An Introduction to R and RStudio for Exploratory Data Analysis

Jessica Minnier, PhD & Meike Niederhausen, PhD
OCTRI Biostatistics, Epidemiology, Research & Design (BERD) Workshop

Part 1: 2020/09/16 & Part 2: 2020/09/17

slides: bit.ly/berd_intro_part1
pdf: bit.ly/berd_intro_part1_pdf

Do this now:

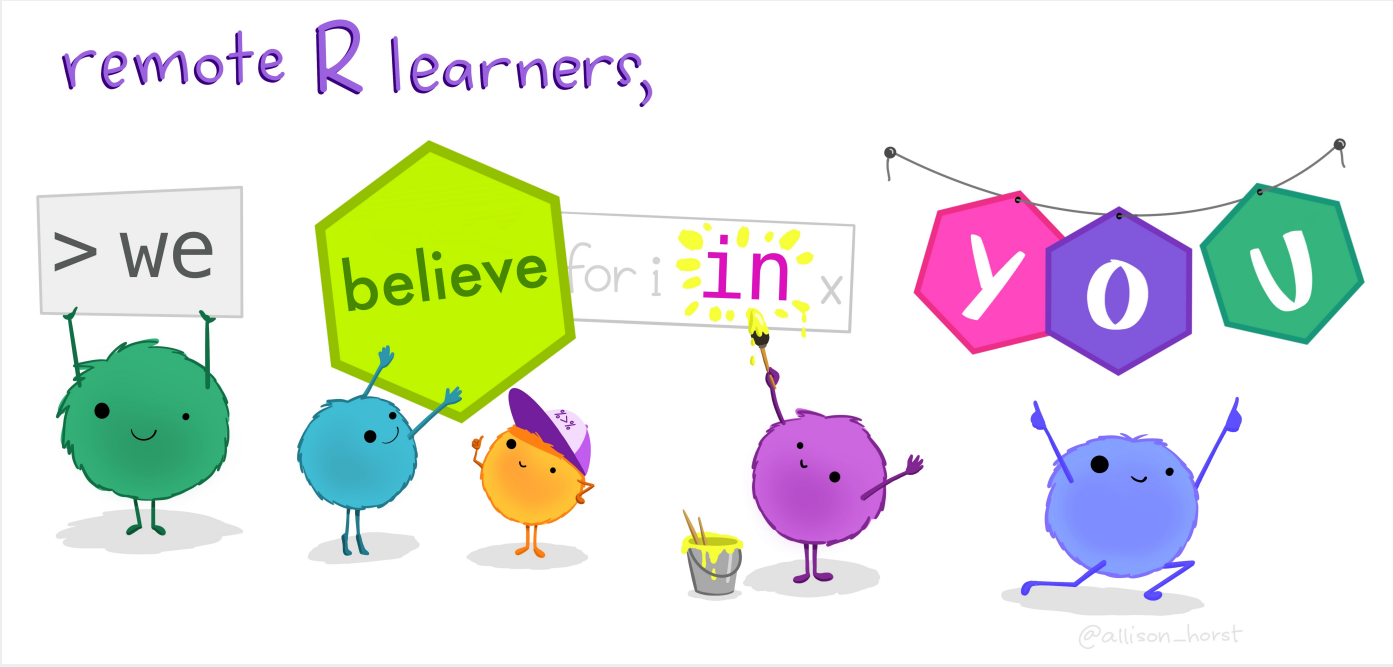
1. **Open html slides**: bit.ly/berd_intro_part1
2. Make sure you have **installed R & Rstudio**: instructions here bit.ly/berd_install
3. **Open google doc** for asking questions: bit.ly/berd_doc

Zoom rules:

1. **Change your name in Zoom** to a made up name/animal/word if you *do not want your name in recording* (show participants list, next to your name at the top there are options to rename yourself). **Turn off your video.**
2. **No private messages to instructors**, we won't see them. **Chat message "Helpers" for help or to go to a breakout room**. The # of your breakout room corresponds to "your" helper. During breaks and exercises, helpers will be in breakout rooms.
3. **Breakout rooms** are for getting help with R issues or with exercises. You won't be able to see what is going on in the main room while you are in your breakout room. You can stay in main room during exercises if you wish, and can ask questions to the presenters in the main room during that time (unmute and speak, or in chat).

Learning Objectives

- Basic operations in R/RStudio
- Understand data structures
- Be able to load in data
- Basic operations on data
- Some data wrangling
- Use Rstudio projects
- Be able to make a plot
- Basics of tidyverse and ggplot
- Know how to get help



Allison Horst

Introduction

Rrrrrr?

What is R?

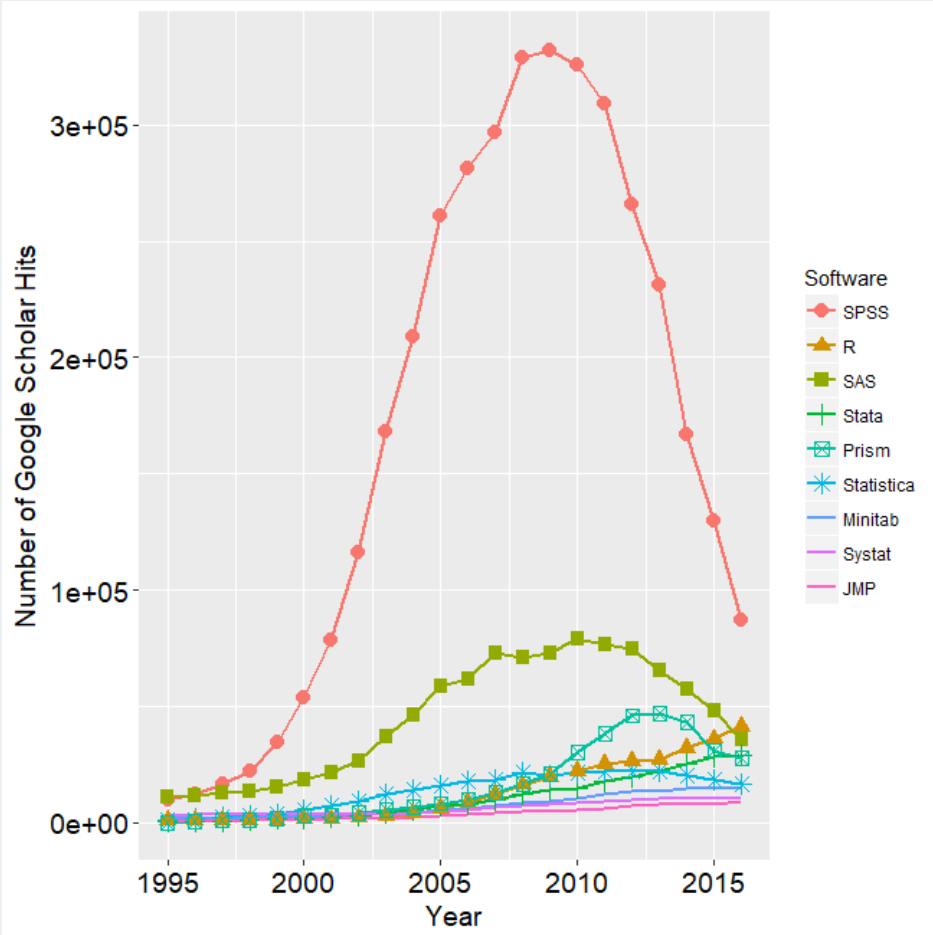
- A programming language
- Focus on statistical modeling and data analysis
  - import data, manipulate data, run statistics, make plots
- Useful for "Data Science"
- Great visualizations
- Also useful for most anything else you'd want to tell a computer to do
- Interfaces with other languages i.e. python, C++, bash

For the history and details: Wikipedia

- an interpreted language (run it through a command line)
- procedural programming with functions
- Why "R"?? Scheme (?) inspired S (invented at Bell Labs in 1976) which inspired R (**free and open source!** in 1992)

Why R?

- Free + Cross-platform (Mac/Windows)
- Flexible, fun, many more modern statistics methods, large community for learning and help
- One of the most popular data science tools for statistics in academia and industry
- SAS and STATA (and SPSS) are still used but becoming less popular (expensive, not as versatile/comprehensive)
- Constantly evolving and improving
- If you want a job doing stats and not be limited to specific research groups or some pharma companies, you absolutely *need to know R*



r4stats Robert A. Muenchen

What is RStudio?

R is a programming language

RStudio is an integrated development environment (IDE) = an interface to use R (with perks!)

- R is like a car's engine
- RStudio is like a car's dashboard

R: Engine	RStudio: Dashboard
	

Modern Dive

Start RStudio

1.1.2 Using R via RStudio

Recall our car analogy from earlier. Much as we don't drive a car by interacting directly with the engine but rather by interacting with elements on the car's dashboard, we won't be using R directly but rather we will use RStudio's interface. After you install R and RStudio on your computer, you'll have two new *programs* (also called *applications*) you can open. We'll always work in RStudio and not in the R application. Figure 1.2 shows what icon you should be clicking on your computer.

R: Do not open this	RStudio: Open this
	

Modern Dive

BuzzR

RStudio anatomy

https://buzzrbeeline.blog/

Emma Rand

Script file

Write code here

To run code put your cursor on the line and click the run button

Edit to correct errors

⇒ record of commands that worked

Save scripts with the **.R** extension

⇒ syntax will be highlighted

⇒ good practice

`<-` is the assignment operator

⇒ puts what is on the right in to the object on the left

⇒ Assign results if you want to use them again

Console

When you click run, code is sent to the console and executed

`>` is the prompt

⇒ do not type it

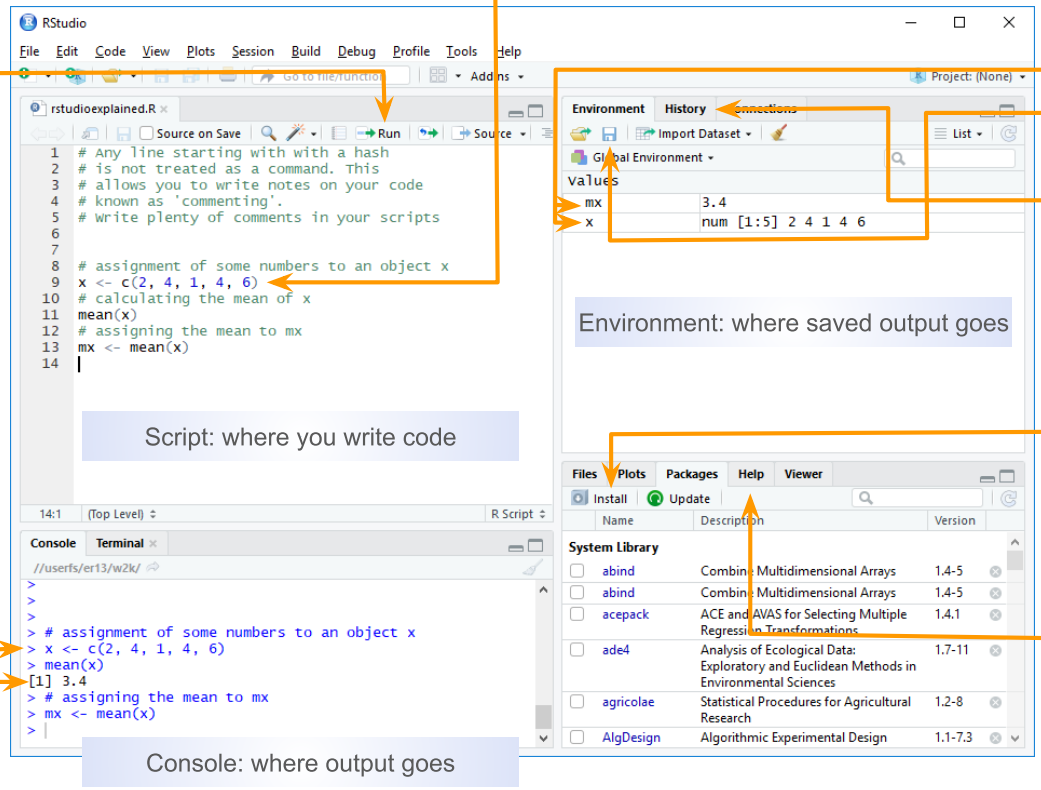
⇒ appears when R is ready for next command

Command output goes here by default

⇒ output is in a different colour

⇒ `[1]` indicates 3.4 is the first element of the output

⇒ many commands will not have output, the prompt just reappears



Environment

Name objects by assignment to use them again

All the objects you created in your session

Saving the environment saves all the objects, but not the code with a **.RData** extension

History

A history of every command you sent to the console, mistakes included.

File can be saved but usually you just need the script

Packages

Many functions come with R

A huge amount of extra functionality is available in packages

Packages can be installed by clicking the Install button

Help

Access to manual pages for all installed packages

Plots

Figure output appears here

Emma Rand

RStudio demo

- Start RStudio and explore

Bonus lessons

- gifs showing how to adjust panels, personalize how Rstudio looks, etc

Installing and using packages

- Packages are to R like apps are to your phone/OS
- Packages contain additional functions and data
- Install packages with `install.packages()`
  - Also can use the "Packages" tab in Files/Plots/Packages/Help/Viewer window
  - *Only install once (unless you want to update)*
  - Installs from Comprehensive R Archive Network (CRAN) = package mothership

```
install.packages("dplyr")   # only do this ONCE, use quotes
```

- Load packages: At the top of your script include `library()` commands to load each required package *every* time you open Rstudio.

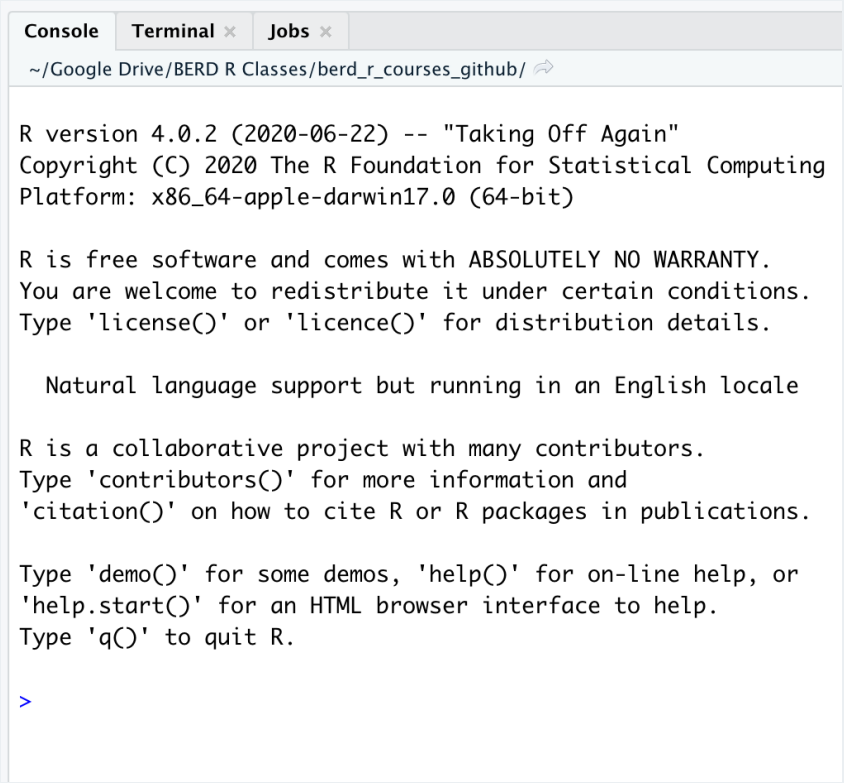
```
library(dplyr)    # run this every time you open Rstudio
```

Let's code!

R Basics

Coding in the console

When you first open R, the console should be empty.



Typing and executing code in the console

- Type code in the console (blue text)
- Press **return** to execute the code
- Output shown below in black

```
> 7
[1] 7
> 3 + 5
[1] 8
> "hello"
[1] "hello"
> # this is a comment, nothing happens
> # 5 - 8
> # separate multiple commands with ;
> 3 + 5; 4 + 8
[1] 8
[1] 12
>
```

We can do math

```
> 10^2
```

```
[1] 100
```

```
> 3 ^ 7
```

```
[1] 2187
```

```
> 6/9
```

```
[1] 0.6666667
```

```
> 9-43
```

```
[1] -34
```

- Rules for order of operations are followed
- Spaces between numbers and characters are ignored

```
> 4^3-2* 7+9 /2
```

```
[1] 54.5
```

The equation above is computed as

$$4^3 - (2 \cdot 7) + \frac{9}{2}$$

Variables

Variables are used to store data, figures, model output, etc.

- Can assign a variable using either = or <-
  - Using <- is preferable
  - type name of variable to print

Assign just one value:

```
> x = 5
> x
```

```
[1] 5
```

```
> x <- 5
> x
```

```
[1] 5
```

Assign a **vector** of values:

- Consecutive integers using :

```
> a <- 3:10
> a
```

```
[1]  3  4  5  6  7  8  9 10
```

- **Concatenate** a string of numbers

```
> b <- c(5, 12, 2, 100, 8)
> b
```

```
[1]   5  12   2 100   8
```

We can do math with variables

Math using variables with just one value

```
> x <- 5
> x
```

```
[1] 5
```

```
> x + 3
```

```
[1] 8
```

```
> y <- x^2
> y
```

```
[1] 25
```

Math on vectors of values: **element-wise** computation

```
> a <- 3:6
> a
```

```
[1] 3 4 5 6
```

```
> a+2; a*3
```

```
[1] 5 6 7 8
```

```
[1]  9 12 15 18
```

```
> a*a
```

```
[1]  9 16 25 36
```

Variables can include text (characters)

```
> hi <- "hello"
> hi
```

```
[1] "hello"
```

```
> greetings <- c("Guten Tag", "Hola", hi)
> greetings
```

```
[1] "Guten Tag" "Hola"      "hello"
```

Using functions

- `mean()` is an example of a function
- functions have "arguments" that are specified within the `()`
- `?mean` in console will show help for `mean()`

Arguments specified by name:

```
> mean(x = 1:4)
```

```
[1] 2.5
```

```
> seq(from = 1, to = 12, by = 3)
```

```
[1]  1  4  7 10
```

```
> seq(by = 3, to = 12, from = 1)
```

```
[1]  1  4  7 10
```

Arguments not specified, but listed in order:

```
> mean(1:4)
```

```
[1] 2.5
```

```
> seq(1,12,3)
```

```
[1]  1  4  7 10
```

Missing values

Missing values are denoted as `NA` and are handled differently depending on the operation. There are special functions for `NA` (i.e. `is.na()`, `na.omit()`).

```
> x <- c(1, 2, NA, 5)
> is.na(x)
```

```
[1] FALSE FALSE  TRUE FALSE
```

```
> mean(x)
```

```
[1] NA
```

```
> mean(x, na.rm=TRUE)
```

```
[1] 2.666667
```

```
> x <- c("a", "a", NA, "b")
> table(x)
```

```
x
a b
2 1
```

```
> table(x, useNA = "always")
```

```
x
   a    b <NA>
   2    1    1
```

Common console errors (1/2)

Incomplete commands

- When the console is waiting for a new command, the prompt line begins with >
  - If the console prompt is +, then a previous command is incomplete
  - You can finish typing the command in the console window

Example:

```
> 3 + (2*6
+ )
```

```
[1] 15
```

Common console errors (2/2)

Object is not found

- This happens when text is entered for a non-existent variable (object)

Example:

```
> hello
```

```
Error in eval(expr, envir, enclos): object 'hello' not found
```

- Can be due to missing quotes

```
> install.packages(dplyr) # need install.packages("dplyr")
```

```
Error in install.packages(dplyr): object 'dplyr' not found
```

Create an R Markdown file (`.Rmd`)

Save the Markdown file (`.Rmd`)

- **Save the file** by
  - selecting `File -> Save`,
  - or clicking on  (towards the left above the scripting window),
  - or keyboard shortcut
    - PC: *Ctrl + s*
    - Mac: *Command + s*

- You will need to specify
  - a **filename** to save the file as
    - ALWAYS use **.Rmd** as the filename extension for R markdown files
  - the **folder** to save the file in

Compare the .Rmd file with its html output

.Rmd file

default_html.Rmd

Knit | Insert | Run

```
---
title: "Untitled"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```{r cars}
summary(cars)
```

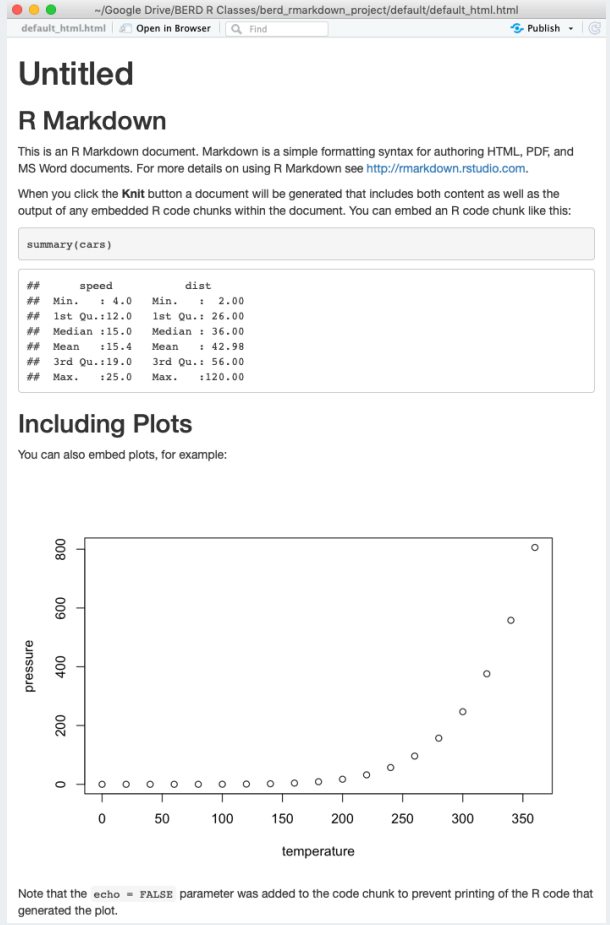
Including Plots

You can also embed plots, for example:

```{r pressure, echo=FALSE}
plot(pressure)
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.
```

html output



Compare the .Rmd file with its html output

.Rmd file

default_html.Rmd

```
---
title: "Untitled"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```{r cars}
summary(cars)
```

Including Plots

You can also embed plots, for example:

```{r pressure, echo=FALSE}
plot(pressure)
```

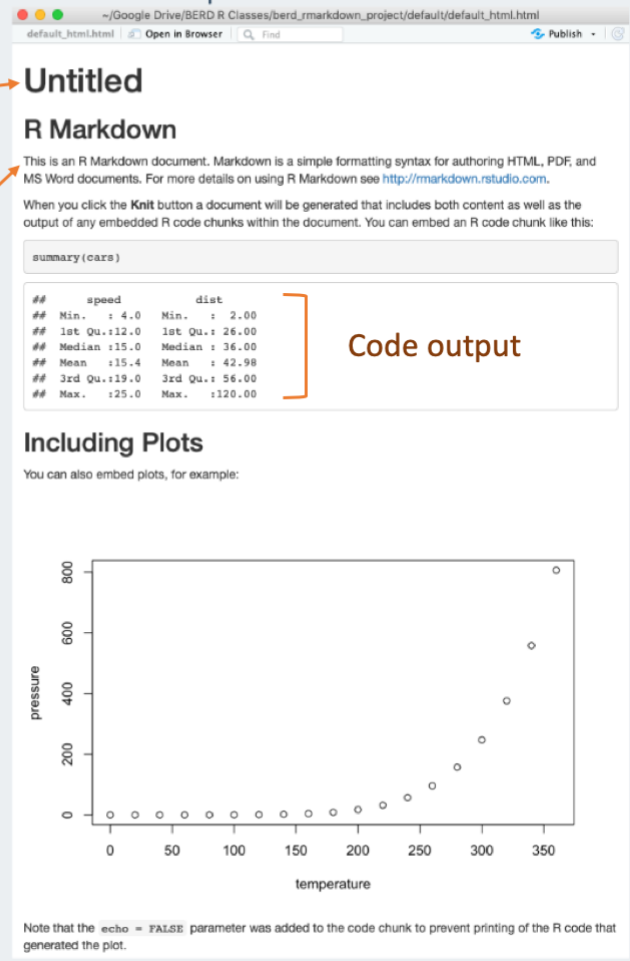
Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.
```

YAML metadata

Text

Code chunk

html output



How to create the html file? *Knit* the .Rmd file!

To **knit** the .Rmd file, either

1. click on the knit icon  at the top of the editor window
2. or use keyboard shortcuts
    - Mac: *Command+Shift+K*
    - PC: *Ctrl+Shift+K*

- A new window will open with the html output.
- You will now see both .Rmd and .html files in the folder where you saved the .Rmd file.

Remarks:

- The template .Rmd file that RStudio creates will knit to an html file by default
- Watch the Reproducible Reports with R Markdown workshop for more customization and output formats (Word, pdf, slides).
    - Slides at https://jminnier-berd-r-courses.netlify.com/03-rmarkdown/03_rmarkdown_slides.html.

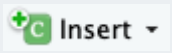
3 types of R Markdown content

1. *Code chunks*: type R code and execute it to see code output
2. Text: write about your analyses
3. YAML metadata: customize the report

We will first focus on using code chunks.

Create a code chunk

Code chunks can be created by either

1. Clicking on  →  at top right of the editor window, or
2. **Keyboard shortcut**
    - Mac: *Command + Option + I*
    - PC: *Ctrl + Alt + I*

- An empty code chunk looks like this:

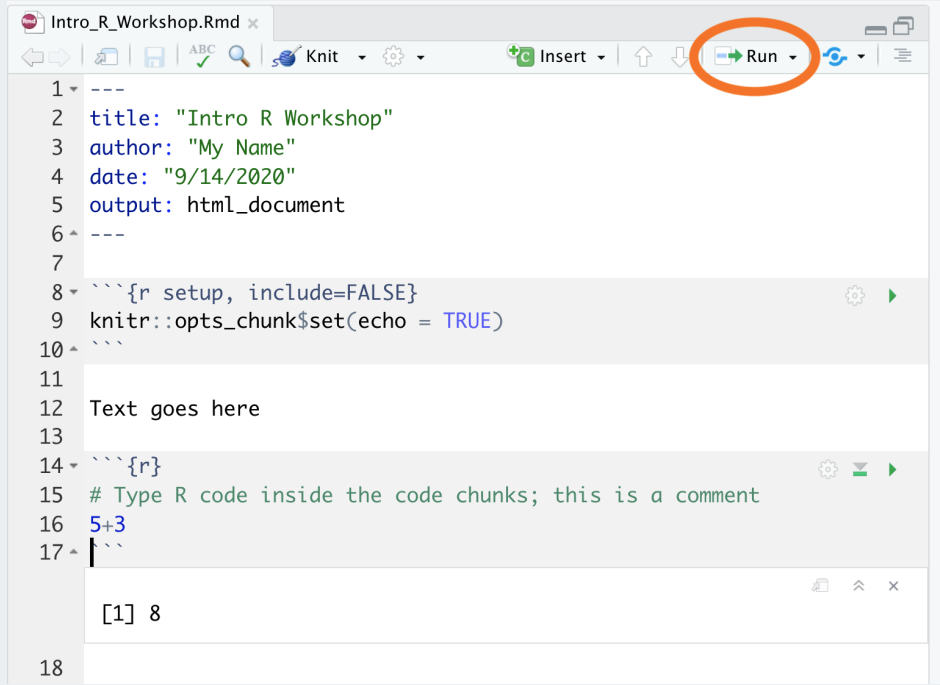
````
```{r}

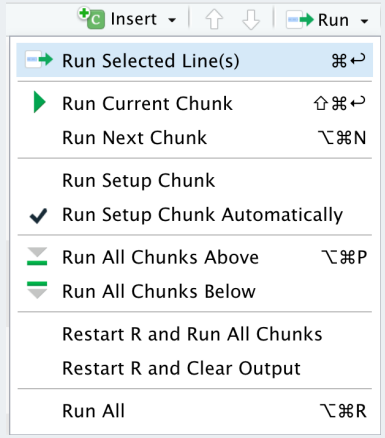

```
````

- Note that a code chunks start with ```` ```{r} ```` and ends with ```` ``` ````.

Enter and run code (1/n)

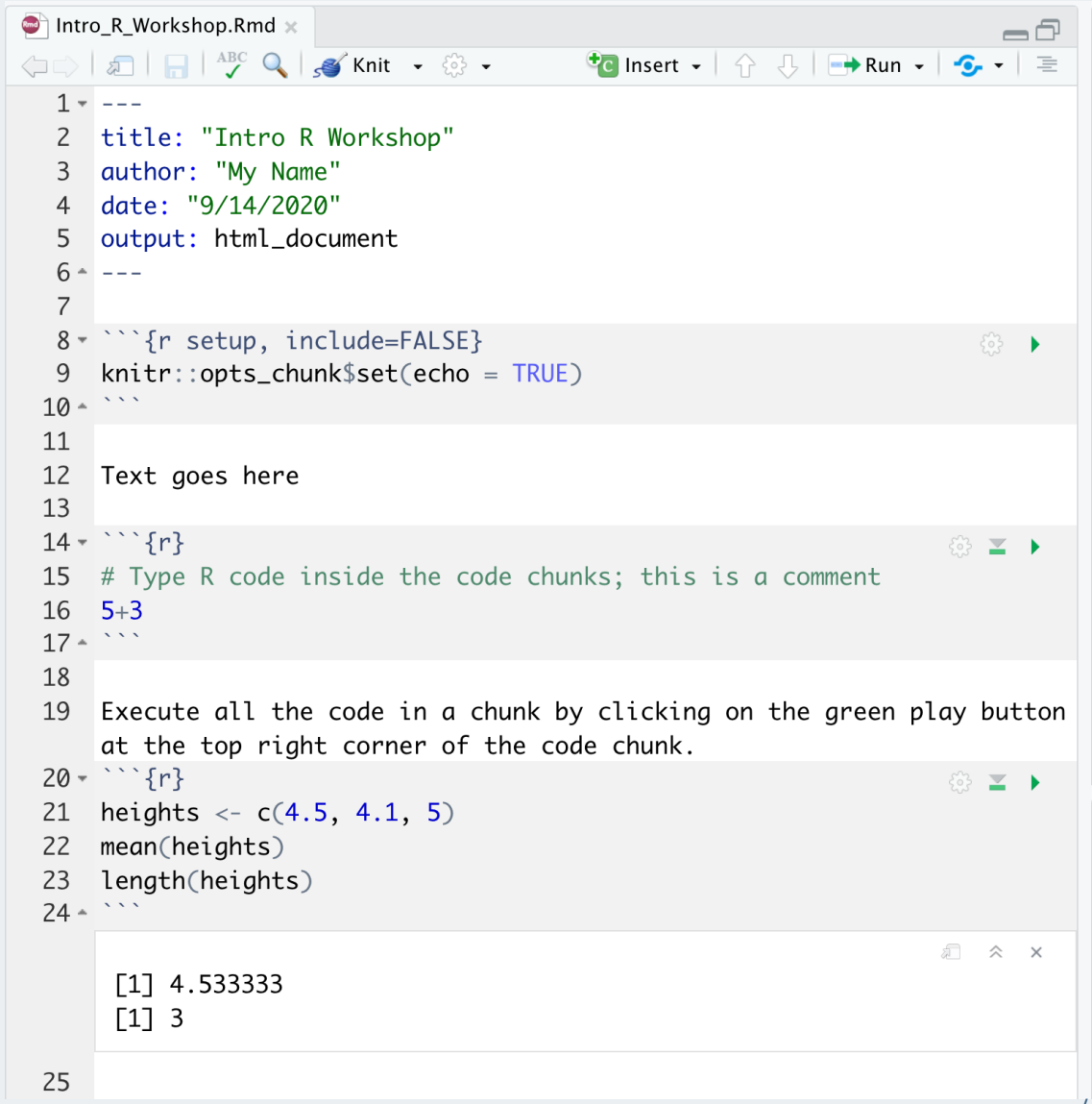
- **Type R code** inside code chunks
- **Select code** you want to run, by
  - placing the cursor in the line of code you want to run,
  - ***or*** highlighting the code you want to run
- **Run selected code** by
  - clicking on the Run button in the top right corner of the scripting window and choosing "Run Selected Line(s)",
  - or typing one of the following key combinations:
    - **Windows**: **ctrl + return**
    - **Mac**: **command + return**





Enter and run code (2/n)

- **Run all code** in a chunk by
  - by clicking the play button in the top right corner of the chunk
- The code output appears below the code chunk



Useful keyboard shortcuts

action	mac	windows/linux
Run code in Rmd or script	cmd + enter	ctrl + enter
<-	option + -	alt + -

Try typing in Rmd (with shortcut) and running

```
y <- 5
y
```

Others: (see full list)

action	mac	windows/linux
interrupt currently executing command	esc	esc
in console, go to previously run code	up/down	up/down
keyboard shortcut help	option + shift + k	alt + shift + k

Practice time!

Practice 1

1. Create a new Rmd file and type the code and answers for tasks below in it. Save as `Practice1.Rmd`
2. Remove any template text you don't need (starting with line 12), and create a new code chunk.
3. Create a vector of all integers from 4 to 10, and save it as `a1`.
4. Create a vector of *even* integers from 4 to 10, and save it as `a2`.
5. What do you get when you add `a1` to `a2`?
6. What does the command `sum(a1)` do?
7. What does the command `length(a1)` do?
8. Use the `sum` and `length` commands to calculate the average of the values in `a1`.
9. Knit the Rmd file.

We will be using functions within the `tidyverse` package in R.

Intro to Data

How is data stored, how do we use it?

- Often, data is in an excel sheet, or a plain text file (.csv, .txt)
- .csv files open in Excel automatically, but actually are plain text
- Usually, columns are variables/measures and rows are observations (i.e. a person's measurements)

Our example data:

Download data csv file link and pay attention to *where* it downloads on your computer

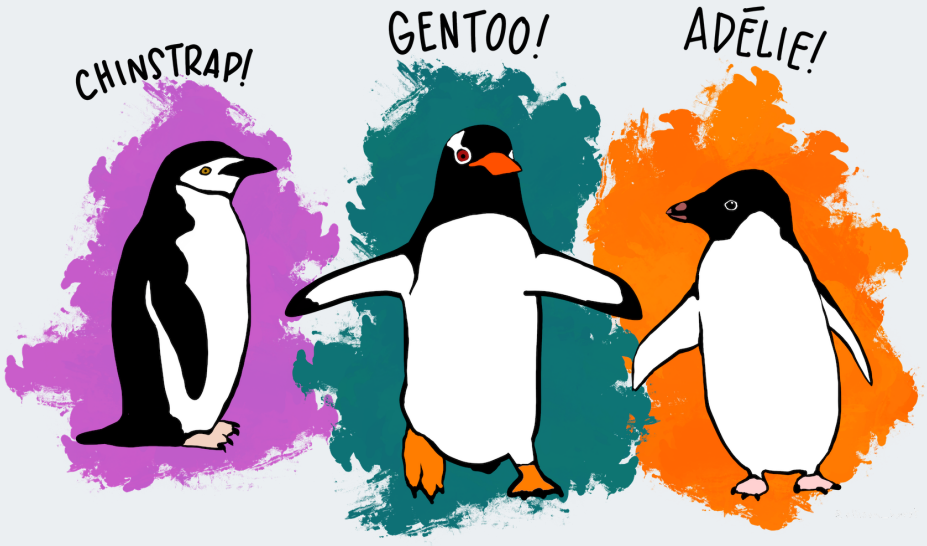
- Make sure it is a .csv file and not a "web archive" or something else.

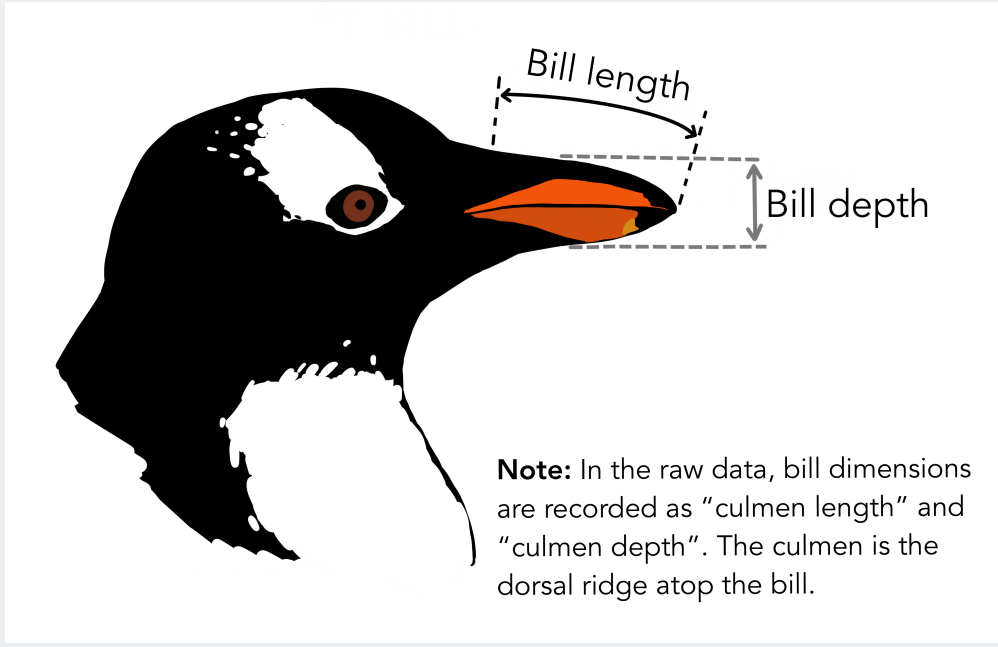
Open the data file `penguins.csv` and look at it

- What are the columns? What are the rows?

About the penguins data

- A data set about penguins at Palmer Station, Antarctica! More info at github.com/allisonhorst/palmerpenguins
- Data were collected and made available by Dr. Kristen Gorman and the Palmer Station, Antarctica LTER, a member of the Long Term Ecological Research Network.
- Each row is a penguin measurement
- Some false missingness was induced for this practice.



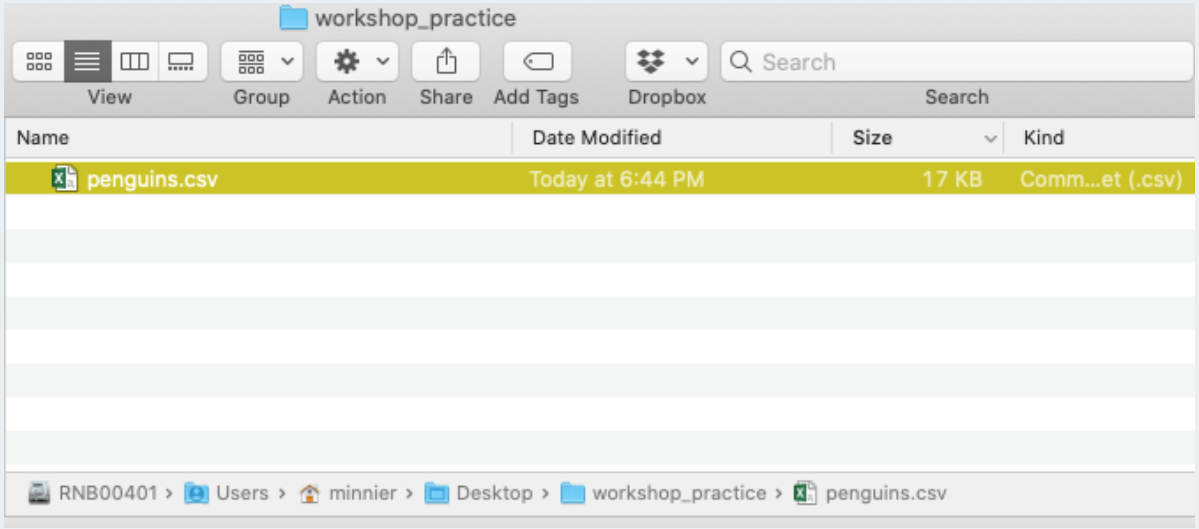


Workflow - Keep it together!

Steps for a new data analysis project or homework:

1. Create a folder to contain all your files.
2. Move data file (`penguins.csv`) into this folder.
3. Create an Rstudio project inside this folder. (next slides)
4. Create a new Rmd for your analyses/homework.

Do steps 1 & 2 now!



R Projects (.Rproj file) & Good Practices

Use projects to keep everything together (read this)

- A project keeps track of your coding environment and file structure.
- Create an RStudio project for each data analysis project, for each homework assignment, etc.
- A project is associated with a directory folder
  - Keep data files there
  - Keep code scripts there; edit them, run them in bits or as a whole
  - Save your outputs (plots and cleaned data) there
- Only use relative paths, never absolute paths
  - relative (good): `read.csv("data/mydata.csv")`
  - absolute (bad): `read.csv("/home/yourname/Documents/stuff/mydata.csv")`

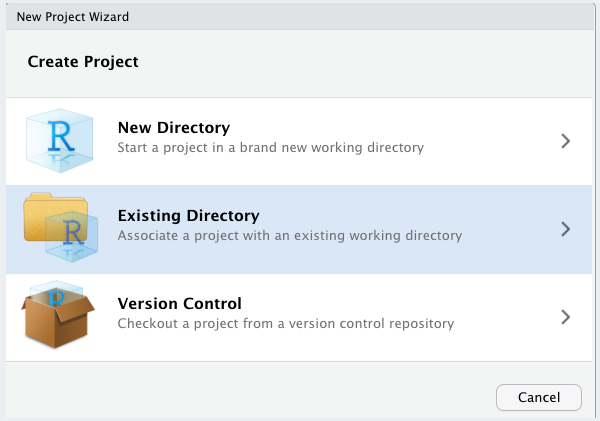
Advantages of using projects

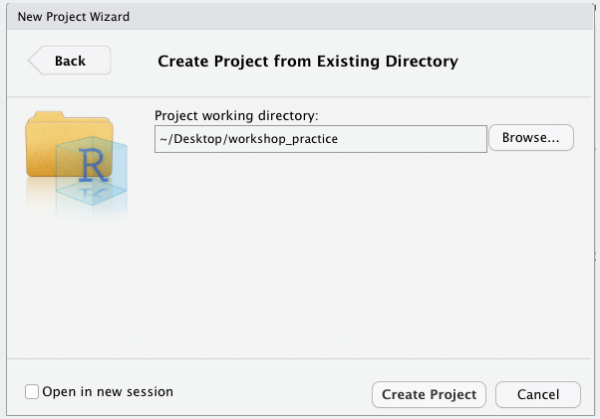
- standardizes file paths
- keep everything together
- a whole folder can be easily shared and run on another computer
- when you open the project everything is as you left it

Create a new R project

Let's go through it together. (Read this for more)

- Click  in top left *or* File -> New Project
- Click *Existing Directory*
- Browse to your folder with the data
- *Optional* Click "Open in new session checkbox"
- Click "Create project"



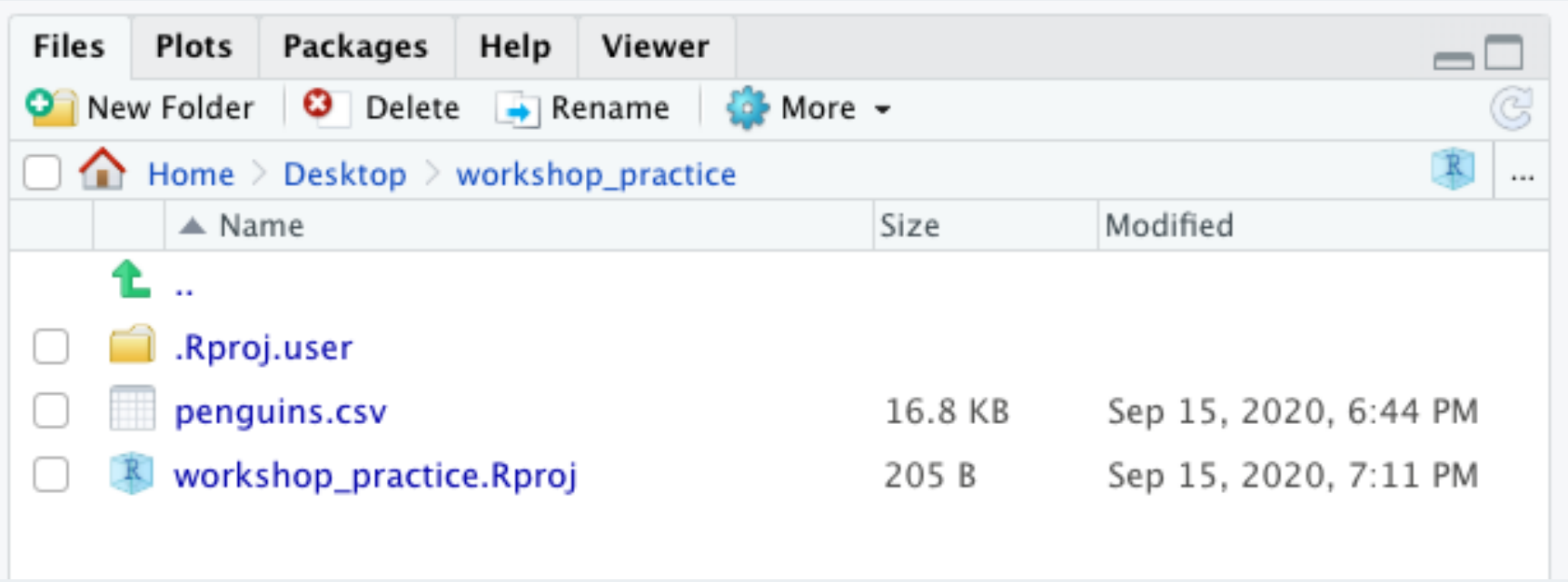


Bonus lessons

- Video on projects in R, most useful info in minutes 2:00-13:00

The data file will be in your Files pane:

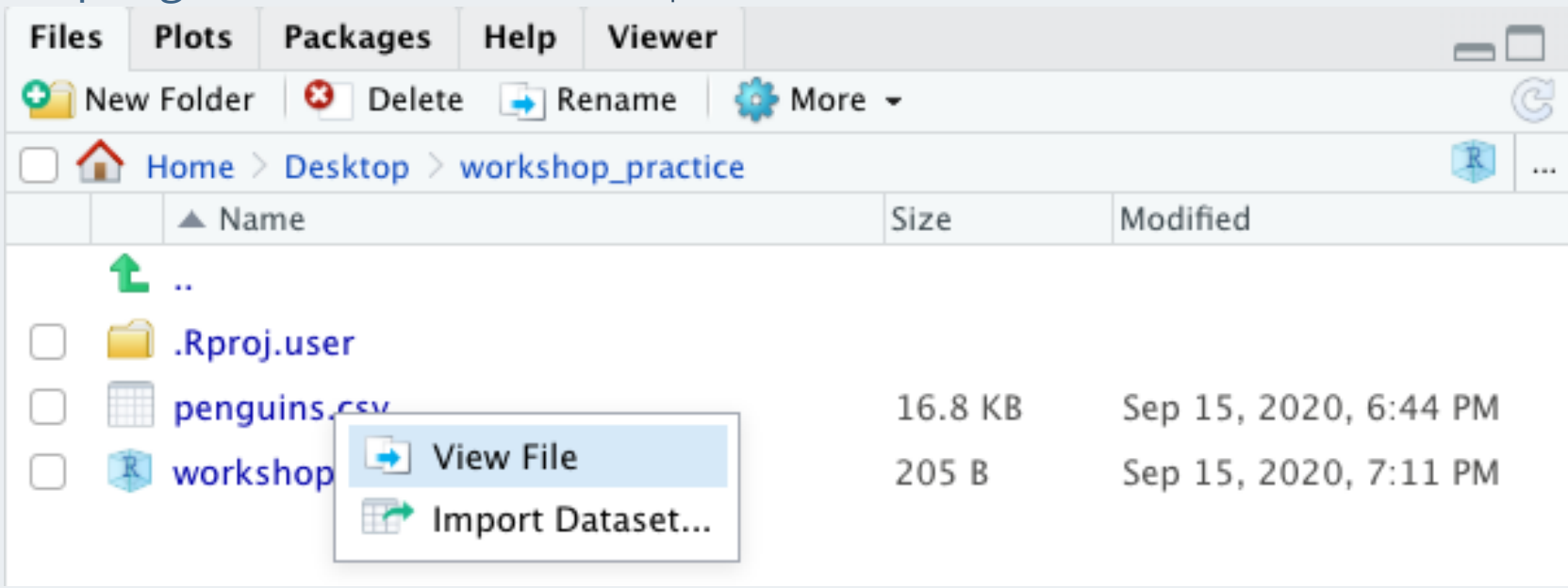
and your workspace folder location will be showing at the top (i.e. `Home/Desktop/workshop_practice`)



Data in R/Rstudio

Open `penguins.csv` in Rstudio and look at it

- Click on `penguins.csv` in the Files pane, click *View File*

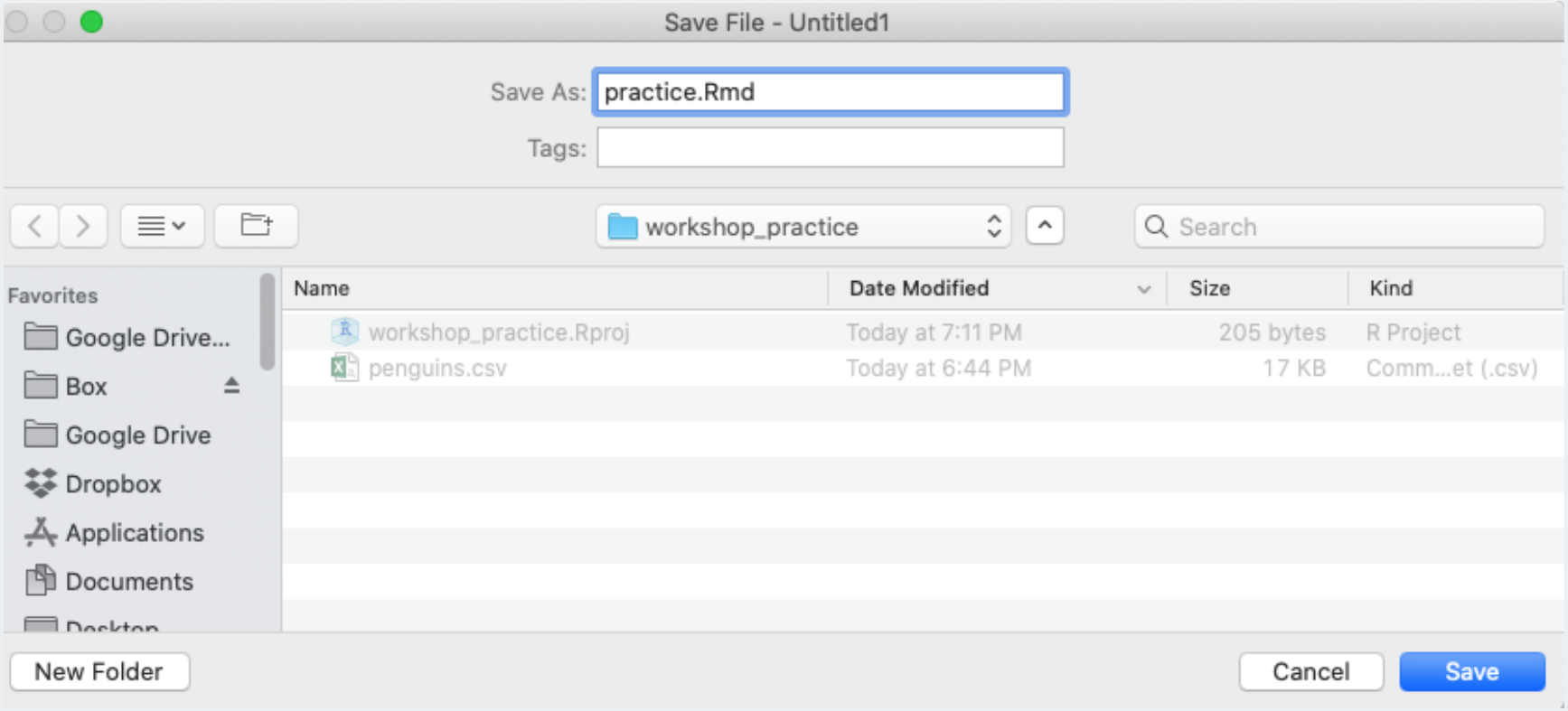


We will show you how to store and use this data in R as a data frame

Currently it is still just a file in your folder.

To run and save your code: Create a new Rmd!

- Then save it with a meaningful filename.
- You will be prompted to save it in your current working folder.



Load the packages we need in the Rmd

Add this code to the setup chunk in the Rmd and run that chunk:

```
library(tidyverse)
library(janitor)
```

````
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
library(tidyverse)
library(janitor)
```
````

Now we can use functions in these packages, such as `read_csv()` and `%>%` and `mutate()` and `tabyl()`

Remove everything in the Rmd below this code

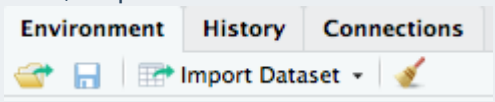
- Loading library code should always be at the top of your Rmd so you can use these packages in code "lower down"

Load the data set into R

- Create a new code chunk (Code -> insert chunk)
- Read in csv file from file path with code (filepath relative to Rproj directory)
- Copy this code to that code chunk and run it.

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

- Or, open saved file using Import Dataset button in Environment window:



+ From Text(readr).
  - If you use this option, **then copy and paste the importing code to your Rmd** so you have a record of from where and how you loaded the data set.

```
# Run in console:
View(penguins)
# Can also view the data by clicking on its name in the Environment tab
```

Your Rmd should look something like this:

Try knitting it!

practice.Rmd

Knit

```
1  ---
2  title: "Practice"
3  author: "Jessica Minnier"
4  date: "9/15/2020"
5  output: html_document
6  ---
7
8  ```{r setup, include=FALSE}
9  knitr::opts_chunk$set(echo = TRUE)
10 library(tidyverse)
11 library(janitor)
12 ```
13
14 ```{r}
15 penguins <- read_csv("penguins.csv")
16 ```
17
18
19
```

Load a data set: bonus lessons

- Importing Data, Rstudio support topic

Object types

Data frames (aka "tibbles" in tidyverse)

Vectors vs. **data frames**: a data frame is a collection (or array or table) of vectors

```
penguins
```

```
## # A tibble: 342 x 9
##       id species island bill_length_mm bill_
##    <dbl> <chr>   <chr>           <dbl>
##  1  1689 Adelie  Torge…           39.1
##  2  4274 Adelie  Torge…           NA
##  3  4539 Adelie  Torge…           40.3
##  4  2435 Adelie  Torge…           36.7
##  5  2326 Adelie  Torge…           39.3
##  6  2637 Adelie  Torge…           38.9
##  7  4443 Adelie  Torge…           NA
##  8  2102 Adelie  Torge…           34.1
##  9  2975 Adelie  Torge…           42
## 10  3966 Adelie  Torge…           37.8
## # … with 332 more rows, and 3 more variables
## #   year <dbl>
```

- Allows different columns to be of different data types (i.e. numeric vs. text)
- Both numeric and text can be stored within a column (stored together as *text*).
- Vectors and data frames are examples of ***objects*** in R.
    - There are other types of R objects to store data, such as matrices, lists.
    - These will be discussed in future R workshops.

Variable (column) types

type	description
double/numeric	**numbers that are decimals**
character	**text, "strings"**
integer	integer-valued numbers
factor	categorical variables stored with levels (groups)
logical	boolean (TRUE, FALSE)

- We will focus on double & character, as most data will be of this type when using `read_csv()` to read in your data sets
- If you see `int` = integer as a column type, you can treat it as a double for most intents and purposes.

Data structure

- What are the different **variable types** in this data set?
- What is `NA`?

```
glimpse(penguins)   # structure of data
```

```
## Rows: 342
## Columns: 9
## $ id                <dbl> 1689, 4274, 4539, 2435, 2326, 2637, 4443, 2102, 297…
## $ species           <chr> "Adelie", "Adelie", "Adelie", "Adelie", "Adelie", "…
## $ island            <chr> "Torgersen", "Torgersen", "Torgersen", "Torgersen",…
## $ bill_length_mm    <dbl> 39.1, NA, 40.3, 36.7, 39.3, 38.9, NA, 34.1, 42.0, 3…
## $ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, 19.3, 20.6, 17.8, 19.6, 18.1, 20.…
## $ flipper_length_mm <dbl> 181, 186, 195, 193, 190, 181, 195, 193, 190, 186, 1…
## $ body_mass_g       <dbl> 3750, 3800, 3250, 3450, 3650, 3625, 4675, 3475, 425…
## $ sex               <chr> "male", "female", "female", "female", "male", "fema…
## $ year              <dbl> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 200…
```

Data set summary

```
summary(penguins)
```

```
##        id         species             island          bill_length_mm
##  Min.   :1001   Length:342         Length:342         Min.   :32.10
##  1st Qu.:2031   Class :character   Class :character   1st Qu.:39.45
##  Median :2984   Mode  :character   Mode  :character   Median :44.70
##  Mean   :3031                                         Mean   :44.00
##  3rd Qu.:4073                                         3rd Qu.:48.52
##  Max.   :4969                                         Max.   :59.60
##                                                       NA's   :6
##  bill_depth_mm   flipper_length_mm  body_mass_g       sex
##  Min.   :13.10   Min.   :172.0     Min.   :2700   Length:342
##  1st Qu.:15.60   1st Qu.:190.0     1st Qu.:3550   Class :character
##  Median :17.30   Median :197.0     Median :4050   Mode  :character
##  Mean   :17.15   Mean   :200.9     Mean   :4202
##  3rd Qu.:18.70   3rd Qu.:213.0     3rd Qu.:4750
##  Max.   :21.50   Max.   :231.0     Max.   :6300
##
##       year
##  Min.   :2007
##  1st Qu.:2007
##  Median :2008
##  Mean   :2008
```

Show (print) whole data frame

Tibble truncates the output to ten rows, so you can't actually see it all.

```
penguins
```

```
## # A tibble: 342 x 9
##       id species island bill_length_mm bill_depth_mm flipper_length_…
##    <dbl> <chr>   <chr>            <dbl>         <dbl>              <dbl>
##  1  1689 Adelie  Torge…            39.1          18.7                181
##  2  4274 Adelie  Torge…            NA            17.4                186
##  3  4539 Adelie  Torge…            40.3          18                  195
##  4  2435 Adelie  Torge…            36.7          19.3                193
##  5  2326 Adelie  Torge…            39.3          20.6                190
##  6  2637 Adelie  Torge…            38.9          17.8                181
##  7  4443 Adelie  Torge…            NA            19.6                195
##  8  2102 Adelie  Torge…            34.1          18.1                193
##  9  2975 Adelie  Torge…            42            20.2                190
## 10  3966 Adelie  Torge…            37.8          17.1                186
## # … with 332 more rows, and 3 more variables: body_mass_g <dbl>, sex <chr>,
## #   year <dbl>
```

View whole data frame

We showed this already, very handy to see *all* data. Run in console since it's more interactive.

```
View(penguins)
```

Data set info

```
dim(penguins)
```

```
## [1] 342   9
```

```
nrow(penguins)
```

```
## [1] 342
```

```
ncol(penguins)
```

```
## [1] 9
```

```
names(penguins)
```

```
## [1] "id"                "species"
## [4] "bill_length_mm"    "bill_depth_mm"
## [7] "body_mass_g"       "sex"
```

View the beginning of a data set

```
head(penguins)
```

```
## # A tibble: 6 x 9
##      id species island bill_length_mm bill_depth_mm flipper_length_… body_mass_
##   <dbl> <chr>   <chr>            <dbl>         <dbl>             <dbl>       <dbl
## 1  1689 Adelie  Torge…            39.1          18.7               181         375
## 2  4274 Adelie  Torge…            NA            17.4               186         380
## 3  4539 Adelie  Torge…            40.3          18                 195         325
## 4  2435 Adelie  Torge…            36.7          19.3               193         345
## 5  2326 Adelie  Torge…            39.3          20.6               190         365
## 6  2637 Adelie  Torge…            38.9          17.8               181         362
## # … with 2 more variables: sex <chr>, year <dbl>
```

View the end of a data set

```
tail(penguins)
```

```
## # A tibble: 6 x 9
##      id species island bill_length_mm bill_depth_mm flipper_length_… body_mass_
##   <dbl> <chr>   <chr>            <dbl>         <dbl>             <dbl>       <dbl
## 1  1947 Chinst… Dream             45.7          17                 195         365
## 2  4452 Chinst… Dream             55.8          19.8               207         400
## 3  2420 Chinst… Dream             43.5          18.1               202         340
## 4  4861 Chinst… Dream             49.6          18.2               193         377
## 5  4865 Chinst… Dream             50.8          19                 210         410
## 6  4162 Chinst… Dream             50.2          18.7               198         377
## # … with 2 more variables: sex <chr>, year <dbl>
```

Specify how many rows to view at beginning or end of a data set

```
head(penguins, 3)
```

```
## # A tibble: 3 x 9
##      id species island bill_length_mm bill_depth_mm flipper_length_… body_mass_
##   <dbl> <chr>   <chr>           <dbl>         <dbl>             <dbl>       <dbl
## 1  1689 Adelie  Torge…           39.1          18.7               181         375
## 2  4274 Adelie  Torge…           NA            17.4               186         380
## 3  4539 Adelie  Torge…           40.3          18                 195         325
## # … with 2 more variables: sex <chr>, year <dbl>
```

```
tail(penguins, 1)
```

```
## # A tibble: 1 x 9
##      id species island bill_length_mm bill_depth_mm flipper_length_… body_mass_
##   <dbl> <chr>   <chr>           <dbl>         <dbl>             <dbl>       <dbl
## 1  4162 Chinst… Dream           50.2          18.7               198         377
## # … with 2 more variables: sex <chr>, year <dbl>
```

Data frame cells, rows, or columns (rarely used)

Specific cell: `DatSetName[row#, column#]`

```
# Second row, Third column
penguins[2, 3]
```

```
## # A tibble: 1 x 1
##   island
##   <chr>
## 1 Torgersen
```

Entire row: `DatSetName[row#, ]`

```
# Second row
penguins[2,]
```

```
## # A tibble: 1 x 9
##      id species island bill_length_mm bill_
##   <dbl> <chr>   <chr>           <dbl>
## 1  4274 Adelie  Torge…             NA
## # … with 2 more variables: sex <chr>, year
```

Entire col: `DatSetName[, column#]`

```
# Third column
penguins[, 3]
```

```
## # A tibble: 342 x 1
##    island
##    <chr>
##  1 Torgersen
##  2 Torgersen
##  3 Torgersen
##  4 Torgersen
##  5 Torgersen
##  6 Torgersen
##  7 Torgersen
##  8 Torgersen
##  9 Torgersen
## 10 Torgersen
## # … with 332 more rows
```

Working with the data

The $

Suppose we want to single out the column of BMI values.

- How did we previously learn to do this?

```
penguins[, 4]
```

```
## # A tibble: 342 x 1
##    bill_length_mm
##             <dbl>
##  1           39.1
##  2           NA
##  3           40.3
##  4           36.7
##  5           39.3
##  6           38.9
##  7           NA
##  8           34.1
##  9           42
## 10           37.8
## # … with 332 more rows
```

The problem with this method, is that we need to know the column number which can change as we make changes to the data set.

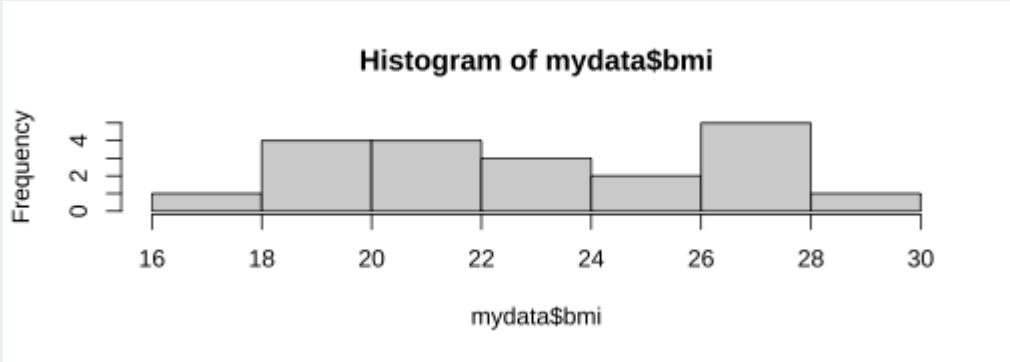
- Use the `$` instead: `DatSetName$VariableName`

```
penguins$bill_length_mm
```

```
##   [1] 39.1   NA 40.3 36.7 39.3 38.9
##  [16] 38.7 42.5 34.4 46.0 37.8 37.7
##  [31] 37.2 39.5 40.9 36.4 39.2 38.8
##  [46] 41.1 37.5 36.0 42.3 39.6 40.1
##  [61] 41.3 37.6 41.1 36.4 41.6 35.5
##  [76] 40.9 37.2 36.2 42.1 34.6 42.9
##  [91] 41.1 34.0 39.6 36.2 40.8 38.1
## [106] 38.6 38.2 38.1 43.2 38.1 45.6
```

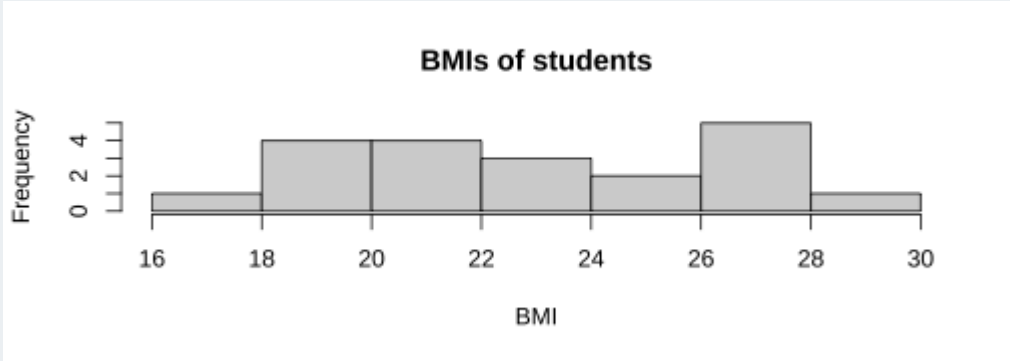
Basic plots of numeric data: Histogram

```
hist(mydata$bmi)
```



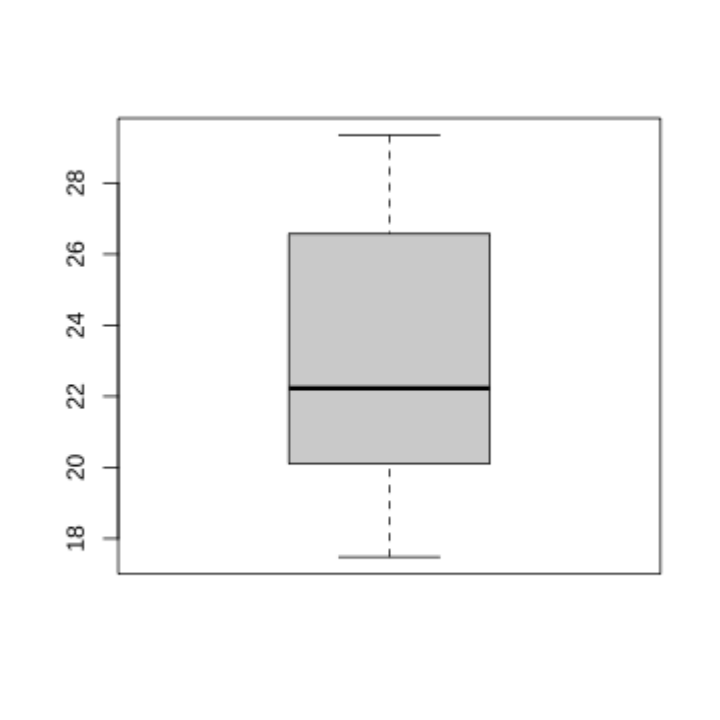
With extra features:

```
hist(mydata$bmi, xlab = "BMI", main="BMIs of students")
```

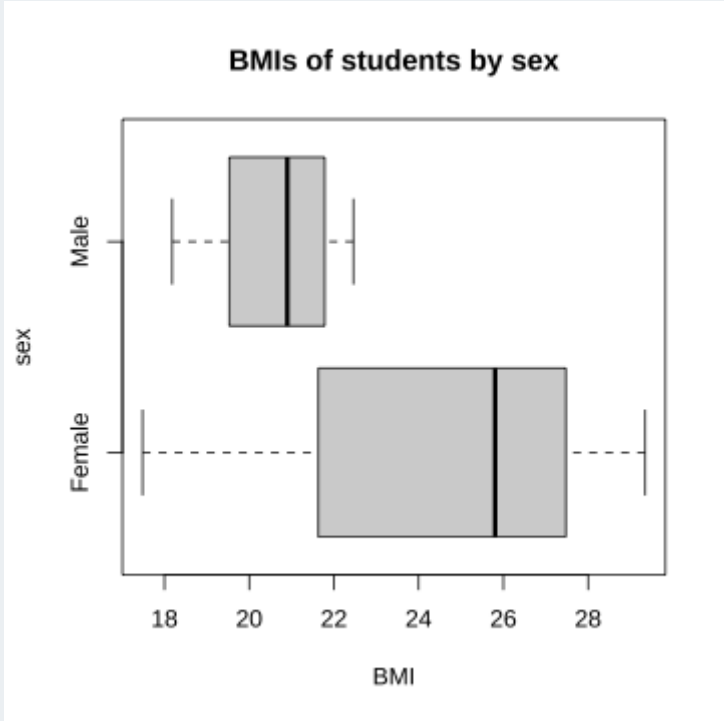


Basic plots of numeric data: Boxplot

```
boxplot(mydata$bmi)
```

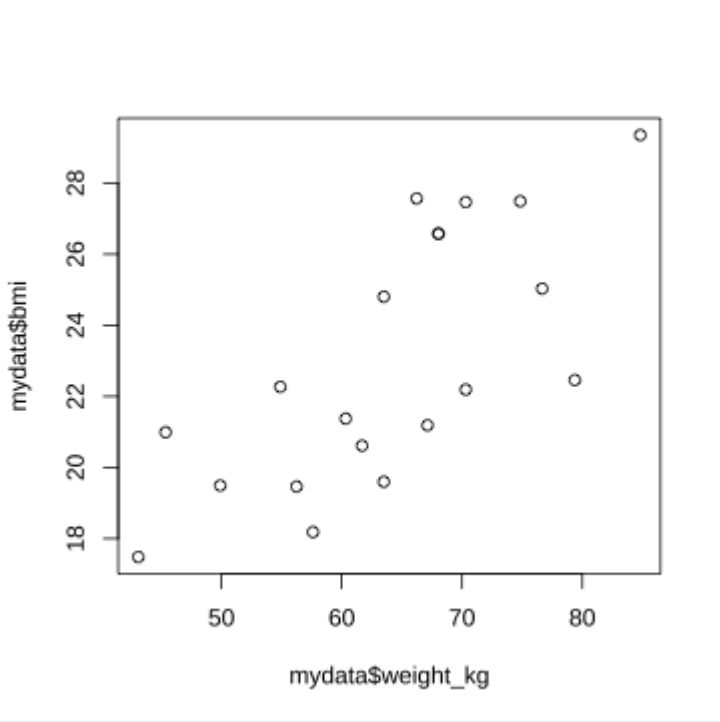



```
boxplot(mydata$bmi ~ mydata$sex,
  horizontal = TRUE,
  xlab = "BMI", ylab = "sex",
  main = "BMIs of students by sex")
```

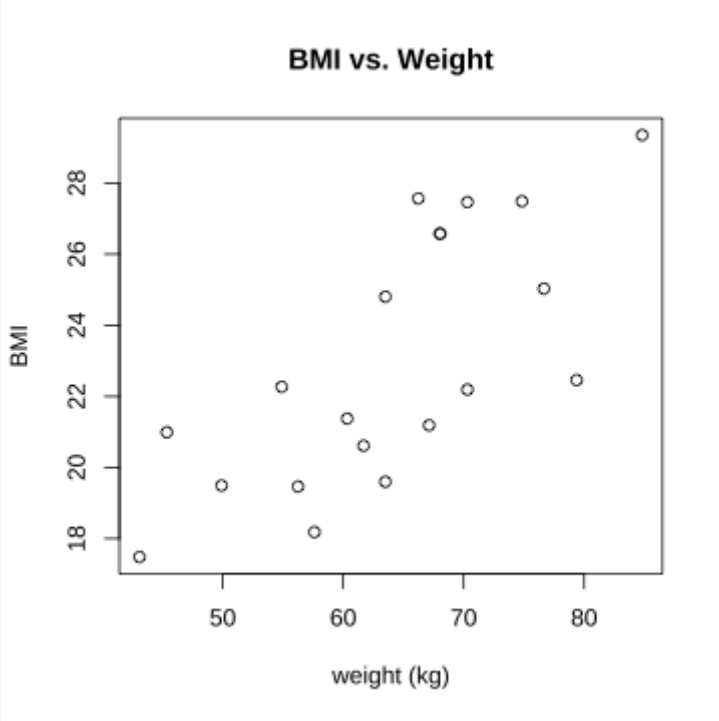


Basic plots of numeric data: Scatterplot

```
plot(mydata$weight_kg, mydata$bmi)
```




```
plot(mydata$weight_kg, mydata$bmi,
   xlab = "weight (kg)", ylab = "BMI",
   main = "BMI vs. Weight")
```



Summary stats of numeric data (1/2)

- Standard R `summary` command

```
summary(mydata$bmi)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   17.48   20.36   22.23   23.01   26.58   29.35
```

- Mean and standard deviation

```
mean(mydata$bmi)
```

```
## [1] 23.00838
```

```
sd(mydata$bmi)
```

```
## [1] 3.56471
```

Summary stats of numeric data (2/2)

- Min, max, & median

```
min(mydata$bmi)
```

```
## [1] 17.4814
```

```
max(mydata$bmi)
```

```
## [1] 29.3495
```

```
median(mydata$bmi)
```

```
## [1] 22.22985
```

- Quantiles

```
quantile(mydata$bmi, prob=c(0, .25, .5, .75, 1))
```

```
##       0%      25%      50%      75%     100%
## 17.48140 20.35878 22.22985 26.57810 29.34950
```

Practice 2

1. Create a new Rmd or continue in your current Rmd.

Working with data, we will use the pipe %>%

The pipe operator %>% is part of the tidyverse, and strings together commands to be performed sequentially

```
penguins %>% head(n=3)      # prounounce %>% as "then"
```

```
## # A tibble: 3 x 9
##      id species island bill_length_mm bill_depth_mm flipper_length_… body_mass_
##   <dbl> <chr>   <chr>           <dbl>         <dbl>             <dbl>       <dbl
## 1  1689 Adelie  Torge…           39.1          18.7               181         375
## 2  4274 Adelie  Torge…           NA            17.4               186         380
## 3  4539 Adelie  Torge…           40.3          18                 195         325
## # … with 2 more variables: sex <chr>, year <dbl>
```

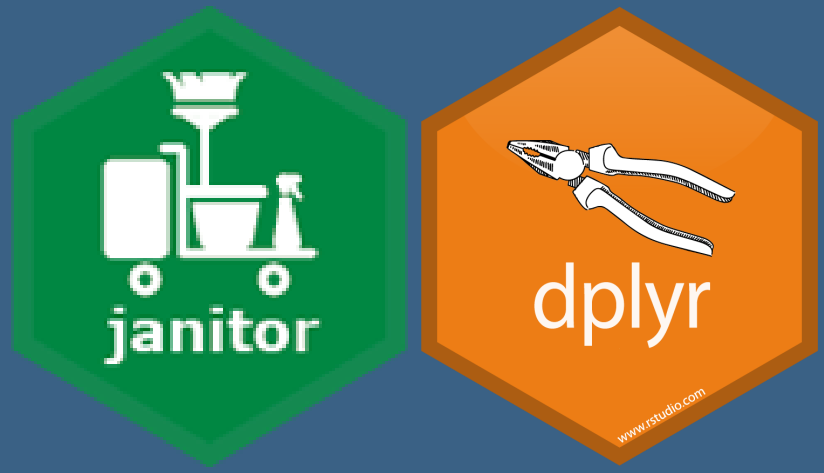
- Always *first list the tibble* that the commands are being applied to
- Can use **multiple pipes** to run multiple commands in sequence
  - What does the following code do?

```
penguins %>% head(n=2) %>% summary()
```

Quick tips on summarizing data

categorical data

numerical data



janitor, dplyr

Numerical data summaries: $ vs summarize()

We saw how to summarize a vector pulled with $, but there are easier ways to summarize multiple columns at once.

```
mean(penguins$body_mass_g)
```

```
## [1] 4201.754
```

```
median(penguins$body_mass_g)
```

```
## [1] 4050
```

```
penguins %>%
  summarize(mean(body_mass_g),
            median(body_mass_g))
```

```
## # A tibble: 1 x 2
##   `mean(body_mass_g)` `median(body_mass_g)`
##                 <dbl>                 <dbl>
## 1               4202.                  4050
```

`summarize()` with NA

- Don't forget `na.rm = TRUE` if you need it.
- You can also name these columns.

```
penguins %>%
  summarize(mean_mass = mean(body_mass_g),
            mean_len = mean(bill_length_mm, na.rm = TRUE))
```

```
## # A tibble: 1 x 2
##   mean_mass mean_len
##       <dbl>    <dbl>
## 1     4202.     44.0
```

By group `summarize()` (1/2)

- We can summarize data as a whole, or in groups with `group_by()`
- `group_by()` is very powerful, see data wrangling cheatsheet

```
# summary of all data as a whole
penguins %>%
  summarize(mass_mean =mean(body_mass_g),
            mass_sd = sd(body_mass_g),
            mass_cv = sd(body_mass_g)/mean(body_mass_g))
```

```
## # A tibble: 1 x 3
##   mass_mean mass_sd mass_cv
##       <dbl>   <dbl>   <dbl>
## 1     4202.    802.   0.191
```

By group `summarize()` (2/2)

- We can summarize data as a whole, or in groups with `group_by()`
- `group_by()` is very powerful, see data wrangling cheatsheet

```
# summary by group variable
penguins %>%
  group_by(species) %>%
  summarize(n_per_group = n(),
            mass_mean =mean(body_mass_g),
            mass_sd = sd(body_mass_g),
            mass_cv = sd(body_mass_g)/mean(body_mass_g))
```

```
## # A tibble: 3 x 5
##   species   n_per_group mass_mean mass_sd mass_cv
##   <chr>           <int>     <dbl>   <dbl>   <dbl>
## 1 Adelie            151     3701.    459.  0.124
## 2 Chinstrap          68     3733.    384.  0.103
## 3 Gentoo            123     5076.    504.  0.0993
```

Advanced `summarize(across())` (1/2)

- Can also use `across()` to summarize multiple variables (more examples)

```
penguins %>%
  summarize(across(c(body_mass_g, bill_depth_mm), mean))
```

```
## # A tibble: 1 x 2
##   body_mass_g bill_depth_mm
##         <dbl>         <dbl>
## 1       4202.          17.2
```

```
penguins %>%
  summarize(across(where(is.numeric), mean, na.rm=TRUE))
```

```
## # A tibble: 1 x 6
##      id bill_length_mm bill_depth_mm flipper_length_mm body_mass_g  year
##   <dbl>          <dbl>         <dbl>             <dbl>       <dbl> <dbl>
## 1 3031.           44.0          17.2              201.       4202. 2008.
```

Advanced `summarize(across())` (2/2)

- Can also use `across()` to summarize multiple variables (more examples)

```
penguins %>%
  summarize(across(where(is.character), n_distinct))
```

```
## # A tibble: 1 x 3
##   species island   sex
##     <int>  <int> <int>
## 1       3      3     3
```

Frequency tables: simple `count()`

```
penguins %>% count(island)
```

```
## # A tibble: 3 x 2
##   island         n
##   <chr>      <int>
## 1 Biscoe       167
## 2 Dream        124
## 3 Torgersen     51
```

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

```
## # A tibble: 5 x 3
##   species   island         n
##   <chr>     <chr>      <int>
## 1 Adelie    Biscoe        44
## 2 Adelie    Dream         56
## 3 Adelie    Torgersen     51
## 4 Chinstrap Dream         68
## 5 Gentoo    Biscoe       123
```

Fancier frequency tables: `janitor` package's `tabyl` function

```
# default table
penguins %>% tabyl(species)
```

```
##    species   n   percent
##     Adelie 151 0.4415205
##  Chinstrap  68 0.1988304
##     Gentoo 123 0.3596491
```

```
# output can be treated as tibble
penguins %>% tabyl(species) %>% select
```

```
##    species   percent
##     Adelie 0.4415205
##  Chinstrap 0.1988304
##     Gentoo 0.3596491
```

adorn_ your table!

```
penguins %>%
  tabyl(species) %>%
  adorn_totals("row") %>%
  adorn_pct_formatting(digits=2)
```

```
##    species   n percent
##     Adelie 151  44.15%
##  Chinstrap  68  19.88%
##     Gentoo 123  35.96%
##      Total 342 100.00%
```

2x2 tabyls

```
# default 2x2 table
penguins %>%
  tabyl(species, sex)
```

```
##    species female male NA_
##     Adelie     73   73   5
##  Chinstrap     34   34   0
##     Gentoo     58   61   4
```

What adornments does the tabyl to right have?

```
penguins %>% tabyl(species, sex) %>%
  adorn_percentages(denominator = "col") %>%
  adorn_totals("row") %>%
  adorn_pct_formatting(digits = 1) %>%
  adorn_ns()
```

```
##    species       female          male
##     Adelie  44.2%  (73)  43.5%  (73)  55.6%
##  Chinstrap  20.6%  (34)  20.2%  (34)   0.0%
##     Gentoo  35.2%  (58)  36.3%  (61)  44.4%
##      Total 100.0% (165) 100.0% (168) 100.0%
```

- Base R has a **`table`** function, but it is clunkier and the output is not a data frame.
- See the tabyl vignette for more information, adorn options, & 3-way **`tabyl`**s

3 way tabyls are possible

```
penguins %>% tabyl(species, island, sex)
```

```
## $female
##    species Biscoe Dream Torgersen
##     Adelie     22    27        24
##  Chinstrap      0    34         0
##     Gentoo     58     0         0
##
## $male
##    species Biscoe Dream Torgersen
##     Adelie     22    28        23
##  Chinstrap      0    34         0
##     Gentoo     61     0         0
##
## $NA_
##    species Biscoe Dream Torgersen
##     Adelie      0     1         4
##  Chinstrap      0     0         0
##     Gentoo      4     0         0
```

Practice 3

1. Continue adding code chunks to your Rmd (or, start a new one! But remember to load the libraries and data at the top.)

2. How many different years are in the data? (Hint: use `tabyl()` or `n_distinct()`)

3. Count the number of penguins measured each year.

4. Calculate the median body mass by each species and sex subgroup. Use `summarize()` and `group_by()` to do this.

5. Create a 2x2 table of number of penguins measured in each year by each island.