A Field Guide to Base R

Josh Allen

Department of Political Science at Georgia State University

1/27/23

A Review of the Basics of R

Setting Your Working Directory

- Your working directory is where all your files live
- You may know where your files are…
- But R does not
- If you want to use any data that does not come with a package you are going to need to tell R where it lives

Cats and Boxes

- You can put a box inside a box
- You can put a cat inside a box
- You can put a cat inside a box inside of a box
- You cannot put a box inside a cat
- You cannot put cat in a cat

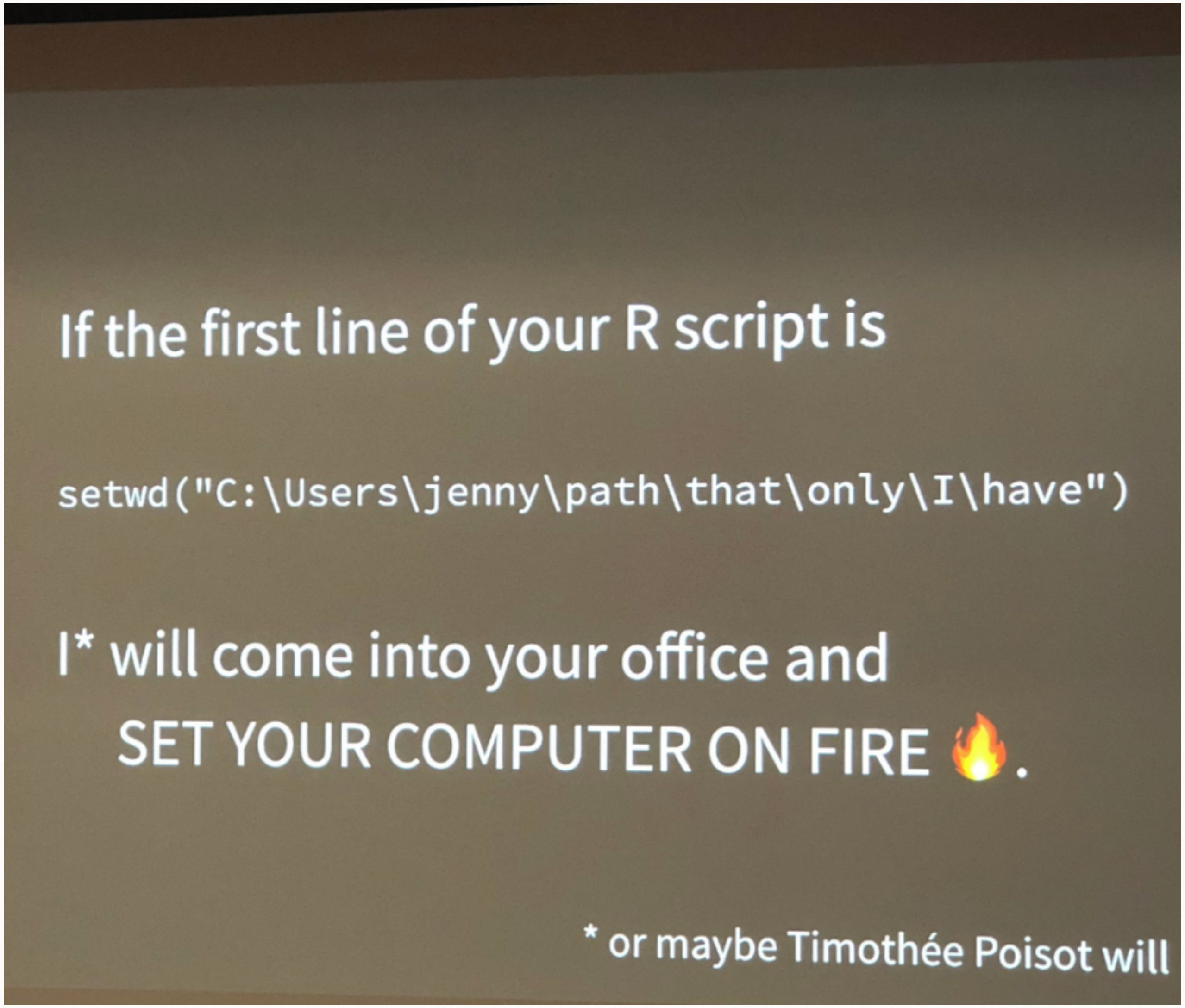
Working Directories

```r
getwd()
```

```
[1] "/Users/josh/Dropbox/Research-Data-Services-Workshops/8810-guest-lecture"
```

```r
setwd("path/to/your/project") #mac/linux
setwd("path\to\your\project") # windows
```

How To Make Your Life Easier



If the first line of your R script is

```
setwd("C:\Users\jenny\path\that\only\I\have")
```

I* will come into your office and
SET YOUR COMPUTER ON FIRE 🔥.

* or maybe Timothée Poisot will

source: Jenny Bryan

How To Make Your Life Easier

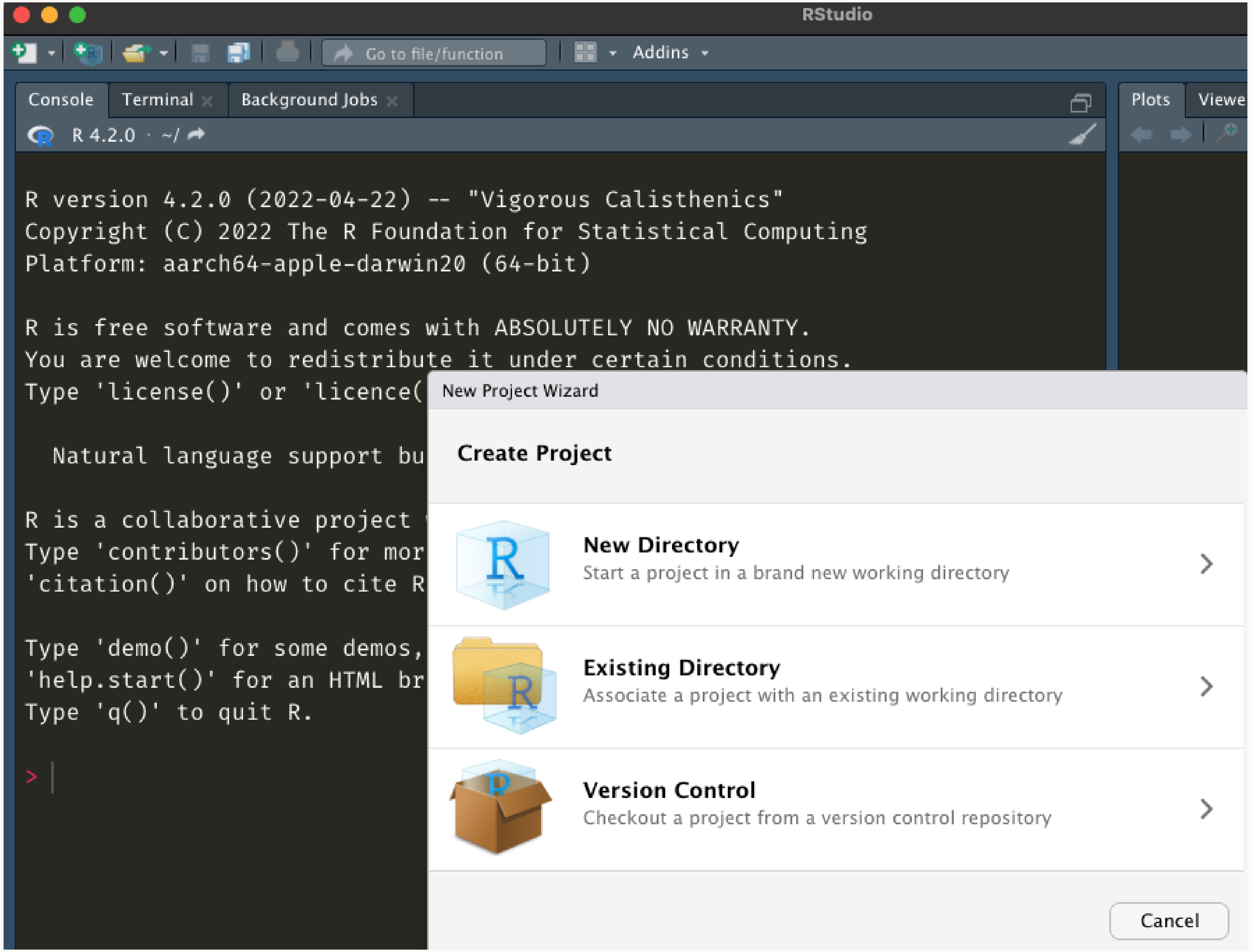
Working Directory for My Laptop

```
"/Users/josh/Dropbox/Research-Data-Services-Workshops/research-data-services-r-workshops/slides"
```

Working Directory of My Office Computer

```
"/Volumes/6TB Raid 10/Dropbox/Research-Data-Services-Workshops/research-data-services-r-workshops/slides"
```

R Projects



```
R version 4.2.0 (2022-04-22) -- "Vigorous Calisthenics"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence(

  Natural language support bu

R is a collaborative project
Type 'contributors()' for mor
'citation()' on how to cite R

Type 'demo()' for some demos,
'help.start()' for an HTML br
Type 'q()' to quit R.

> 
```

New Project Wizard

Create Project

New Directory
Start a project in a brand new working directory

Existing Directory
Associate a project with an existing working directory

Version Control
Checkout a project from a version control repository

The Mantra

- Everything in R is an Object
- Everything has a name
- You do stuff with functions
- Packages(i.e. libraries) are homes to pre-written functions.
  - You can also write your own functions and in some cases should.

An Example

```r
digi <- c("1","2","3","4")
mean(digi)
```

```
[1] NA
```

```r
numbs <- c(1:4)
mean(numbs)
```

```
[1] 2.5
```

```r
class(digi)
```

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

```r
lets <- letters
class(lets)
```

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

R Some Basics

Basic Maths

- R is equipped with lots of mathematical operations

```r
2+2 ## addition
```

```
[1] 4
```

```r
4-2 ## subtaction
```

```
[1] 2
```

```r
600*100 ##multiplication
```

```
[1] 60000
```

```r
100/10 ##division
```

```
[1] 10
```

```r
10*10/(3^4*2)-2 ## Pemdas
```

```
[1] -1.382716
```

```r
log(100)
```

```
[1] 4.60517
```

```r
sqrt(100)
```

Basic Maths

R is also equipped with modulo operations (integer division and remainders), matrix algebra, etc

```r
100 %/% 60 # How many whole hours in 100 minutes?
```

```
[1] 1
```

```r
100 %% 60 # How many minutes are left over?
```

```
[1] 40
```

```r
m <- matrix(1:8, nrow=2)
n <- matrix(8:15, nrow=4) # this is just me creating matrices
mat <- matrix(1:15, ncol = 5)
m %*% n # Matrix multiplication
```

```
     [,1] [,2]
[1,]  162  226
[2,]  200  280
```

```r
t(mat) # transpose a matrix
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

```
[3,]    7    8    9
[4,]   10   11   12
[5,]   13   14   15
```

Logical Statements & Booleans

Test	Meaning	Test	Meaning
`x < y`	Less than	`x %in% y`	In set
`x > y`	Greater than	`is.na(x)`	Is missing
`==`	Equal to	`!is.na(x)`	Is not missing
`x <= y`	Less than or equal to		
`x >= y`	Greater than or equal to		
`x != y`	Not equal to		
`x | y`	Or		
`x & y`	And		

Booleans and Logicals in Action

```
1>2
```

```
[1] FALSE
```

```
1<2
```

```
[1] TRUE
```

```
1 == 2
```

```
[1] FALSE
```

```
1 < 2 | 3 > 4 ## only one test needs to true to return true
```

```
[1] TRUE
```

```
1 < 2 & 3>4 ## both tests must be true to return true
```

```
[1] FALSE
```

Logicals, Booleans, and Precedence

- R like most other programming languages will evaluate our logical operators(==, >, etc) before our booleans(|, &, etc).

```
1 > 0.5 & 2
```

```
[1] TRUE
```

- What's happening here is that R is evaluating two separate "logical" statements:
- 1 > 0.5, which is is obviously TRUE.
- 2, which is TRUE(!) because R is "helpfully" converting it to as.logical(2).
- It is way safer to make explicit what you are doing.
- If your code is doing something weird it might just be because of precedence issues
  - See R Cookbook 2.11

```
1 > 0.5 & 1 > 2
```

```
[1] FALSE
```

Other Useful Tricks

Value matching using `%in%`

To see whether an object is contained within (i.e. matches one of) a list of items, use `%in%`.

```r
4 %in% 1:10
```

```
[1] TRUE
```

```r
4 %in% 5:10
```

```
[1] FALSE
```

Cool Now What?

- While this is boring it opens up lots
- We may need to set up a group of tests to do something to data.
- We may need all this math stuff to create new variables
- However we need to ***Assign them*** to reuse them later in functions.
  - Including datasets

Everything is an Object

What are Objects?

- Objects are what we work with in `R`

```
 [1] "is.array"                 "is.atomic"
 [3] "is.call"                  "is.character"
 [5] "is.complex"               "is.data.frame"
 [7] "is.double"                "is.element"
 [9] "is.environment"           "is.expression"
[11] "is.factor"                "is.finite"
[13] "is.function"              "is.infinite"
[15] "is.integer"               "is.language"
[17] "is.list"                  "is.loaded"
[19] "is.logical"               "is.matrix"
[21] "is.na"                    "is.na.data.frame"
[23] "is.na.numeric_version"    "is.na.POSIXlt"
[25] "is.na<-"                  "is.na<-.default"
[27] "is.na<-.factor"           "is.na<-.numeric version"
```

Vectors

- Come in **two flavors**
- **Atomic**: all the stuff must be the same type
- **Lists**: stuff can be different types

```r
my_vec <- c(1:10)
is.vector(my_vec)
```

```
[1] TRUE
```

```r
my_list <- list(a = c(1:4), b = "Hello World", c = data.frame(x = 1:10, y = 1:10))
is.vector(my_list)
```

```
[1] TRUE
```

Atomic Vectors

- Come in a variety of flavors
- Numeric: Can contain whole numbers or decimals
- Logicals: Can only take two values TRUE or FALSE
- Factors: Can only contain predefined values. Used to store categorical data
    - Ordered factors are special kind of factor where the order of the level matters.
- Characters: Holds character strings
    - Base R will often convert characters to factors. That is bad because it will choose the levels for you

Lists

- Lists are everywhere in R

```r
data_frame <- data.frame(a = rnorm(3),
                         b = rnorm(3))
typeof(data_frame)
```

```
[1] "list"
```

```r
dats_wrong <- data.frame(a = 1:3,
                         b = 1:4)
```

```
Error in data.frame(a = 1:3, b = 1:4): arguments imply differing
number of rows: 3, 4
```

```r
example_mod <- lm(body_mass_g ~ bill_depth_mm, data = penguins)
typeof(example_mod)
```

```
[1] "list"
```

```r
length(example_mod$residuals);length(example_mod$coefficients)
```

```
[1] 342
[1] 2
```

A Quick Aside on Naming Stuff

- Things we can never name stuff

```
if
else
while
function
for
TRUE
FALSE
NULL
Inf
NaN
NA
```

There are more see this website for a more complete list

A Quick Aside on Naming Stuff(cont)

Semi-reserved words

For simple things like assigning `c <- 4` and then doing `d <- c(1,2,3,4)` R will be able to distinguish between assign c the value of 4 and the `c` that calls `concatenate` which is way more important in R.

However it is generally a good idea, *unless you know what you are doing*, to avoid naming things that are functions in `R` because `R` will get confused.

```r
my_cool_fun <- function(x){
 x <- x*5
return(x)
}

datas <- c(1:10)

my_cool_fun(datas)
```

```
[1]  5 10 15 20 25 30 35 40 45 50
```

```r
my_cool_fun[1]
```

```
Error in my_cool_fun[1]: object of type 'closure' is not subsettable
```

How and What to Name Objects

The best practice is to use concise descriptive names

When loading in data typically I do `raw_my_dataset_name` and after data all of my cleaning I do `clean_my_dataset_name`

- Objects must start with a letter. But can contain letters, numbers, `_`, or `.`
  - snake_case_like_this_is_what_I_use
  - somePeopleUseCamelCase
  - some_People.are_Do_not.like_Convention

Example and Discussion provided in R for Data Sciency by Hadley Wickham

Your Turn

- Create a vector from 1:100
- Create a character vector named hp with only the value of harry potter
- Find the length of each vector
- create a vector named pak to install “marginaleffects”, “modelsummary”

04:00

Navigating Objects in R

Our Data

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	b
Adelie	Torgersen	39.1	18.7	181	
Adelie	Torgersen	39.5	17.4	186	
Adelie	Torgersen	40.3	18.0	195	
Adelie	Torgersen	NA	NA	NA	
Adelie	Torgersen	36.7	19.3	193	
Adelie	Torgersen	39.3	20.6	190	

Indexing [ ]

- We can use column position to index objects.
- There are two slots we can use *rows* and *columns* in the brackets if we are using a dataframe like this.
- `object_name[row number, column number]`
- We can also subset our data by column position using `:` or `c(column 1, column 2)`

```r
penguins[1,1]
```

species
Adelie

```r
penguins[1,1:2]
penguins[1,c(1,4)]
```

species	island
Adelie	Torgersen

species	bill_depth_mm
Adelie	18.7

Indexing [ ] (cont)

- We can tell R what element of a list using a combo of `[]` and `[[]]`

```r
my_list <- list(a = 1:4, b = "Hello World", c = data.frame(x = 1:3, y = 4:6))
```

```r
my_list[[1]][2] ## get the first item in the list and the second element of that item
```

```
[1] 2
```

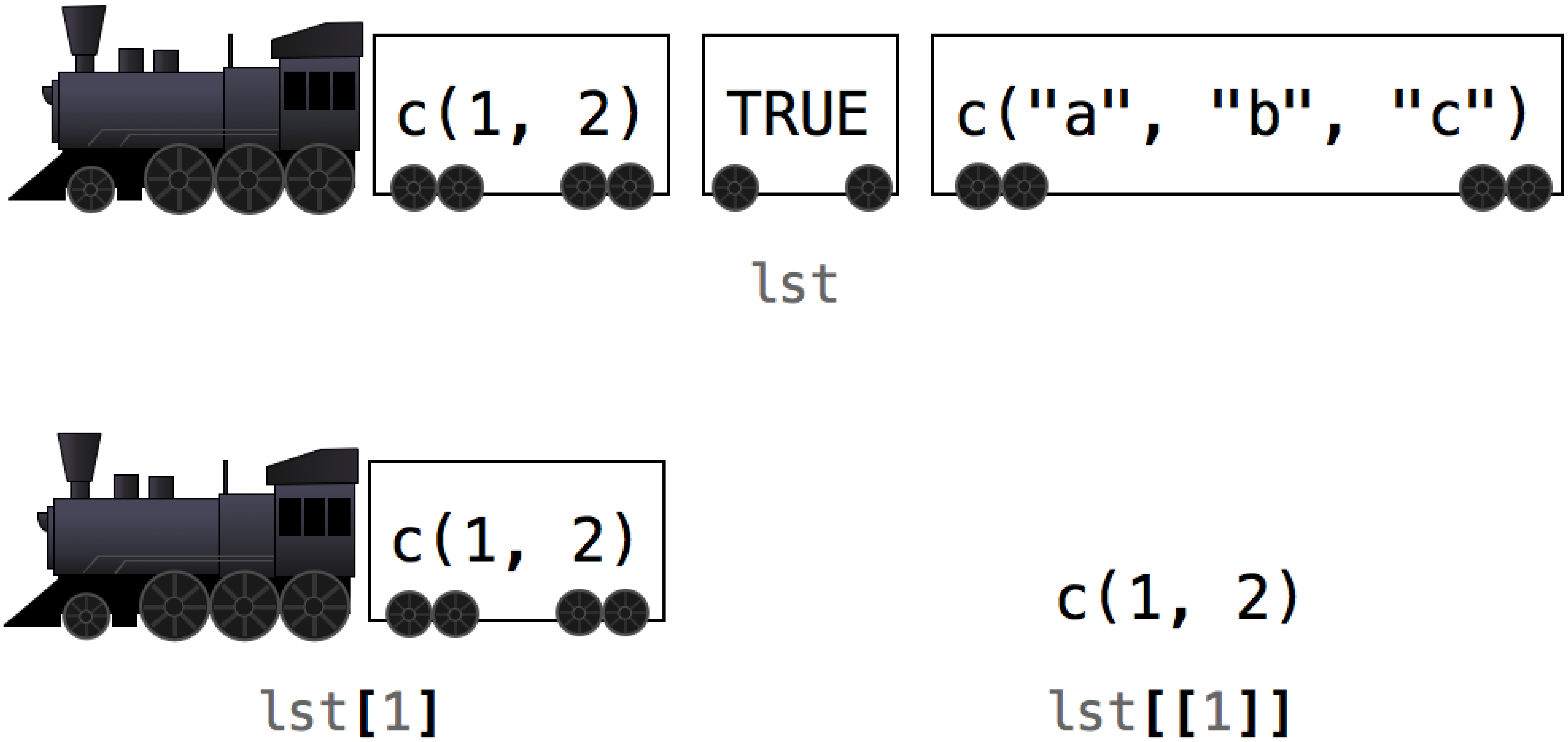
```r
my_list[2]
```

```
$b
[1] "Hello World"
```

```r
my_list[[3]][[1]]
```

```
[1] 1 2 3
```

[ ] vs [[ ]]



c(1, 2)

TRUE

c("a", "b", "c")

lst

c(1, 2)

lst[1]

c(1, 2)

lst[[1]]

Negative Indexing

- We can also exclude various elements using - and/or tests that I showed you earlier

```
penguins[,-1]
```

island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_
Torgersen	39.1	18.7	181	375
Torgersen	39.5	17.4	186	380
Torgersen	40.3	18.0	195	325
Torgersen	NA	NA	NA	N
Torgersen	36.7	19.3	193	345
Torgersen	39.3	20.6	190	365

Negative Indexing(cont)

- We can use - or : as well to subset stuff

```r
penguins[,-(1:4)]
```

flipper_length_mm	body_mass_g	se
181	3750	ma
186	3800	fe
195	3250	fe
NA	NA	NA
193	3450	fe
190	3650	ma

```r
penguins[,-c(2,3,5,8)]
```

species	bill_depth_mm	body_mass_
Adelie	18.7	375
Adelie	17.4	380
Adelie	18.0	325
Adelie	NA	N
Adelie	19.3	345
Adelie	20.6	365

Subsetting By Tests

```r
penguins[penguins["sex"] == "female", c("species", "sex")]
```

species	sex
Adelie	female
Adelie	female
NA	NA
Adelie	female
Adelie	female
NA	NA
NA	NA
NA	NA
NA	NA
Adelie	female

$ Indexing

A really useful way of indexing in `R` is referencing stuff by name rather than position. - The way we do this is throught the `$`

```r
my_list$a
```

```
[1] 1 2 3 4
```

```r
my_list$b
```

```
[1] "Hello World"
```

```r
my_list$c
```

```
  x y
1 1 4
2 2 5
3 3 6
```

Indexing(cont)

```
my_list[[3]][[2]] ## these are just returning the same thing
```

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

```
my_list$c$y
```

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

$ in action

This will just subset things

```r
penguins[penguins$species == "Gentoo", c("species", "island", "bill_length_mm")]
```

species	island	bill_length_mm
Gentoo	Biscoe	46.1
Gentoo	Biscoe	50.0
Gentoo	Biscoe	48.7
Gentoo	Biscoe	50.0
Gentoo	Biscoe	47.6
Gentoo	Biscoe	46.5
Gentoo	Biscoe	45.4
Gentoo	Biscoe	46.7
Gentoo	Biscoe	43.3
Gentoo	Biscoe	46.8

Comparing what we know how to do

Select Filter Mutate

Tidyverse

```r
penguins |>
select(species, island, sex)
```

species	island	sex
Adelie	Torgersen	male
Adelie	Torgersen	female
Adelie	Torgersen	female
Adelie	Torgersen	NA
Adelie	Torgersen	female

Base R

```r
penguins[, c("species", "island", "sex")]
```

species	island	sex
Adelie	Torgersen	male
Adelie	Torgersen	female
Adelie	Torgersen	female
Adelie	Torgersen	NA
Adelie	Torgersen	female

Sometimes it is just quicker

```r
penguins_base$range_body_mass <- max(penguins_base

penguins_base$bill_ratio  <- penguins_base$bill_le

mean(penguins_base$body_mass_g, na.rm = TRUE)
```

```
[1] 4201.754
```

```r
penguins <- penguins |>
mutate(range_body_mass = max(body_mass_g, na.rm =
       bill_ration = bill_length_mm/bill_depth_mm)

summarise(penguins, mean(body_mass_g, na.rm = TRUE
```

```
# A tibble: 1 × 1
  `mean(body_mass_g, na.rm =
TRUE)`

<dbl>
1
4202.
```

Sometimes the Original is Just as Good as the Wrapper

```r
data("starwars")

filter(starwars, str_detect(eye_color, "blu"))
```

name	eye_color
Luke Skywalker	blue
Owen Lars	blue
Beru Whitesun lars	blue
Obi-Wan Kenobi	blue-gray
Anakin Skywalker	blue
Wilhuff Tarkin	blue
Chewbacca	blue
Jek Tono Porkins	blue

```r
starwars[grepl("blu",starwars$eye_color),]
```

name	eye_color
Luke Skywalker	blue
Owen Lars	blue
Beru Whitesun lars	blue
Obi-Wan Kenobi	blue-gray
Anakin Skywalker	blue
Wilhuff Tarkin	blue
Chewbacca	blue
Jek Tono Porkins	blue

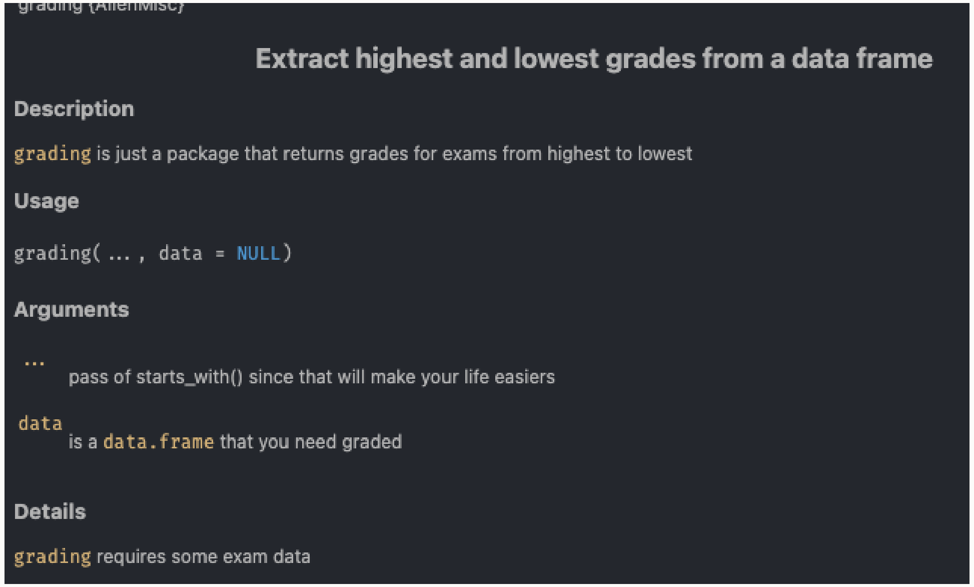
Finding Help

- Asking for help in R is easy the most common ways are `help(thingineedhelpwith)` and `?thingineedhelpwith`

```r
?grepl
```

- `?thingineedhelpwith` is probably the most common because it requires less typing.
- Base and Tidy functions differ in many ways other than naming conventions

Finding Help



grading {AllenMisc}

Extract highest and lowest grades from a data frame

Description

`grading` is just a package that returns grades for exams from highest to lowest

Usage

```
grading( ... , data = NULL)
```

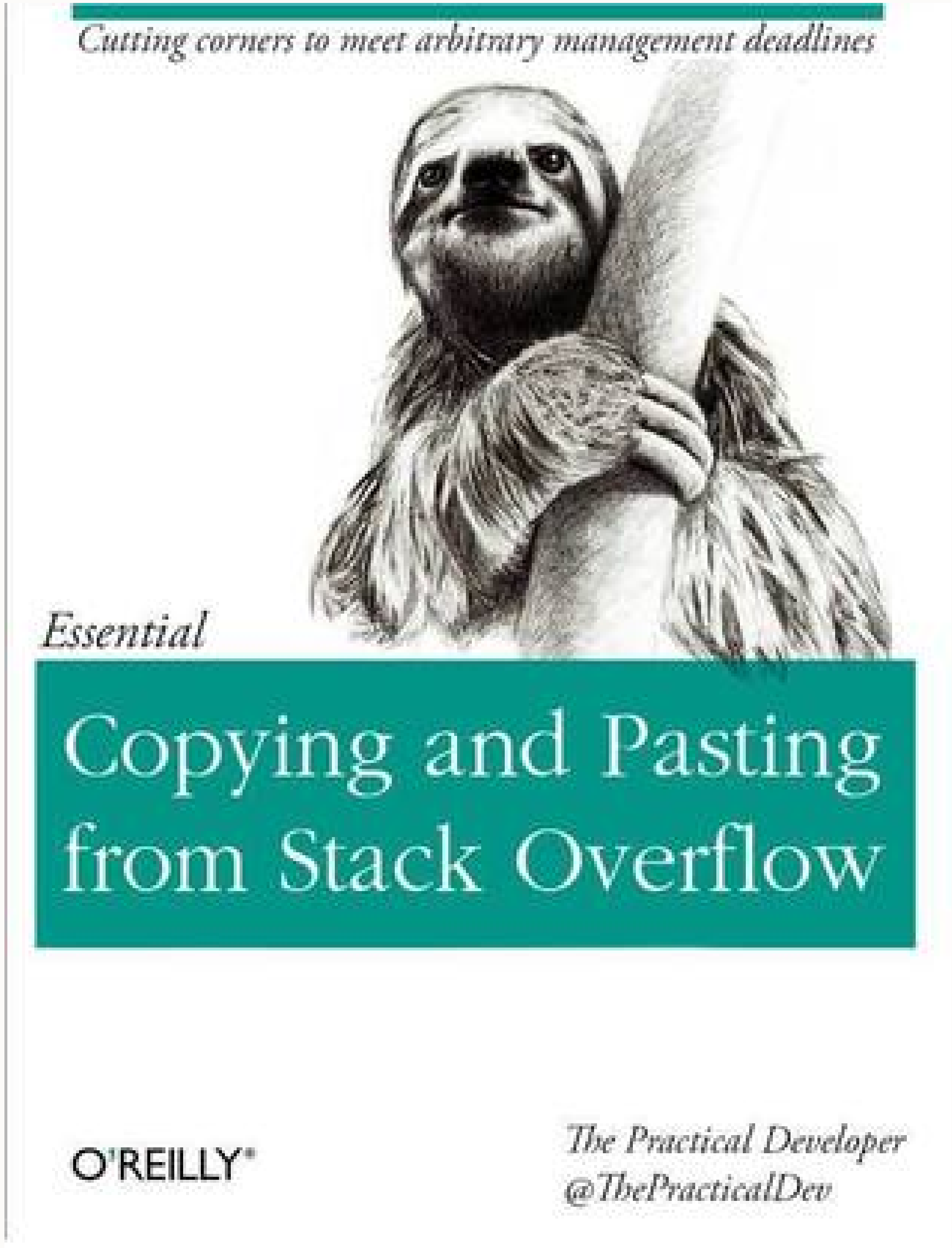
Arguments

`...` pass of starts_with() since that will make your life easiers

`data` is a `data.frame` that you need graded

Details

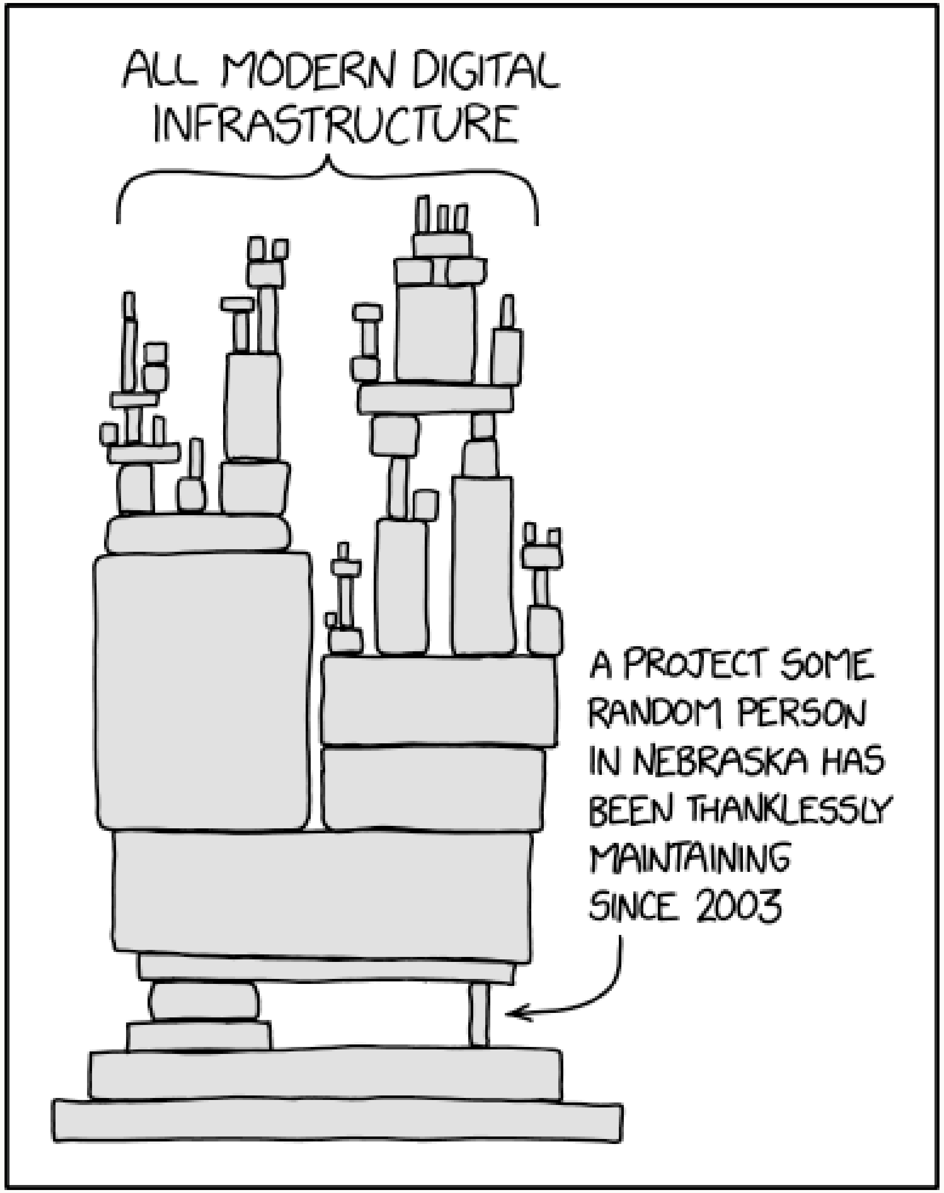
`grading` requires some exam data


Cutting corners to meet arbitrary management deadlines
Essential
Copying and Pasting from Stack Overflow
O'REILLY®
The Practical Developer
@ThePracticalDev

Your Turn

- Find the minimum value of `bill_length_mm`
- Find the maximum value of `body_mass_g`
- Subset the penguins data any way you want using column position or `$`
- Assign each of them to an object
- Create a vector from 1:10 index that vector using `[]` to return 2 and 4

The Tidyverse issue


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

Learning to Live With Each other

```r
penguins$big_peng <-  dplyr::case_when(penguins$body_mass_g > mean(penguins$body_mass_g, na.rm = TRUE) ~ "Big
      penguins$body_mass_g < mean(penguins$body_mass_g, na.rm = TRUE) ~ "Smol Penguin",
      penguins$body_mass_g == mean(penguins$body_mass_g, na.rm = TRUE) ~ "Average Penguin")

penguins$body_mass_g[is.na(penguins$body_mass_g)] <- 0

penguins |>
select(body_mass_g, big_peng)
```

```
# A tibble: 344 × 2
   body_mass_g big_peng
         <dbl> <chr>
 1        3750 Smol Penguin
 2        3800 Smol Penguin
 3        3250 Smol Penguin
 4           0 <NA>
 5        3450 Smol Penguin
 6        3650 Smol Penguin
 7        3625 Smol Penguin
 8        4675 Big Penguin
 9        3475 Smol Penguin
```

Learning to Live with Each Other

- Lots of stuff is repetitive
- Repetition isn’t necessarily bad but it can easily lead to mistakes

```r
penguins |>
drop_na() |>
mutate(body_mass_g = body_mass_g - min(body_mass_g, na.rm = TRUE) /
    (max(body_mass_g, na.rm = TRUE) - min(body_mass_g, na.rm = TRUE)),
    flipper_length_mm = flipper_length_mm - min(flipper_length_mm, na.rm = TRUE) /
    (max(flipper_length_mm, na.rm = TRUE) - min(flipper_length_mm, na.rm = TRUE)),
    bill_length_mm = bill_length_mm - min(bill_length_mm, na.rm = TRUE) /
    (max(bill_length_mm, na.rm = TRUE) - min(flipper_length_mm, na.rm = TRUE)))
```

One way this helps us

```r
# we can rewrite this code pretty easily and iterate over the entire dataset
rescale <- function(x){
    rng <- range(x, na.rm = TRUE, finite = TRUE)

   x-rng[1]/(rng[2] - rng[1])

}

penguins |>
mutate(across(where(is.numeric), \(x) rescale(x)))
```

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	b
Adelie	Torgersen	37.93273	17.14048	178.0847	
Adelie	Torgersen	38.33273	15.84048	183.0847	
Adelie	Torgersen	39.13273	16.44048	192.0847	

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	b
Adelie	Torgersen	NA	NA	NA	
Adelie	Torgersen	35.53273	17.74048	190.0847	

Reading in Data Gets easier

```r
rm(list = ls())
penguins <- palmerpenguins::penguins
starwars <- dplyr::starwars
data("mpg")
data("mtcars")

data_names = c("mpg", "penguins", "starwars", "mtcars")

for(i in 1:length(data_names)) {
  readr::write_csv(get(data_names[i]),
             paste0("data/",
                    data_names[i],
                    ".csv"))
}
my_files <- list.files(path = "data/",pattern = "*.csv", full.names = TRUE)

# Further arguments to read.csv can be passed in ...
```

Which Gives Us

```
[1] "all_csv"    "data_names" "i"          "mpg"        "mtcars"    
[6] "my_files"   "penguins"   "starwars"  
```