00_qa

Ari Anisfeld

9/3/2020

Key points: class 0

- Rstudio has a console to access R and a text editor to write code for reproducible projects.
  - Analogy: R is to RStudio as Tony Stark is to Ironman's suit
- R extensible through packages
  - use `install.packages("")` once and then `library()` each session
- Use <- to assign *any* object to a name
- functions take inputs and return outputs
  - input "understood" based on position or name
  - find out more about functions with `?` (e.g. `?filter`)

Up next: Watch videos for lab 1: reading data into R and manipulating it with `dplyr` verbs.

Questions from TAs / Piazza

- How do you assign a name to data? (<-)
- What is the difference between console and text editor section of R markdown?
- What is %>%?
- How do I read errors?
- What's the deal with backticks and quotes?
- Please add additional questions to the chat.

Reading files and `dplyr`

Key points: class 1: Reading files

- Tabular data is stored in a lot of different formats
  - e.g. `.csv`, `.xlsx`, `.dta`
- Read tabular data of a given type with the proper function
  - e.g. for `csvs` we have `read_csv()`
  - if you get a new type, google "How to read xxx files into R tidyverse"
- We need to be aware of the file path and can `setwd()`
- We know there are useful tools built into the `read_xxx()` functions
  - though we just scratched the surface

Questions from piazza / ta sessions.

- What do the errors mean? Can you write code to get the same errors on your machine?

```
setwd("~/Docutents")
Error in setwd("~/Docutents") :
  cannot change working directory
```

```
setwd(~/Documents/)
Error: unexpected '/' in "setwd(~/"
```

```
 # Finally I got it to work
setwd("~/Documents")
```

Questions from piazza

- @35 a funky data issue in excel files on some Windows machines
- Why does `read_dta("some_name.RData")` fail?

Key points: class 1: Manipulating data with `dplyr()`

- Choose columns with `select()`
- Choose rows based on a match criteria with `filter()`
  - We were introduced to comparison operators like == and `%in%` (more in class 3)
- Make new columns with `mutate()`
  - We were introduced to math functions (more in class 2)
- Sort data with `arrange()` and `arrange(desc())`
- Create summary statistics with `summarize()`

Up next: Watch videos for lab 2: vectors and data types. Try the warm-up

Question from Piazza

- @38 Debugging with your eyes.
- @41 `tidyverse` vs `baseR`

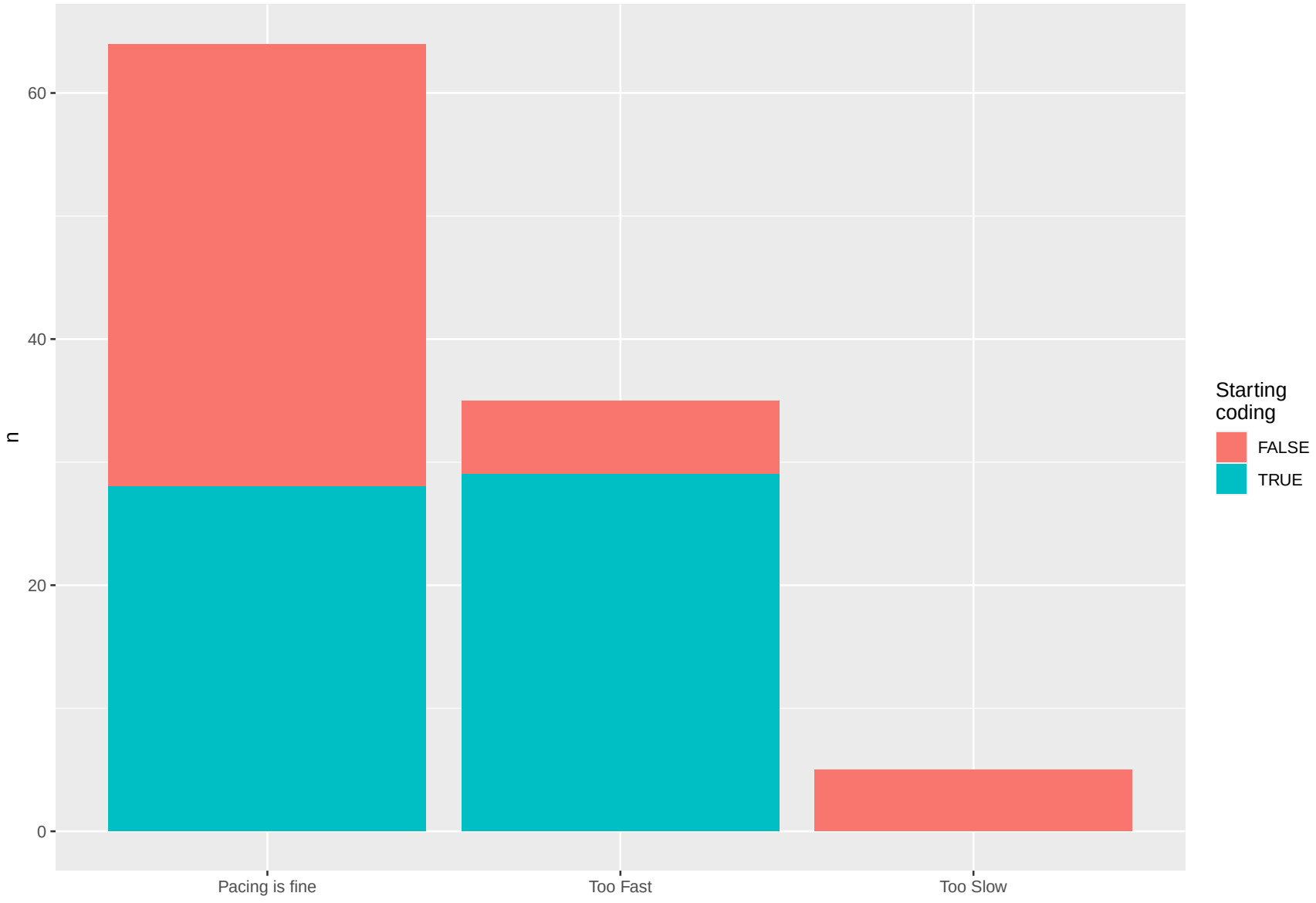
Practice

How many children live in Ohio?

- Restrict data to Ohio only.
- We have `poptotal` and `popadults`. Create a new column called `pop_children`.
- Aggregate data from Ohio counties to Ohio state total.

```
# midwest is a data set that comes with the tidyverse
library(tidyverse)
midwest
```

Feedback:


60
40
20
0
n
Pacing is fine
Too Fast
Too Slow
Starting coding
FALSE
TRUE

Response:

- Adding 30 minutes before lab for more direct instruction.
  - This is geared towards students who feel we're moving too fast.
- Now that we know you better, TAs will group more deliberately.
- We will provide more "basics" practice.
  - "Optional" worksheets that should be doable after watching the videos
- We will also provide code from the pdf, so when you need to copy and paste you can use an R file.

Course logistics:

- When should we start working on the final project?
  - Start looking for a dataset now.
  - Write code to read it into R and start investigating with `dplyr` verbs.
    - Ask simple questions that can be addressed with your current tools.

Vectors

What's the deal with fall coding lab?

Two tracks:

Accelerated: 2 lessons

- 2 lessons covering loops and functions.

Not accelerated: 5 lessons.

- 3 lessons review summer camp material
- 2 lessons covering loops and functions

Logistics for both tracks

- Instructors Ari and Terence + wonderful TAs - 80 minutes per week - Not graded but will have a final project - Access to TAs for coding specific problems throughout the quarter

Key points: Class 2 vectors and data types

vectors and vectorized coding

- Vectors are the fundemental way to store data in R
- We can operate on vectors element-by-element without loops
  - `dplyr` verbs rely on this!
- We introduced built-in functions to build vectors and do operations on vectors.

data types

- (Atomic) Vectors have a single data type
  - most often: `logical`, `integer`, `double`, or `character`
- Certain operations expect a certain data type and will try to coerce the data if it can.
  - coercion can lead to unexpected behavior such as making `NA`s.

Up next: Watch videos for lab 3: using `ifelse` for control flow.

Exercise

Use R to calculate the sum

$$\sum_{n=0}^{10} \frac{1}{2^n} = \frac{1}{2^0} + \frac{1}{2^1} + .. + \frac{1}{2^{10}}$$

$$\sum_{n=0}^{10} \frac{1}{2^n} = 1 + 0.5 + .. + 0.00098$$

1. Use vectorized math to create a vector with the correct numbers
2. Use a built-in function to add up all the numbers in the vector.

Bonus What happens to the sum as you increase `n`?

Automatic type coercion

Type coercion is done automatically when R knows how. Usually, simpler types can be coerced to more complex types.

- `logical < integer < double < character`.

```
# paste0() is a function that combines two chr into one
paste0("str", "ing")
```

```
## [1] "string"
```

```
paste0(1L, "ing")
```

```
## [1] "1ing"
```

`1L` is an `int`, but R will coerce it into a `chr` in this context.

Automatic coercion

Logicals are coercible to numeric or character. This is very useful!

Determine the rule for how R treats `TRUE` and `FALSE` in math.

```
TRUE + 4
FALSE + 4
sum(c(FALSE, FALSE, FALSE, FALSE))
mean(c(TRUE, TRUE, FALSE, FALSE, TRUE))
```

Automatic coercion

```
TRUE + 4
```

```
## [1] 5
```

```
FALSE + 4
```

```
## [1] 4
```

```
sum(c(FALSE, FALSE, FALSE, FALSE))
```

```
## [1] 0
```

```
mean(c(TRUE, TRUE, FALSE, FALSE, TRUE))
```

```
## [1] 0.6
```

Question from Piazza Type coercion

@63

```
1/n

Error in 1/n : non-numeric argument to binary operator


1/as.numeric(n)

Error in as.numeric(n) :
  cannot coerce type 'closure' to vector of type 'double'
```

control flow

Key points: control flow with `if` and contingent column creation with `ifelse`

- Use `ifelse()` with `mutate()` to create new columns contingently.
  - `ifelse()` is vectorized so can operate on a logical vector to produce new results
- Understand how logical operators (i.e. `!`, `|`, `&`) work together with `ifelse` and conditional operators.
- Use `if()` (and `else`) to control whether an action is completed outside of a data context.

We also introduced `Rmd`s and saw how to knit the `Rmd` to html or pdf.

Up next: watch video for class 4 on using `group_by` to do grouped analysis.[1] Read the introduction to lab 4 before lab tomorrow!

[1] This is way more exciting than it sounds.

Execercise

We want to make a new column called `famous_storm` that is `1` for "Katrina" and "Rita" and `0` otherwise.

This code fails.

```
# bad code :(
storms %>%
  mutate(famous_storm =
           ifelse(name == "Katrina" | "Rita", 1, 0))
```

Find two alternative ways to write this code that work.

Example: Creating a simulation dataset

You want to understand the impact of discrimination on gifted education.

```
discrimination_simulation <-
  tibble(group = rep(c(1, 2), 5000),
         probabilty_gets_tested = runif(10000),
         iq = rnorm(10000))
```

- Students in group 1 get tested with probability 60 percent
- Students in group 2 get tested with probability 10 percent
- Students get gifted education if `iq > 1` and they're tested

Question from piazza

@87

What are the order of operations when using vectorized functions within `mutate()`?

```
practice <- tibble(col = c(1, NA, 1, NA, 4))

practice %>%
  mutate(col =
           ifelse(is.na(col),
                  var(col, na.rm = TRUE),
                  col)
         )
```

Question from piazza

@82 (@79)

"How to aggregate across rows?"

```
df <- tibble(sam = c(1, 0, 1, 1),
             casey = c(1, 1, 0, 0),
             abdul = c(1, 0, NA, 0))

df %>%
  mutate(rank = sum(sam, casey, abdul, na.rm = TRUE))
```

```
## # A tibble: 4 x 4
##     sam casey abdul  rank
##   <dbl> <dbl> <dbl> <dbl>
## 1     1     1     1     6
## 2     0     1     0     6
## 3     1     0    NA     6
## 4     1     0     0     6
```

Discussion questions

warm-up take aways

- `NA | TRUE` returns `TRUE`. Why does `FALSE | NA` return `NA`?
- `TRUE & NA` returns `NA`. Why does `FALSE & NA` return `FALSE`?

Grouped analysis

Key points: Grouped analysis with `group_by()`

- *groups* are a set of rows that belong together.
  - `group_by()` adds information about groups without changing the "data"
- Use `group_by()` with `summarize()` to create summary tables at group-level.
  - Use with functions that *reduce* data from a vector to a single value per group.
  - Expected output: a table with one row per group and one column per summary statistic and one column per grouping column.
- we can also use `group_by` to do grouped analysis with
  - `mutate` with window functions or to add a summary stat as column for further analysis.
  - it also can impact `arrange` and `filter`

Up next: watch video for class 5 on using `ggplot`. In lab 5, your

Exercise

Rewrite this code to

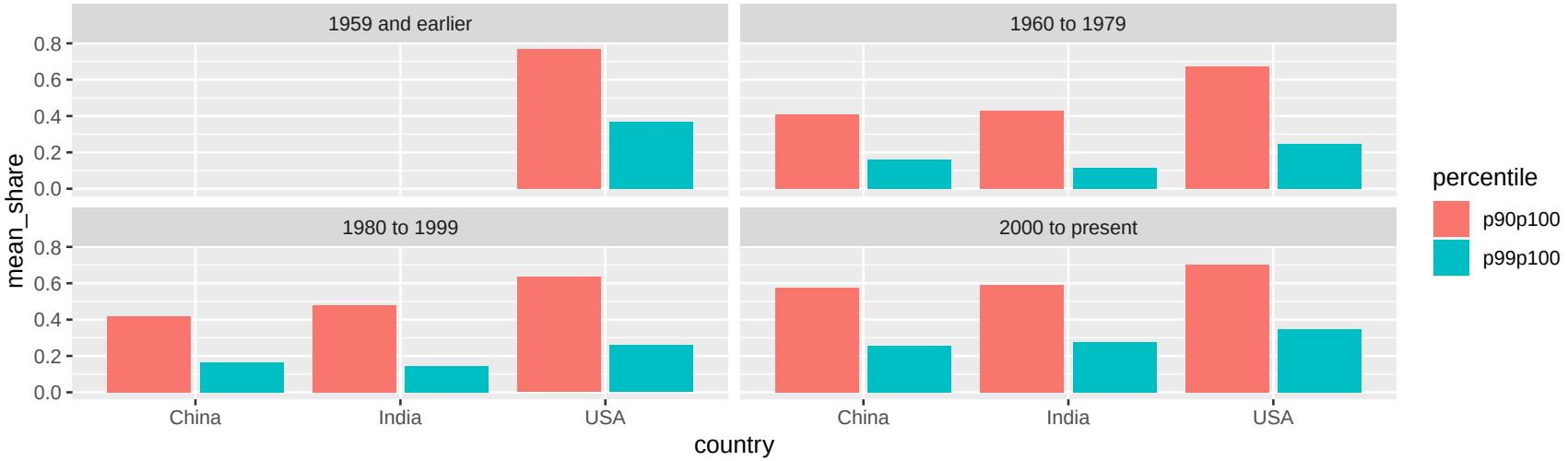
```
wid_data %>%
  filter(percentile == "p99p100")  %>%
  group_by(country) %>%
  summarise(mean = mean(value, na.rm = TRUE),
            sd = sd(value, na.rm = TRUE))

wid_data %>%
  filter(percentile == "p90p100")  %>%
  group_by(country) %>%
  summarise(mean = mean(value, na.rm = TRUE),
            sd = sd(value, na.rm = TRUE))
```

Discussion

How do we make the data for this graph?

- What "groups" are required for the visualization?

```
## Warning: Removed 4 rows containing missing values (geom_
```



Exercise

A student came during office hours and asked why `mean(percentile=="p90p100")` doesn't calculate the average wealth shares (`value`) for the top 10 percentile group.

```
wid_data %>%
  filter(country %in% c("China", "India", "USA")) %>%
  group_by(country) %>%
  summarize(p90_mean = mean(percentile == "p90p100",
                            na.rm = TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups`
```

```
## # A tibble: 3 x 2
##   country p90_mean
##   <chr>       <dbl>
## 1 China        0.25
## 2 India        0.25
## 3 USA          0.25
```

Where does n come from?

@98

```
disposition_by_race <-
   traffic_data %>%
     mutate(Disposition = str_to_lower(Disposition),
            Disposition =
              case_when(
                Disposition %in% citation_strings ~ "citation",
                Disposition %in% arrest_strings ~ "arrest",
                TRUE ~ Disposition)) %>%
     count(Race, Disposition) %>%
     group_by(Race) %>%
     mutate(freq = round(n / sum(n), 3))
```