STAT 215A Fall 2023 Week 2

Chengzhong Ye

Announcements

- Make sure your **stat-215-a** repo is private
- Lab 1 and HW 1 will be released by the end of today. **Due Friday Sep 22 at 11:59pm**
- Lab 1 will be submitted to your **stat-215-a** repo. HW 1 will be submitted to **Gradescope**

GitHub repos I have access to:

Yangch301
riverrbell
ShuyiJudyYang
rpalmaka
matutinus
mchxo
Yishu09
maya-madhavan
armwong9
lijiaxun-smart
NicolasNunezSahr
b-huck77
jasiakm
jmorimoto125
eric-shchiu
jv-rv
dylanwebbc
tzhou2801
anthonyozerov
valeskafk
stephen-quiton
zachrewolinski
miloscola
zhiweixiao

If you don’t see your github id, please email me.

Today’s outline

- `here()`
- Workflows
- Lecture on data cleaning
- Lab 1 Introduction


here: find your PATH!
seriously, just here.
"/NOT/HERE"
here!
IT'S ALL RIGHT HERE!

here()

here()

- here is a very simple package that **increases reproducibility**
- When you run library(here) it checks the current working directory (i.e. whatever getwd() returns) for:
  - A file named .here
  - An RStudio project: foo.Rproj
  - An R package: DESCRIPTION
  - A git repo: git
  - Some others
- If it doesn't find any of those, it moves up to the parent directory and starts over.

`here()` example

PWD is `week2`

`here()` starts at the git repo top-level directory

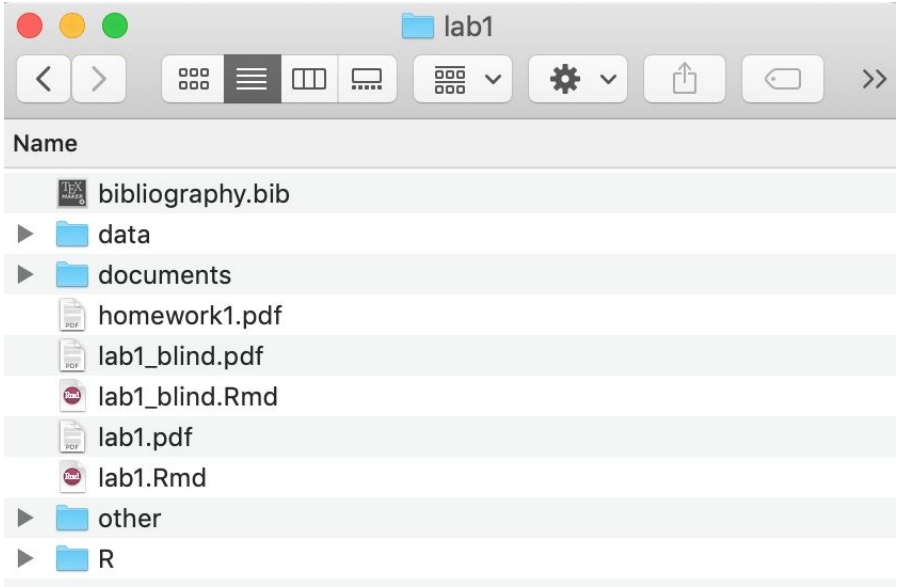
```
> getwd()
[1] "/home/james/school/215a/stat-215a-fall-2020/week2"
> library(here)
here() starts at /home/james/school/215a/stat-215a-fall-2020
> here()
[1] "/home/james/school/215a/stat-215a-fall-2020"
> here("week2", "data", "mtcars.rds")
[1] "/home/james/school/215a/stat-215a-fall-2020/week2/data/mtcars.rds"
> mtcars2 <- readRDS(here("week2", "data", "mtcars.rds"))
> head(mtcars2)
                  mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
```

`here()` concatenates the path

Getting prepared for Lab 1

Project file structure

- **data/:** store raw and processed data
- **documents/:** store relevant papers, instructions, meeting notes, etc.
- **R/:** store R code, utility functions, scripts
- **other/:** miscellaneous



Project File Structure

R/

- **load.R** – file containing function(s) for reading in the data

```
> loadData(path_to_data)
```

- **clean.R** – file containing function(x) for cleaning the loaded data

```
> cleanData(loaded_data)
```

data/

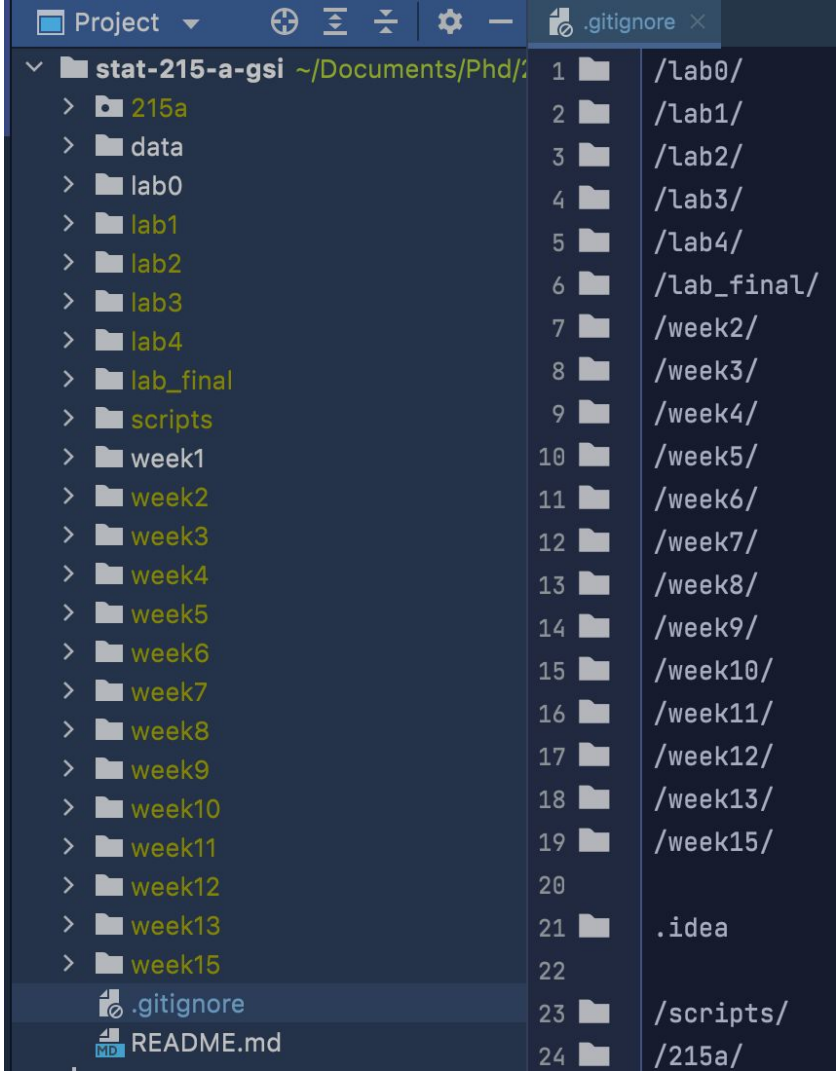
- Contains datasets
- Not uploaded to GitHub (can automate this using .gitignore)

.gitignore

The `.gitignore` file is a text file that tells Git which files or folders to ignore in a project.

e.g.

./lab1/data



Project File Structure

lab1.Rmd – your final report combining code (not printed in the output) and text/narrative

- Should be written like a paper; focus on communicating well

lab1.pdf – pdf output from lab1.Rmd

lab1_blind.Rmd – same as lab1.Rmd but without name

lab1_blind.pdf – pdf output from lab1_blind.Rmd

explore.Rmd (optional) – a separate .Rmd file that contains your exploratory code and figures

- A useful place for exploring the data and saving avenues of exploration that you don't necessarily want to include in your final report

bibliography.bib (optional) – a .bib file for easy citations within the lab reports

Workflow: General Tips

Make code readable

- Be kind to both your peer reviewer and your future self

Keep your code modular – write functions

- Separate your functions from your analysis file (lab1.Rmd) and store them in R/
- In doing so, you create a bank of useful functions that you can load into any analysis script for your project (or future projects)
  - To load in a single file:

    ```
    > source("./R/filename.R")
    ```

  - To load in all files in the R/ directory:

    ```
    > library(R.utils)
    > sourceDirectory("./R/", modifiedOnly = F, recursive = F)
    ```

- Group together related functions in the same .R script

  (e.g. put all data cleaning functions in clean.R)

Workflow: General Tips

Documentation

- Write lots of comments in your code and ask yourself: why are you writing this particular piece of code?
- Document functions (think about the R help pages)
  - Always add comments section immediately below the function definition line
  - What does this function do?
  - Describe the inputs and outputs

```
CalculateSampleCovariance <- function(x, y, verbose = TRUE) {
  # Computes the sample covariance between two vectors.
  # Args:
  #   x: One of two vectors whose sample covariance is to be calculated.
  #   y: The other vector. x and y must have the same length, greater than one,
  #      with no missing values.
  #   verbose: If TRUE, prints sample covariance; if not, not. Default is TRUE.
  # Returns:
  #   The sample covariance between x and y.
...
}
```

Workflow: General Tips

Test your code

- Write tests to make sure your functions are doing the right thing
- Write these tests as you go

Don’t Repeat Yourself (DRY)

- If you find yourself copying and pasting similar lines of code, write a reusable function instead

Establish consistencies – follow Google R Style Guide

Workflow: Code Style

Follow Google's R Style Guide when writing code

(See https://google.github.io/styleguide/Rguide.xml and part I Analyses of https://style.tidyverse.org/syntax.html#object-names)

Variable names

- All lowercase
- Separate words by "." or "_" (be consistent with the one you choose)

Good: `avg.tmp, avg_tmp`

Bad: `AvgTmp`

Workflow: Code Style

Function names

- Camel-case
- Make function names verbs

Good: `CalculateAvgClicks, calculateAvgClicks`

Bad: `calculate_avg_clicks, calculuate.avg.clicks`

Line Length: maximum length of 80 characters

- Go to: Preferences > code > display > check show margin and set margin column = 80

Indentation: When indenting your code, use two spaces (rather than tabs)

Workflow: Code Style

Spacing

- Place spaces around all binary operators (=, +, –, <–, etc.)
- Always put a space after a comma, never before, just like in regular English

Good:
```
df.prior <- df[df$days.from.opt < 0, "campaign.id"]
x[, 1]
```

Bad:
```
calculate_avg_clicks, calculuate.avg.clicks
x[,1], x[ , 1]
```

Assignment

- Use <- instead of =

Workflow: Code Style

Most importantly, BE CONSISTENT

Data Cleaning Lecture

Lab 1
PECARN TBI Data

Due: Fri,
Sep 22 @ 11:59pm

Lab 1 Goals

Data cleaning

Exploratory Data Analysis and Visualization

Lab 1 PECARN TBI Introduction

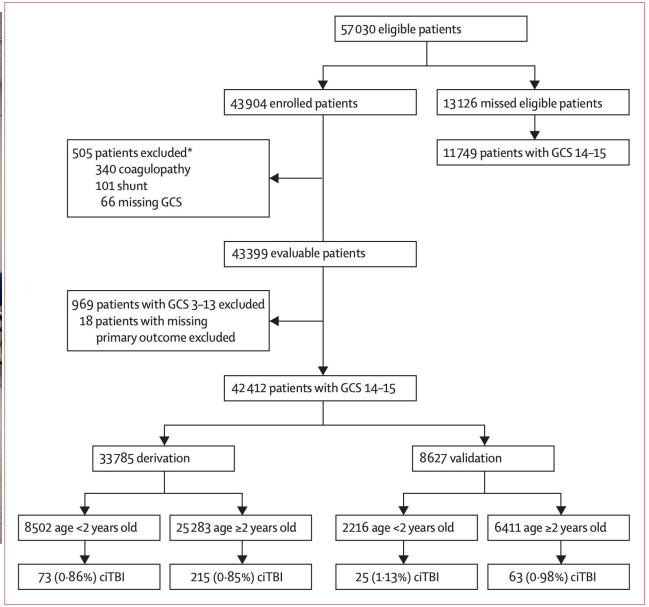
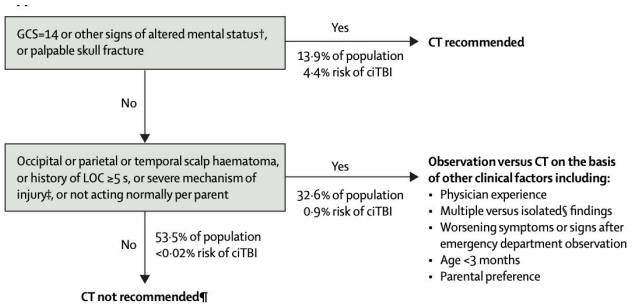


Figure 1: **Flow chart**
GCS=Glasgow Coma Scale. ciTBI=clinically-important traumatic brain injury. *Two patients had more than one exclusion.



Lab 1 Introduction

- Read the paper carefully **Kuppermann_2009_The-Lancet_000.pdf** in the **lab1/documents** folder on GitHub. For descriptions of the features included in the data set, see **TBI PUD Documentation 10-08-2013.xlsx** in the **lab1/data** folder on GitHub.
- The **lab1** folder will also contain a template to follow when putting together your lab
- Do **not** push **data/** folder to GitHub.
    - Can easily do this with `.gitignore` file

Lab 1 outline

PCS documentation

- **Introduction/Domain problem formulation**
  - Describe the background and the goal of the study, how this dataset can address the problem.

- **Data collection**
  - How the data is collected? Refer to data documentations. Discuss anything you are uncertain about.

- **Data cleaning**
  - Checklist: Invalid values, Missing values, Data format, Column names, Variable type, Other dataset specific explorations

Lab 1 outline

- **Data exploration**
  - visualize your data in a general way. For example,

    - A table reporting the min, max, and average of each numeric variable

    - Histograms or boxplots of continuous numeric variables (perhaps separating the data into meaningful groups)

    - Bar charts for discrete variables

    - A scatterplot to show the relationships between a few select pairs of variables (or a scatterplot matrix, if you're into those)

Lab 1 outline

- **Three findings**
  - Interesting observations of the data. Make one high-quality plot for each finding
- **Reality Check**
  - Compare you cleaned data with reality
- **Stability Check**
  - Perturb one judgement call in one of your findings and compare the results
- **Discussion**

Collaboration Policy

- You are allowed to discuss **ideas** with others, but you must submit your own report
- Do not share code or copy/paste any part of the writeup
- If you do discuss ideas with others, be sure to acknowledge these students in your report

Lab 1 Rubric

PECARN Data Lab (~60 points)

- Readability and grammar
- Readability of code (+ comments)
  - Follow Google's R Style Guide (a slight modification of the Tidyverse Style Guide)
- Reproducibility of report
  - I should be able to pull your `lab1/` folder from GitHub, manually add the `data/` folder, open `lab1.Rmd`, click knit, and get the same .pdf file as you.
- Data cleaning (description and validity)
  - Describe *any* problems/inconsistencies you see with the data, how you cleaned the data, and why you cleaned the data in that way
- Three findings (creativeness, interestingness, and quality of figure)
  - Fix titles, axis and legend titles, choose appropriate color schemes, adjust size of figure
- Figures that are not for the findings (relevance and quality)
- Overall quality and level of detail of report
  - Attempts to incorporate domain information (from the paper) and place your analysis in the domain context

Homework – Some Basic Statistics (8 points)

Start Early!!!!!!!!

Week 2 exercise

- About Gapminder: https://www.gapminder.org/about/
- Resources for this tutorial:
  - ggplot: http://swcarpentry.github.io/r-novice-gapminder/08-plot-ggplot2/
  - dplyr: http://swcarpentry.github.io/r-novice-gapminder/13-dplyr/
- See **lab_gapminder.Rmd** in the week2 folder on my GitHub