
# In the console:
# install.packages("here")
# Then, load the package
library(here)
```

{here}

Think of `here()` simply as a function to print a path

Run this code in your project (console is fine)

```
here()
```

This is the "top level" directory of our project

{here}

Think of `here()` simply as a function to print a path

Run this code in your project (console is fine)

```
here()
```

```
## [1] "C:/Users/jnese/Desktop/BRT/Teaching/Intro-Data-Science/c1-intro-fall-2
```

This is the "top level" directory of **my** project

{here}

Think of `here()` simply as a function to print a path

Run this code in your project (console is fine)

```
here()
```

```
here("data")
```

This is the path to the "data" folder in our project directory

{here}

Think of `here()` simply as a function to print a path

Run this code in your project (console is fine)

```
here()
```

```
here("data")
```

Question: What was the difference in the output between these?

Question: What will the following produce?

```
here("data", "ecls-k_samp.sav")
```

Let's see files are in our "data" folder

```
list.files(here("data"))
```

You can also use the *Files* tab in RStudio

Use `here()` to access data or any file in your project

`here()` is simply a function to print a path

[demo]

{rio}

- {rio} is a wrapper around many different packages that import/export data in different formats
- Great package
- Most of the time "it just works" regardless of the source file type
  - this might not impress you, but it really should!
  - any package that turns a complex task into a simple procedure and “just works” is invaluable

`import()`

`import(file, format, setclass, ...)`

`file` = character string naming a file

`format` = (**optional**) character string of file format; e.g., `","` for comma-separated values

`setclass` = (**optional**) character vector specifying one or more classes to set on the import. Default is `"data.frame"`. We would probably prefer `"tbl_df"` for a `tibble`

3a. Read in data

Try it!

```
library(rio)
```

```
## The following rio suggested packages are not installed: 'arrow', 'feather',
## Use 'install_formats()' to install them
```

```
# .sav
eclsk <- import(here("data", "ecls-k_samp.sav"), setclass = "tbl_df")

# .txt
fatality <- import(here("data", "Fatality.txt")) %>%
  as_tibble()

#.csv
exam1 <- import(here("data", "Project_Reads_Scores.csv")) %>%
  as_tibble()
```

You can even read directly from the web 😳

Fatality.txt

```
import("https://raw.githubusercontent.com/uo-datasci-specialization/c1-i
       setclass = "tbl_df")
```

```
## # A tibble: 336 x 10
##    state  year mrall beertax  mlda jaild comserd vmiles unrate perinc
##    <int> <int> <dbl>   <dbl> <dbl> <chr> <chr>    <dbl>  <dbl>  <dbl>
##  1     1  1982  2.13   1.54   19   no    no        7.23  14.4  10544.
##  2     1  1983  2.35   1.79   19   no    no        7.84  13.7  10733.
##  3     1  1984  2.34   1.71   19   no    no        8.26  11.1  11109.
##  4     1  1985  2.19   1.65   19.7 no    no        8.73   8.90 11333.
##  5     1  1986  2.67   1.61   21   no    no        8.95   9.80 11662.
##  6     1  1987  2.72   1.56   21   no    no        9.17   7.80 11944
##  7     1  1988  2.49   1.50   21   no    no        9.67   7.20 12369.
##  8     4  1982  2.50   0.215  19   yes   yes       6.81   9.90 12309.
##  9     4  1983  2.27   0.206  19   yes   yes       6.59   9.10 12694.
## 10     4  1984  2.83   0.297  19   yes   yes       6.71   5    13266.
## # ... with 326 more rows
```

`export()`

Save data just as easily with `export()`

You need two things:

1. What to export?
2. Where to export it?

```
export(x, file, format, ...)
```

`x` = data frame (tibble) to be written into a file

`file` = character string naming a file

export()

Save data just as easily with `export()`

You need two things:

1. What to export?
2. Where to export it?

`export(x, file, format, ...)`

`x` = data frame (tibble) to be written into a file

`file` = character string naming a file

```
export(exam1, here("data", "exam1.sav"))

export(exam1, here("data", "exam1.txt"))

export(exam1, here("data", "exam1.dta"))
```

`convert()`

Another really useful feature is `convert()`, which just takes a file of one type and converts it to another

Say your advisor uses SPSS, but their colleague uses Stata (and you use R 😎)

convert()

Another really useful feature is `convert()`, which just takes a file of one type and converts it to another

Say your advisor uses SPSS, but their colleague uses Stata (and you use R 😎)

Just run one line of code!

```
convert(in_file, out_file, ...)
```

`in_file` = character string naming an input file

`out_file` = character string naming an output file

```
convert(here("data",ecls-k_samp.sav)),
        (here("data",ecls-k_samp.dta))
```

How is this all working?

`{rio}` wraps a variety of faster, more stream-lined packages than those provided by base `R`

- `{data.table}` for delimited formats
- `{haven}` for SAS, Stata, and SPSS files
- `{readxl}` and `{openxlsx}` for reading and writing Excel workbooks
- `data.table::fread()` for text-delimited files to automatically determine the file format regardless of the extension
  - also *very* fast
- again, "just works"

Maintaining labels

Question: How many of you or your colleagues use SPSS or Stata?

In SPSS, numeric data are often encoded with labels

`{rio}` and `{haven}` allow you to transform the data into the character/factor version

`{rio}` and `{haven}` store metadata from rich file formats (SPSS, Stata, etc.) in variable-level attributes in a consistent form regardless of file type or underlying import function

- `rio::characterize()` converts a single variable or all variables in the data that have *label* attributes into character vectors
- `rio::factorize()` does the same but returns factor variables

```
eclsk %>%
  select(child_id, k_type:sex) %>%
  head()
```

```
## # A tibble: 6 x 4
##   child_id k_type school_type   sex
##   <chr>     <dbl>       <dbl> <dbl>
## 1 0842021C      1           0     0
## 2 0905002C      1           1     0
## 3 0150012C      1           1     1
## 4 0556009C      1           1     1
## 5 0089013C      1           0     0
## 6 1217001C      0           0     1
```

```
eclsk %>%
  select(child_id, k_type:sex) %>%
  head()
```

```
eclsk %>%
  characterize() %>%
  select(child_id, k_type:sex) %>%
  head()
```

```
## # A tibble: 6 x 4
##   child_id k_type   school_type sex
##   <chr>    <chr>    <chr>       <chr>
## 1 0842021C full-day public      male
## 2 0905002C full-day private     male
## 3 0150012C full-day private     female
## 4 0556009C full-day private     female
## 5 0089013C full-day public      male
## 6 1217001C half-day public      female
```

```
eclsk %>%
  select(child_id, k_type:sex) %>%
  head()
```

```
## # A tibble: 6 x 4
##   child_id k_type school_type   sex
##   <chr>     <dbl>       <dbl> <dbl>
## 1 0842021C      1           0     0
## 2 0905002C      1           1     0
## 3 0150012C      1           1     1
## 4 0556009C      1           1     1
```

{readr}

- Great package; most of the time "it just works" regardless of the source file type
- Loads with `{tidyverse}`
- Default loads data as a `tibble`
  - this is nice!

- `read_csv()`: comma separated (CSV) files
- `read_tsv()`: tab separated files
- `read_delim()`: general delimited files
- `read_fwf()`: fixed width files
- `read_table()`: tabular files where columns are separated by white-space
- `read_log()`: web log files

`read_csv()`

You just need one thing:

1. **where** the data is located

```
read_csv(file, ...)
```

`file` = a **path** to a file, a connection, or literal data (either a single string or a raw vector)

And how do we get a path string?

3b. Read in data

Try it!

```
library(tidyverse)

# {haven} package
eclsk <- haven::read_sav(here("data", "ecls-k_samp.sav"))

# read_table() for space separated data
fatality <- read_table(here("data", "Fatality.txt"))

#read_csv(), the one I most often use
exam1 <- read_csv(here("data", "Project_Reads_Scores.csv"))

# read directly from the web, in this case a .csv
web <- read_csv("https://github.com/datalorax/ncme_18/raw/master/data/pu
```

`write_*()`

Save data just as easily with `write_*()`

You need two things:

1. What to export?
2. Where to export it?

- For example:

```
write_csv(x, file, ...)
```

`x` = data frame (tibble) to be written into a file

`file` = character string naming a file to write to

write_*()

Save data just as easily with `write_*()`

You need two things:

1. What to export?
2. Where to export it?

- For example:

`write_csv(x, file, ...)`

`x` = data frame (tibble) to be written into a file

`file` = character string naming a file to write to

Basically

`write_*(what, "where")`

```
write_csv(exam1, here("data", "exam1.csv"))
```

Scripts

.R

.R is R script (code) file

File > New File > R Script

- Everything is code
- Text - comments! - need to begin with "#"
  - Comments are a **great** habit!
  - Use them to document the what & why of your code

Run code

- `Ctrl/Command + Enter`
- Highlight specific code, `Ctrl/Command + Enter`
- Put mouse on line, `Ctrl/Command + Enter`, and it will execute the all connected (piped) code

[demo]

Quick Peek

What is R Markdown?

- An authoring framework for data science
- A document format (.Rmd)
- An R package named `{rmarkdown}`
- A file format for making dynamic documents with R
- A tool for integrating prose, code, and results
- A computational document
- Everything

-

Source: Allison Hill

Quick Peek

What is R Markdown?

- An authoring framework for data science
- A document format (.Rmd)
- An R package named `{rmarkdown}`
- A file format for making dynamic documents with R
- A tool for integrating prose, code, and results
- A computational document
- Everything

-

Source: Allison Hill

.Rmd

.Rmd is a document format file that combines code **AND** prose

- *File > New File > R Markdown...*

Code goes into **"code chunks"**

````
```{r}
# comment

data %>%
  select(id, read, math)
```
````

Prose goes outside the code chunks

[demo]

.R & .Rmd

- Both are great
- Serve different purposes

Organization tip

- .Rmd with headers
- .R with `# Header ----`

[demo]

Packages

Packages

First the package must be *installed* on your machine

```
install.packages("package_name")
```

- you will only need to do this the first time you want to use the package
- **never** keep this code line (you can run it in the console, or delete it from your script)
- notices the quotes

Any time you want to use a package it should be *loaded*

```
library(package_name)
```

- you will do this each time you use the package in your scripts
- notices no quotes

{janitor}

A fantastic package!

Cleaning up common problems

- `remove_empty_rows()`
- `remove_empty_cols()`
- `excel_numeric_to_date()`
  - changes numeric dates imported from Excel to actual dates
- `tabyl()`
  - frequency table with *n* and %
- `clean_names()`
  - styles **column names** (NOT data itself) with "snake_case": lower case and underscore between words
  - can choose other "`*_case`"

Let's put it all together

New script

Let's work Within the "my_first_project" .RProj

(1) Open a new R Markdown file

[demo]

(2) Clean it up

- Modify the YAML
- Save the file as "practice.Rmd" in the **scripts** folder
- Knit it!

[demo]

practice.Rmd script

- Let's remove all chunks after the first (`setup`)
- This is what you will do with .Rmd going forward

Reading in data

(1) What packages will we use to read in data?

- `library(here)`
- `library(rio)`
- `library(tidyverse)`
- `library(janitor)`

```
##
## Attaching package: 'janitor'

## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

(2) Read in the Penguins.csv data

- Let's name the data "*penguins*"
- Three options

```
# rio::import()
penguins <- import(here("data", "Penguins.csv"), setclass = "tbl_df")
```

What do our data look like?

```
penguins
```

```
## # A tibble: 344 x 8
##    Species Island    `Bill Length (mm)` `Bill Depth (mm)` `Flipper Length (
##    <chr>   <chr>                  <dbl>             <dbl>                 <
##  1 Adelie  Torgersen                 39.1              18.7
##  2 Adelie  Torgersen                 39.5              17.4
##  3 Adelie  Torgersen                 40.3              18
##  4 Adelie  Torgersen                 NA                NA
##  5 Adelie  Torgersen                 36.7              19.3
##  6 Adelie  Torgersen                 39.3              20.6
##  7 Adelie  Torgersen                 38.9              17.8
##  8 Adelie  Torgersen                 39.2              19.6
##  9 Adelie  Torgersen                 34.1              18.1
## 10 Adelie  Torgersen                 42                20.2
## # ... with 334 more rows
```

Or use `View()` to take a look at the full data in RStudio

```
View(penguins)
```

clean_names()

```
# re-assign the "reads" object by reading the data in again
penguins <- read_csv(here("data", "Penguins.csv")) %>%
  clean_names()

# or just work with the existing "penguins" object
penguins <- penguins %>%
  clean_names()
```

Looking at the data structure

structure

```
str(penguins)
```

```
## spec_tbl_df [344 x 8] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ species          : chr [1:344] "Adelie" "Adelie" "Adelie" "Adelie" ...
##  $ island           : chr [1:344] "Torgersen" "Torgersen" "Torgersen" "Torg
##  $ bill_length_mm   : num [1:344] 39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.
##  $ bill_depth_mm    : num [1:344] 18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1
##  $ flipper_length_mm: num [1:344] 181 186 195 NA 193 190 181 195 193 190 ..
##  $ body_mass_g      : num [1:344] 3750 3800 3250 NA 3450 ...
##  $ sex              : chr [1:344] "male" "female" "female" NA ...
##  $ year             : num [1:344] 2007 2007 2007 2007 2007 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Species = col_character(),
##   ..   Island = col_character(),
##   ..   `Bill Length (mm)` = col_double(),
##   ..   `Bill Depth (mm)` = col_double(),
##   ..   `Flipper Length (mm)` = col_double(),
##   ..   `Body Mass (g)` = col_double(),
##   ..   Sex = col_character(),
```

Looking at the data properties

dimensions (rows × columns)

```
dim(penguins)
```

```
## [1] 344   8
```

```
nrow(penguins)
```

```
## [1] 344
```

```
ncol(penguins)
```

```
## [1] 8
```

head()

View the six first elements

```
head(penguins)
```

```
## # A tibble: 6 x 8
##   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mas
##   <chr>   <chr>              <dbl>         <dbl>             <dbl>       <d
## 1 Adelie  Torgersen           39.1          18.7               181        3
## 2 Adelie  Torgersen           39.5          17.4               186        3
## 3 Adelie  Torgersen           40.3          18                 195        3
## 4 Adelie  Torgersen           NA            NA                  NA
## 5 Adelie  Torgersen           36.7          19.3               193        3
## 6 Adelie  Torgersen           39.3          20.6               190        3
```

```
head(penguins$flipper_length_mm)
```

```
## [1] 181 186 195  NA 193 190
```

Another cool package {skimr}

```
library(skimr)

skim(penguins)
```

[demo]

The pipe operator (%>%)

- The %>% operator (Super + Shift + M)
  - inserts the input as the first argument in the next function
- To start, you can read it as "then"
- It is crucial for work in the {tidyverse}

```
penguins %>%
  count(species)
```

```
## # A tibble: 3 x 2
##   species       n
##   <chr>     <int>
## 1 Adelie      152
## 2 Chinstrap    68
## 3 Gentoo      124
```

Let's look at ?count and ?tabyl

Or with {janitor}

```
penguins %>%
  tabyl(species)
```

```
##    species   n   percent
##     Adelie 152 0.4418605
##  Chinstrap  68 0.1976744
##     Gentoo 124 0.3604651
```

Why use %>%

Chaining arguments is **efficient** and **easy to read**

```
penguins %>%
  filter(species == "Adelie",
         bill_length_mm > 40) %>%
  select(island, bill_length_mm, body_mass_g) %>%
  arrange(bill_length_mm) %>%
  slice(1:3)
```

```
## # A tibble: 3 x 3
##   island    bill_length_mm body_mass_g
##   <chr>              <dbl>       <dbl>
## 1 Biscoe              40.1        4300
## 2 Torgersen           40.2        3450
## 3 Dream               40.2        3975
```

Equivalent to:

```
slice(arrange(select(filter(penguins, species == "Adelie", bill_length_m
```

The %>% cont.

The %>% works so well in the {tidyverse} because the first argument in (nearly) all functions is the data (tibble)

So you don't need to name the data each time

So this:

```
penguins %>%
  count(sex)
```

```
## # A tibble: 3 x 2
##   sex        n
##   <chr>  <int>
## 1 female   165
## 2 male     168
## 3 <NA>      11
```

Is equivalent to this:

```
count(penguins, sex)
```

```
## # A tibble: 3 x 2
##   sex        n
##   <chr>  <int>
## 1 female   165
## 2 male     168
## 3 <NA>      11
```

Next time

Homework Notes

Script on website

- Download **Homework 1** from the course Schedule or Assignments page
- Work with this .Rmd file

Submit a rendered .html file to Canvas

- *Assignments* > *Homeworks* > *HW 1*

You **may** need to install `{tinytex}` to get your .Rmd file to render to an .html file

- If you want to try this, read the embedded link above first
- If you are having trouble contact me

Before next class

- Reading
  - R4DS Ch 3
- Supplemental Learning
  - RStudio Primer: Data Visualization Basics
  - Rbootcamp: Ch 2
  - Codecademy: Introduction to Visualization with R
- Homework
  - **Homework 1**
- Final project

  - Finalize Groups

How do we access variables?

- Generally, in this course, with `{tidyverse}` tools
- Sometimes with `$` or with `[]`
  - `data_name$variable_name`
  - `data_name["variable_name"]`

Selecting varaibles

Select the `species` variable with `$`

```
penguins$species
```

```
##   [1] "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"
##  [12] "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"
##  [23] "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"
##  [34] "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"
##  [45] "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"
##  [56] "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"
##  [67] "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"
##  [78] "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"
##  [89] "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"
## [100] "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"
## [111] "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"
## [122] "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"
## [133] "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"
## [144] "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"    "Adelie"
## [155] "Gentoo"    "Gentoo"    "Gentoo"    "Gentoo"    "Gentoo"    "Gentoo"
## [166] "Gentoo"    "Gentoo"    "Gentoo"    "Gentoo"    "Gentoo"    "Gentoo"
## [177] "Gentoo"    "Gentoo"    "Gentoo"    "Gentoo"    "Gentoo"    "Gentoo"
## [188] "Gentoo"    "Gentoo"    "Gentoo"    "Gentoo"    "Gentoo"    "Gentoo"
```

Look at the `str`ucture of a variable

```
str(penguins$species)
```

```
##  chr [1:344] "Adelie" "Adelie" "Adelie" "Adelie" "Adelie" "Adelie" "Adelie"
```

Or and object

```
str(penguins)
```

```
## spec_tbl_df [344 x 8] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ species          : chr [1:344] "Adelie" "Adelie" "Adelie" "Adelie" ...
##  $ island           : chr [1:344] "Torgersen" "Torgersen" "Torgersen" "Torg
##  $ bill_length_mm   : num [1:344] 39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.
##  $ bill_depth_mm    : num [1:344] 18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1
##  $ flipper_length_mm: num [1:344] 181 186 195 NA 193 190 181 195 193 190 ..
##  $ body_mass_g      : num [1:344] 3750 3800 3250 NA 3450 ...
##  $ sex              : chr [1:344] "male" "female" "female" NA ...
##  $ year             : num [1:344] 2007 2007 2007 2007 2007 ...
##  - attr(*, "spec")=
##   .. cols(
```

Because penguins is a tibble

```
penguins
```

```
## # A tibble: 344 x 8
##    species island    bill_length_mm bill_depth_mm flipper_length_mm body_ma
##    <chr>   <chr>              <dbl>         <dbl>             <dbl>       <
##  1 Adelie  Torgersen           39.1          18.7               181
##  2 Adelie  Torgersen           39.5          17.4               186
##  3 Adelie  Torgersen           40.3          18                 195
##  4 Adelie  Torgersen           NA            NA                  NA
##  5 Adelie  Torgersen           36.7          19.3               193
##  6 Adelie  Torgersen           39.3          20.6               190
##  7 Adelie  Torgersen           38.9          17.8               181
##  8 Adelie  Torgersen           39.2          19.6               195
##  9 Adelie  Torgersen           34.1          18.1               193
## 10 Adelie  Torgersen           42            20.2               190
## # ... with 334 more rows
```

When to use $?

- Often when you need to use {base} R
- Execute a function casually

```
table(penguins$species)
```

```
##
##    Adelie Chinstrap    Gentoo
##       152        68       124
```

```
hist(penguins$body_mass_g)
```

Your turn

Run `table()` on the `island` variable

Produce a `hist`ogram of the `bill_length_mm` variable

```
xaringanBuilder::build_pdf(here::here("slides", "w2_workflow.Rmd"), comp
```