summer_2021_qa

Ari Anisfeld

9/1/2021

Class 1: Reading files and 'dplyr'

Do now

Do now:

- Complete the intro poll at `bit.ly/acc_intro_poll`

After the poll

- Download `lab_1` from the course webpage: `harris-coding-lab.github.io`.

Notice

- Earlier we had a typo for the link to `lab_0` on canvas, which is now fixed. Sorry for inconvience.

Expectations

From you:

- do the work (i.e. watch video, try basics, do lab, bring questions)
- engage in course! (i.e. work with partners, answer questions, do polls)
- have R and RStudio installed!

From us:

- prepare engaging lesson materials
- address your questions
- help you be confident for core (confident != R expert)

From everyone:

- be nice to each other and create a growth-focused environment

Do the work

- Step 1. Videos
- Step 1a. Basics
- Step 2. QA
- Step 3. Lab

Not an expert

We cover:

- how to work with basic data structures (tibbles, vectors)
- how to read and manipulate data
- programmer logic (if statements, loops, functions)

We won't cover in depth:

- most statistical tools
- how to join data together
- how to convert data from long to wide (pivoting)
- how to deal with very messy data
- how to work with specific data types (e.g. dates, advanced strings)
- among other things like webscrapping, package development and so forth

Today's session

- Set up working directories
- Review some questions from QA
- Highlight key points and open up for live questions

Setting up working directory and coding environment

1. Do you have a folder on your computer for coding lab material?
   If not, create one and make sure you know the path to the folder.
2. We recommend creating a `problem_set` folder inside your coding lab folder.
3. Make folder called `data` inside the `problem_set` folder.

Putting your files in place

4. Create a new R script. Save your script in the `problem_set` folder. From now on, when you start a script or `Rmd` save it there.
5. Download the first data set here and put the data in your `data` folder. Find the link in the lab pdf!

Tell R where to find files

- Local paths are like addresses on your computer. Use `getwd()` to see how your computer paths look.

```
# In lab0 we downloaded data form a URL which is an address on the inte
covid_data <-
  read_csv(
    "https://data.cdc.gov/api/views/qfhf-uhaa/rows.csv?"
    )

# Compare to a local path
wid_data <-
  read_xlsx(
    "~/coding-lab/harris-coding-lab.github.io/data/world_wealth_inequal
  )
```

6. Add a line to your script where you `setwd()` to the data folder.

Working with the files

7. Finally, we are using data in an excel format. We need the package `readxl` to process data of this type. In the console, run `install.packages("readxl")`.
8. Add code to load the `tidyverse`.
9. If you followed the set-up from above, you should be able to run the following code with no error.

```
wid_data <- read_xlsx("world_wealth_inequality.xlsx")
```

What to do when something is confusing?

- use ?
- test code in console. try to break it.
- ask teammates / try googling
- ask us!

If it's not "mission critical", you can safely move on without full understanding. (Imagine learning a language and trying to figure out all the grammar and vocabulary at the same time!)

Question:

- What's the deal with `col_types = cols(Suppress = col_character())`?
- Do we need that "accessType=DOWNLOAD&bom=true&format=true%20target=" part?

```
covid_data <-
  read_csv(
    paste0("https://data.cdc.gov/api/views/qfhf-uhaa/rows.csv?",
           "accessType=DOWNLOAD&bom=true&format=true%20target="),
    col_types = cols(Suppress = col_character()))
```

- Note: In URLs after the ? you send meta information about your request.

Question: Can you explain pipes?

- Pipes %>% take the left hand side and put them into the first position on the right hand side.

```
storms %>% filter(year > 2010) %>% glimpse()

recent_storms <- filter(storms, year > 2010)
glimpse(recent_storms)
```

Notice

- `filter()` takes data in the first position and then an arbitrary number of filtering expressions.
- `glimpse()` takes data in the first position

Lesson 0: Intro to R, RStudio and the `tidyverse`

- navigate and use Rstudio's features
  - particularly, the console, the text editor and help
- assign objects to names with <-
- use functions by providing inputs and learn more with ?
- `install.packages()` (once) and then load them with `library()` (each time you restart R)

Lesson 1: Key points: 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.

Lesson 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%`.
- Make new columns with `mutate()`.
- Sort data with `arrange()` and `arrange(desc())` or `arrange(-x)`.
- Create summary statistics with `summarize()`.

Class 2: Vectors and data types

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.

lab 1 solutions will be available on the course website.

Getting started with Rmarkdown (Rmd)

- What's an Rmd?
- How to make an Rmd
- How to work with an Rmd

Knitting: making the frustrating part less frustrating

- Install `tinytex`

```
install.packages("tinytex")
tinytex::install_tinytex()
```

- Knit early and often.

When to use Rmds vs scripts?

Rmd

- Exploration of data
- Presentations and reports

script

- Projects with interrelated code (e.g. an R package)
- Working on a server that does not have Rstudio installed

Questions from QA

Question 1: - Why do I need the function `summarize` in the following bit of code?

```
michigan_population_total <-
  midwest %>%
    filter(state == "MI") %>%
  summarize(total_pop = sum(poptotal))
```

- Why can't I just pipe directly into `sum`?

Question 2: Why do we need to use `pull()`?

R's primary data structures: Vectors vs. tibbles

- Why do we need different data structures?
- How are vectors and tibbles related?

Why do we need different data structures?

In theory, we can do all our work with vectors.

```
names_vec <- c("Ari", "Qiwei", "Jay", "Thomas")
surnames_vec <- c("Anisfeld", "Lin",  "Zaleski", "Whamond")
position_vec <- c("Instructor", "TA", "TA", "TA")
```

why might I want a tibble?

R's primary data structures: Vectors vs. tibbles

Tibbles encapsulate vectors

```
names_vec <- c("Ari", "Qiwei", "Jay", "Thomas")
surnames_vec <- c("Anisfeld", "Lin",  "Zaleski", "Whamond")
positions_vec <- c("Instructor", "TA", "TA", "TA")

my_tibble <- tibble(names = names_vec,
                    surnames = surnames_vec,
                    positions = positions_vec)
```

- Keep data tidy -> rows are a single observation or record.
- Keep meta-data (column names)
- Keep track of relationships between vectors
- Can hold various data types

R's primary data structures: Vectors vs. tibbles

- vectors are simpler
- have a single data type
- some functions expect vectors or make more sense on them.

Reviewing 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 vectors into a single vector
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
```

Exercise

1: 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`?

2: Use `paste0()` to convert `v1` and `v2` into `"hello!"`

```
v1 <- c("h","l", "o")
v2 <- c("e", "l", " !")
```

```
## [1] "h el lo  !"
```

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.

Over weekend: Attempt lab 2. **For Tuesday:** Watch video about control flow + try basics.

Class 3: Control flow

Outline

- lingering questions about Rmds
- `ifelse()` questions

Reviewing the anatomy of an Rmd:

Write text in the document

```
# ^^^ start an R chunk ```{r}
# sometimes {} have meta information

# R code goes in a chunk
ex <- seq(1, 12)

# and output prints below
print(ex)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

A meta examples: {r, echo = FALSE}

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

Another meta examples: {r, eval = FALSE}

```
print(ex)
```

Naming chunks

Chunk names help you debug

```
# code
```

Two chunks cannot have the same name

```
# If I change the name to "example" we will get an error.
```

Quick hitters

Q: Why don't we get LaTex with $ in this example?

```
$$
  \sqrt{p}
$$
```

Quick hitters

Q: Why don't we get LaTex with $ in this example? *A:* In a code chunk, knitr expects R code! So a dollar sign is interpreted to be referring to a named entity in an object like `data$column_name`.

So put LaTex in the "text" area of an Rmd!

$$\sqrt{p}$$

Quick hitters

Q: What does this error tell you?

```
setwd("~/Documents/coding")
```

Error in setwd("~/Documents/coding") : cannot change working directory

Quick hitters

Q: What does this error tell you? *A:* Usually, it means your directory name is wrong somehow!

```
setwd("~/Documents/coding_lab/")
```

Note: "~/" does not work on Windows!

An aside on relative paths

You can refer to directories **relative** to your current directory using `.` and `..`

- `.` means current directory.
- `..` means "go back" the directory path.

an example

File structure:

- coding_lab datafile.csv
  - problem_sets my_rmd.Rmd

You could access the data with `read_csv("../datafile.csv")` in a chunk.

What's the difference between & and &&?

- Test your hypothesis with additional examples in the console?

```
c(TRUE, TRUE) & c(TRUE, FALSE)
```

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

```
c(TRUE, TRUE) && c(TRUE, FALSE)
```

```
## [1] TRUE
```

- Which plays nicely with `ifelse()`?
- Which plays nicely with `if()`?

What's the difference between | and ||?

Similarly OR has a vectorized | and singleton || version.

- generally, you can get by with | and &!
- when you are working on code for general use (like writing a package) there will be times when you want to ensure

Exercise

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))


storms %>%
  mutate(famous_storm =
           ifelse(name == "Katrina"| name == "Rita", 1, 0))


storms %>%
  mutate(famous_storm =
           ifelse(name %in% c("Katrina", "Rita", "Amy", "Bo
```

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),
         prob_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

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: prepare lab 3 for tomorrow. **Friday** watch video for class 4 on using `group_by` to do grouped analysis.

Key points: `if()` versus `ifelse()`

	`if()`/ `else()`
Used to conditionally evaluate code	yes
Vectorized?	no, only uses first element
Handles NA	no, error `missing value where T`
baseR	yes

- *Takeaway:* When we are focusing on data analysis use `ifelse()` (or `if_else()`).

[1] there's a tidyverse `if_else()` that works slightly differently

Quick hitters

Class 2 basics

- *Q* Why did I get double when your code shows the type is an int?

```
typeof(seq(1, 12, 1))
```

```
## [1] "double"
```

Quick hitters

Class 2 basics

- *Q* Why did I get double when your code shows the type is an int?
- *A* In base R, data types are not always predictable.

```
typeof(seq(1, 12))
```

```
## [1] "integer"
```

```
typeof(seq(1, 12, 1))
```

```
## [1] "double"
```

Tidyverse functions tend to be more careful to avoid this sort of behavior.

Question from QA, what's the deal with subsetting?

How many ways can you pull out the even numbers from `vec`?

```
vec <- 1:10
vec[c(2, 4, 6, 8, 10)]
```

```
## [1]  2  4  6  8 10
```

```
vec[rep(c(FALSE, TRUE), 5)]
```

```
## [1]  2  4  6  8 10
```

```
vec[seq(2, 10, 2)]
```

```
## [1]  2  4  6  8 10
```

```
vec[vec %% 2 == 0]
```

```
## [1]  2  4  6  8 10
```

Notice `filter()` works like the final option!

Horoscope game:

Make a game where you ask the user to enter their birth month and you tell them their fortune.

- Give people born in December, January or February a "cold" fortune
- Give people born in June through September a "warm" fortune
- Give people born in November a great fortune
- Give everyone else an okay fortune

e.g. `birth_month <- 2` the code should return something like `I see penguins in your forecast`.

Class 4: Grouped analysis

practice

We are working in `our_rmd.Rmd` with file structure like so:

- dir1 file1.csv
  - dir2 our_rmd.Rmd
  - dir3 file2.csv

Question

- What is our working directory?
- How do we access file2.csv?
- How do we access file1.csv?

Class 5: Loops

Class 6: Functions