Methods Camp 2025: Day 1

Christina Pao, Sofia Avila, Florencia Torche

August 30, 2025

Table of contents

Intro

Welcome to Methods Camp!

We are excited to welcome you to Princeton and to help you prep for your first year. :)

Instructor Introductions

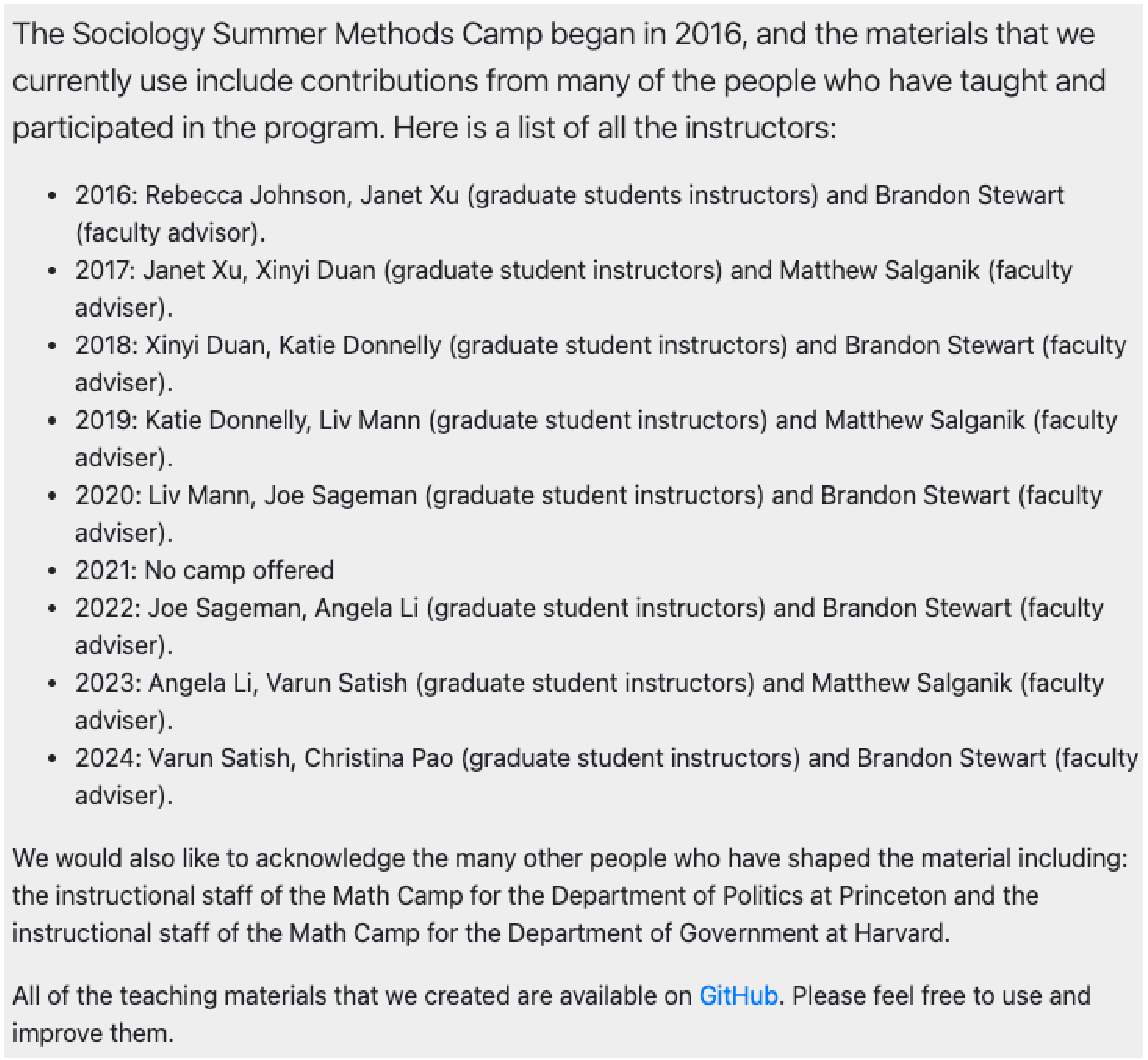
- Christina (any pronouns)
  - G4: Sociology & Social Policy, OPR affiliation
- Sofia (she/her)
  - G3: Sociology & Social Policy, OPR affiliation
- Florencia Torche (she/her)
  - Faculty advisor of Methods Camp and instructor of SOC 500(!)

Why Methods Camp?

Why are we excited about teaching you methods?

How do we think about methods?

The Methods Camp Legacy



The Sociology Summer Methods Camp began in 2016, and the materials that we currently use include contributions from many of the people who have taught and participated in the program. Here is a list of all the instructors:

- 2016: Rebecca Johnson, Janet Xu (graduate students instructors) and Brandon Stewart (faculty advisor).
- 2017: Janet Xu, Xinyi Duan (graduate student instructors) and Matthew Salganik (faculty adviser).
- 2018: Xinyi Duan, Katie Donnelly (graduate student instructors) and Brandon Stewart (faculty adviser).
- 2019: Katie Donnelly, Liv Mann (graduate student instructors) and Matthew Salganik (faculty adviser).
- 2020: Liv Mann, Joe Sageman (graduate student instructors) and Brandon Stewart (faculty adviser).
- 2021: No camp offered
- 2022: Joe Sageman, Angela Li (graduate student instructors) and Brandon Stewart (faculty adviser).
- 2023: Angela Li, Varun Satish (graduate student instructors) and Matthew Salganik (faculty adviser).
- 2024: Varun Satish, Christina Pao (graduate student instructors) and Brandon Stewart (faculty adviser).

We would also like to acknowledge the many other people who have shaped the material including: the instructional staff of the Math Camp for the Department of Politics at Princeton and the instructional staff of the Math Camp for the Department of Government at Harvard.

All of the teaching materials that we created are available on GitHub. Please feel free to use and improve them.

The materials we have put together draw heavily on materials developed by past preceptors and faculty instructors!

Logistics of Camp

Dates

Saturday, August 30

Friday, September 5

Saturday, September 6

- NB: Makeup dates are available the following Mondays as needed! Talk to Christina and Sofia.

General Daily Layout

9-9:30am Breakfast

9:30am-12pm Coding Activities / Lecture

12pm-1pm Lunch

1pm-2pm Coding Activities / Lecture

2pm-3pm Group Assignments

Our Goals

Learning Objectives

- Ensure that all students feel confident with using R for SOC 500
- Empower students to learn where to find help when they’re stuck

Personal Objectives

- Meet your peers in a structured but informal space!
- Meet two upper-years who are very happy to be in touch & connect you to others in the department!

Assignments

Summer Pre-assignment

Questions for you:

1. How was Gradescope?
2. How did you find using Quarto?
3. Were there any things that you all wanted to clarify as a class?

Notes from us:

1. Remember to select pages for each question when you're submitting to Gradescope!
2. No need to load libraries multiple times! (NB: `library(tidyverse` has many subsidiary packages in it!)
3. Remember to set up code chunk settings correctly and linebreak your comments/codes! It'll make your output more readable.

In-class Assignments

- Ideally, no take-home work!
- At the end of each methods camp day, we’ll match you with a randomized buddy/set of buddies. You’ll work with them on a task based on the day’s materials.
- Only one person in the pair has to email at the end of the day.

Philosophy of Pair/Group Programming

- Person who has **less experience** should be the one at the keyboard (chalkboard).
  - Allows the person who knows more to practice the ability to explain their knowledge / teach
  - Also keeps it from devolving to the person who knows more doing everything.

AI Tools

How many people use AI (generally)? For their coding?

Ways to Use AI

- Prompt Engineering: *I am a social scientist using tidyverse to analyze a dataset called addh. It has the following columns: age, gender…. Why doesn't the following code doesn't run?*
  - Create a question persona
  - Specify packages
  - Specify basic about the data
  - Ask to explain
  - Keep code input small
- Show code chunk AND code output/error message.

AI in Methods Camp & SOC 500

Rule of Thumb: If AI isn't *writing your code for you*, AI can be a useful learning tool! At this stage, it means you can't use Copilot for course materials.

In the past, we have asked students who use AI to submit a record of their conversations with ChatGPT/Claude etc… Check with Florencia about requirements for the course (and for SOC 504 later down the line!)!

Example AI Use Cases

- Debugging a code chunk with a discrete error message
- Package/function support (essentially a supercharged StackOverflow search)
- Assisting with problem set formatting questions ($\LaTeX$)

AI in my materials!

- Topic aggregation for assignment flow
- Incorporating multiple useful R packages practically
- Creating unifying topic/story for the problem

This was all predetermined via conversation and all materials were scaffolded—but Claude was a helpful aggregating tool!

Questions?

- Does everyone have R installed and can open it *right now*?
- Did folks get the camp Day 1 materials that we shared this morning?

Let’s get started!

Today’s Material

Outline

1. Getting Started in R
2. Managing Workflow
3. R Settings
4. Objects in R
5. Tidyverse and Data Manipulation
6. Recoding Variables
7. Matrices

Getting Started in R

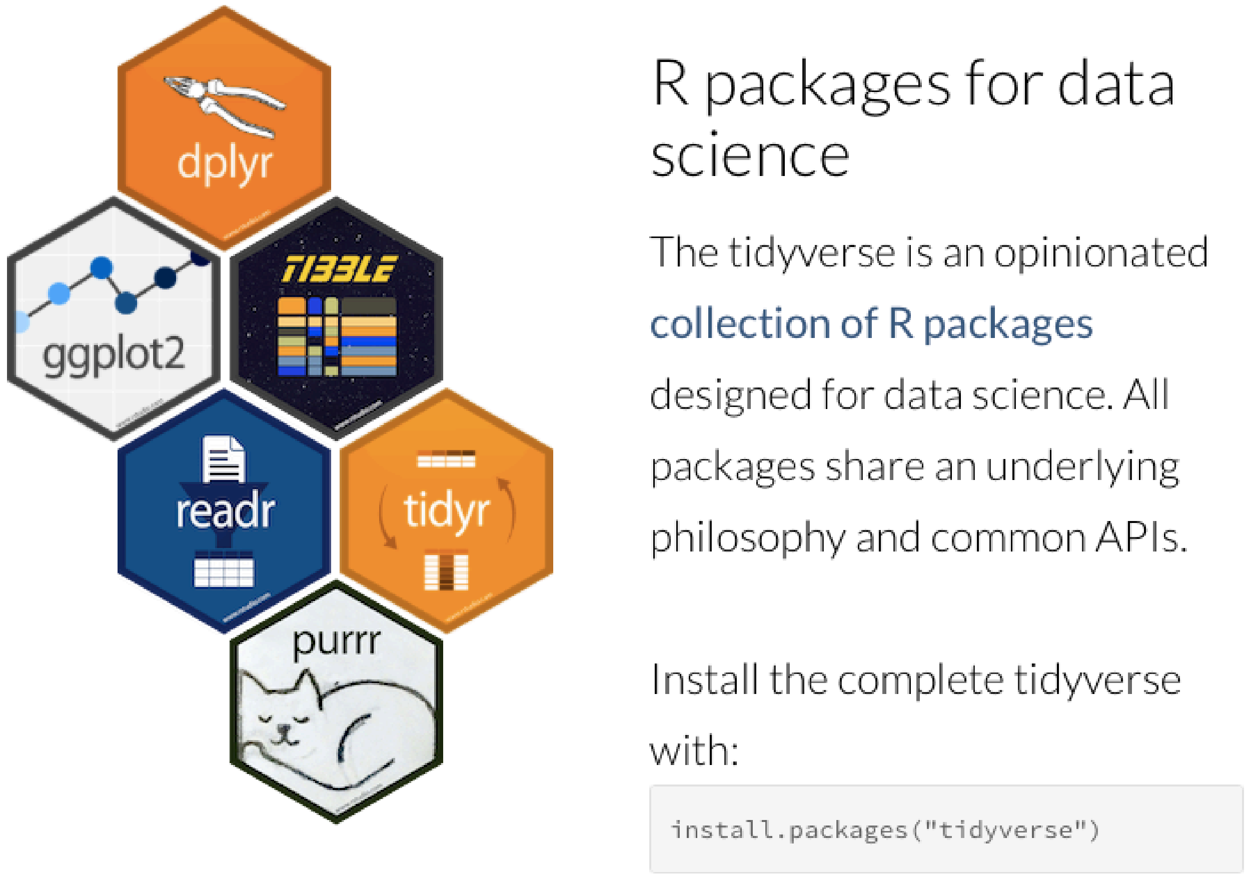
- R originated from statisticians: maximize statistical performance.
- Most recently R is used by a much wider group, including computer scientists and social scientists.

Reproducibility

- There is a distinct movement to push towards reproducible and readable code with a certain lean:
  - Consistency between readability across languages
  - Reproducibility and Version Control
    - write code for humans, write data for computers
    - commenting extensively
    - avoid duplicate/incremental files
    - produce markdown documents with all reproducible steps included etc.

Tidyverse

All of this culminates in the Tidyverse (a philosophy and a collection of packages).



R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying philosophy and common APIs.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

What we teach

We will focus on the `tidyverse` to start!

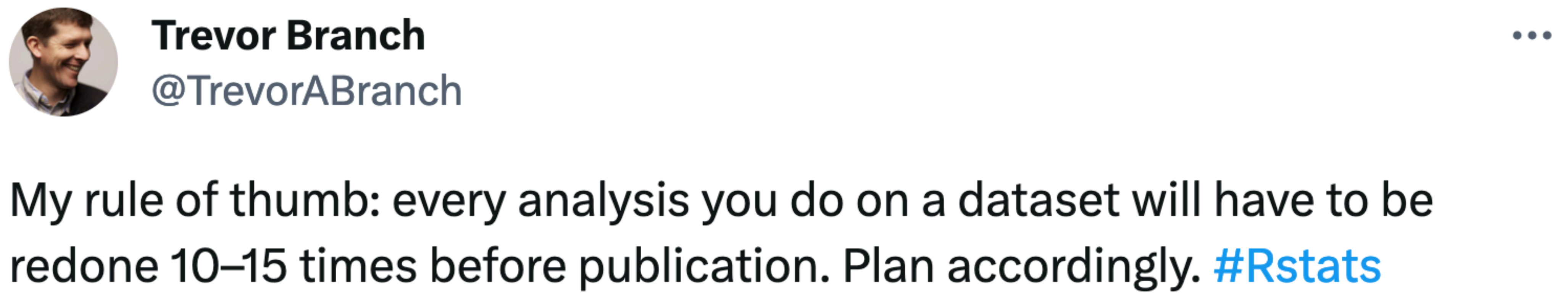
- Why?
  - For many, it is easier to understand
  - You can do most data manipulation you need in social science research with these tools

Other options…

We will also expose you to “base R” family functions like $ and [ ], loops and conditionals, data types, and *apply*:

- Why?
  - It is occasionally easier to use base R functions, though almost all of the data manipulation you’ll need to do has a good tidyverse equivalent.
  - If you want to write an R package or need to debug a weird issue, understanding **base R** will be useful
- For more complicated analyses, you may want to look into **data.table** for certain speed improvements

Good practices for research code


Trevor Branch
@TrevorABranch

My rule of thumb: every analysis you do on a dataset will have to be redone 10–15 times before publication. Plan accordingly. #Rstats

Project Organization

Project Organization WILL Save You

- Expect that you’ll have to run every piece of research code multiple times in the process of writing a paper
- Expect that you’ll want to be able to share your results and figures in a reproducible way
- The earlier you learn good practices, the better!

How to keep your code organized[1]

- Project-oriented workflow (R projects, `here()` package)
- Keep everything you need in source (ie, your Rmd or .R file)
- Use filenames readable by you and your computer
- Split up your research code into smaller files for cleaning, analysis, and figures
- Keep a copy of your raw data untouched, and create intermediate data output as you go

Project Workflow and R Settings

Exercise/Live Demo:

1. Set up an R project and organize the Methods Camp files we sent you in a reasonable way.
2. Set up an R project and file folders (preemptively!) for SOC 500.
3. Discuss some of the key settings features of RStudio that you may want to toggle with.

Helpful Keyboard Shortcuts[1]

Shortcut	Mac	PC
New document	Cmd+Shift+N	Ctrl+Shift+N
Change working directory	Ctrl+Shift+H	Ctrl+Shift+H
Knit/Render	Cmd+Shift+K	Ctrl+Shift+K
Insert code section	Cmd+Option+I	Ctrl+Alt+I
Run current line/selection	Cmd+Return	Ctrl+Enter
Run current document	Cmd+Option+R	Ctrl+Alt+F
Pipe operator (dplyr)	Cmd+Shift+M	Ctrl+Shift+M

R Basics: Code-along

Exercise/Live Demo:

1. Open up the `day1_morning_codealong.qmd`
2. Practice annotating your code!

This is not just an exercise of reviewing R but also making materials that you can easily refer back to (and copy and paste from!) later down the line for SOC 500 and onward!

Data Types: Atomic Vectors

Atomic vectors are the building block of the data structures in R. They contain elements of the same type.

```r
# numeric
a <- c(2, 5.6, 9)
class(a)
```

```
[1] "numeric"
```

```r
# Integer
c <- 1:10
class(c)
```

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

```r
# Character
d <- c("Christina", "Sofia")
class(d)
```

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

```r
# Logical
e <- c(5>3, 5<3)
class(e)
```

Data Structures: Matrices

Matrices are 2-dimensional arrays with elements of the same type.

```r
# Creating matrices
mat1 <- matrix(1:12, nrow = 3, ncol = 4)
mat2 <- matrix(c(1, 2, 3, 4), nrow = 2, byrow = TRUE)
```

```r
# Accessing elements
mat1[2, 3]       # Element at row 2, column 3
```

```
[1] 8
```

```r
mat1[1, ]        # First row
```

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

```r
mat1[, 2]        # Second column
```

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

Matrix Operations

```r
dim(mat1)          # Dimensions
```

```
[1] 3 4
```

```r
nrow(mat1)         # Number of rows
```

```
[1] 3
```

```r
ncol(mat1)         # Number of columns
```

```
[1] 4
```

```r
print(mat1)         # Viewing the matrix
```

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

Data Structures: Lists

Lists can contain elements of different types and lengths. (You may see this from regression output, for instance!)

```r
# Creating lists
person <- list(
  name = "Christina Pao",
  age = 27,
  camp_instructor_years = c(2024, 2025))
```

List Operations

```r
length(person)          # Number of elements
```

```
[1] 3
```

```r
names(person)           # Element names
```

```
[1] "name"                  "age"                   "camp_instructor_years"
```

Accessing List Elements

```r
person$name                    # Using $
```

```
[1] "Christina Pao"
```

```r
person[["age"]]          # Using [[]]
```

```
[1] 27
```

```r
person[[1]]              # By position
```

```
[1] "Christina Pao"
```

```r
person["camp_instructor_years"]      # Returns a list
```

```
$camp_instructor_years
[1] 2024 2025
```

```r
person[["camp_instructor_years"]]   # Returns the vector
```

```
[1] 2024 2025
```

Data Structures: Data Frames/Tibbles

Data frames are the most common structure that we will be working with for social science (tabular) data!

```r
# Creating data frames
df <- data.frame(
  id = 1:4,
  name = c("Christina", "Sofia", "Will", "Max"),
  camp_instructor = c(T, T, F, F)
)

df <- tibble(
  id = 1:4,
  name = c("Christina", "Sofia", "Will", "Max"),
  camp_instructor = c(T, T, F, F)
)
```

Accessing data frame elements

```r
df$name                  # Column by name
```

```
[1] "Christina" "Sofia"     "Will"      "Max"
```

```r
df[, "camp_instructor"]        # Column by name (alternative)
```

```
# A tibble: 4 × 1
  camp_instructor
  <lgl>
1 TRUE
2 TRUE
3 FALSE
4 FALSE
```

```r
df[2, ]                  # Second row
```

```
# A tibble: 1 × 3
     id name  camp_instructor
  <int> <chr> <lgl>
1     2 Sofia TRUE
```

```r
df[1:2, c("name", "camp_instructor")]  # Subset
```

```
# A tibble: 2 × 2
  name      camp_instructor
  <chr>     <lgl>
```

```
1 Christina TRUE
2 Sofia     TRUE
```

Using Tidyverse: Data Exercise

Data: 3rd wave of AddHealth containing people's ratings of how important the respondent believes the following are for a "successful marriage or serious committed relationship":

- love
- fidelity (no cheating)
- money

Documentation for AddHealth is online[1].

Loading the Data

- When you use an R project, the working directory is automatically set to where the .Rproj file is located
  - R’s commands for reading in data are specific to the file type – the most common is read_csv() for csv files
  - Another common one is importing foreign data types like STATA .dta files using haven::read_dta()

Installing/calling packages

```r
# install.packages(tidyverse)
library("tidyverse")
```

Check Working Directory

```r
getwd()
```

```
[1] "/Users/cp5045/Library/CloudStorage/Dropbox/MethodsCamp/2025/Slides"
```

Reading in Data

```r
## read in from where .Rproj file is
addh <- read_csv("addhealthlec1.csv")
```

Preliminary: explore data

When you load a tabular dataset, you should:

- check the data types of each variable
- check the dimensions of the data
- look at a few rows and variables

To look at the type for a single variable, use **base R notation with a $**:

```r
class(addh$age)
```

```
[1] "numeric"
```

```r
class(addh$gender)
```

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

Preliminary: explore data

To look at the type of an R object, put the whole object into the `class` function:

```r
class(addh)
```

```
[1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

Notes:

- You can see it is a tibble (a tidyverse dataframe, indicated by `tbl_df`)
- You can use certain functions like `class()` and `summary()` on vectors as well as dataframes

Preliminary: explore data

To get information about the whole tibble, use the following functions:

- `summary()`:
- `str()`:
- `colnames()` or `names()`:
- `dim()`:
- `View()`:
- `head()`:
- `tail()`:
- (in dplyr) `slice_sample()`:

Think/Pair/Share: What do each of these functions do? Run `?head()` to look at the help documentation for the function in R.

Preliminary: explore data

To get information about the whole tibble, use the following functions:

- `summary()`: numeric summaries
- `str()`: data types and sample data
- `colnames()` or `names()`: names of columns/variables
- `dim()`: dimensions
- `View()`: view all data in RStudio viewer (can be slow if data is large)
- `head()`: top 10 rows, can adjust `n`
- `tail()`: bottom 10 rows, can adjust `n`
- (in dplyr) `slice_sample()`: randomly select `n` rows

Preliminary: explore variables

To get information about one variable, you can pull out a single variable using “$” and use the following functions:

- `table()`: get a table summarizing counts
- `unique()`: get the unique responses for a variable
- `sort()`: sort the numerically (or alphabetically)
- `hist()`: produce a histogram summary (for a numeric variable)

```r
table(addh$gender) # number of respondents in each category
```

```
female   male 
  1575   1475 
```

Preliminary: ask a few questions

You may want to answer a few questions about your data before analyzing it. Hereare a few things you could answer:

- What's the median income of this sample? What's the mean age?
- On average, do the young adults surveyed think money, no cheating, or love is more important in a relationship?
- What are the answer choices for `debt`?

Exercise: In your buddy groups, manually write down R code to answer one (or more) of these questions. Can you think of other questions?

What’s the median income of this sample? What’s the mean age?

```r
median(addh$income)
```

```
[1] 13000
```

```r
mean(addh$age)
```

```
[1] 21.86984
```

On average, do the young adults surveyed think money, no cheating, or love is more important in a relationship?

```r
mean(addh$money)
```

```
[1] 6.368852
```

```r
mean(addh$nocheating)
```

```
[1] 9.736393
```

```r
mean(addh$love)
```

```
[1] 9.681967
```

```r
# using dplyr
summarize(addh,
          mean_money = mean(money),
          mean_nocheating = mean(nocheating),
          mean_love = mean(love))
```

```
# A tibble: 1 × 3
  mean_money mean_nocheating mean_love
       <dbl>           <dbl>     <dbl>
1       6.37            9.74      9.68
```

What are the answer choices for `debt`?

```r
unique(addh$debt)
```

```
[1] "nodebt"  "yesdebt"
```

Data Manipulation with Dplyr

Outline of dplyr review

- dplyr “verbs”:
  - select
  - filter
  - arrange
  - mutate
  - group_by
  - summarise
- rename
- chaining together verbs with pipe operator %>%
  - “base R” now has a native pipe |> that also works!

`select`: a way to extract columns

Can be used in combination with other dplyr verbs such as: contains, starts_with, and ends_with

Example: extract any column with the word “pay”: `paypercent` and `logpaypercent`

```r
paycold <- select(addh, contains("pay"))
head(paycold, 3)
```

```
# A tibble: 3 × 2
  paypercent logpaypercent
       <dbl>         <dbl>
1       64.2          4.16
2       90.8          4.51
3       19.4          2.97
```

filter: a way to extract rows

Can be used in combination with logical statements

Example: extract observations with an income <20,000 year but no debt

```r
nodebtd <- filter(addh, debt == "nodebt" &
                             income < 20000)
nrow(nodebtd)
```

```
[1] 1275
```

`filter`: a way to extract *missing* rows

Can be used in combination with logical statements for missing data

Example: remove observations where income is missing

```r
nomissinginc <- filter(addh, !is.na(income)) # only keep obs that are NOT (

nomissinginc <- drop_na(addh, income) # alternate function from tidyr

nrow(nomissinginc)
```

```
[1] 3050
```

arrange: a way to arrange rows by the order of their column values

Example: find the two observations who think money is extremely important for a relationship (10 on money variable) but who pay for the fewest percentage of dates (paypercent)

```r
addh %>%
  filter(money == 10) %>%
  arrange(paypercent) %>%
  head(2)
```

```
# A tibble: 2 × 11
     id   age gender income logincome debt     love nocheating money
paypercent
  <dbl> <dbl> <chr>   <dbl>     <dbl> <chr>   <dbl>      <dbl> <dbl>
<dbl>
1  3058    21 female   1044      6.95 yesdebt    10         10    10
18.7
2   506    24 female   1200      7.09 nodebt     10         10    10         19
# ℹ 1 more variable: logpaypercent <dbl>
```

`mutate`: a way to add new variables

Example: add a variable with the average rating for nocheating, money, and love's importance for a relationship (sum divided by 3) and another variable that logs that rating

```r
addhd <- mutate(addh,
               rateavg = (love + money + nocheating)/3,
               rateavglog = log(rateavg))

# look at the first 3 rows and some columns
addhd %>%
  select(love, money, nocheating, rateavg, rateavglog) %>%
  head(1)
```

```
# A tibble: 1 × 5
   love money nocheating rateavg rateavglog
  <dbl> <dbl>      <dbl>   <dbl>      <dbl>
1    10     7         10       9       2.20
```

`mutate`: storing new variables

Be aware that your choice of names affects whether a new object or column is created with mutate!

- By using the same column name or same object name, you overwrite the original object or column.
- Unlike Stata, the default in R is not to change the underlying data, so you must intentionally save it with `<-`.

Example: multiple ways to store a new variable that logs the rating of love

```r
# New column, new dataframe
addhnew <- mutate(addh,
                  loglove = log(love))

# New column, same dataframe
addh <- mutate(addh,
                loglove = log(love))

# Overwrite old column, same dataframe
addh <- mutate(addh,
               love = log(love))

# Overwrite old column, new dataframe
addhnew <- mutate(addh,
                  love = log(love))
```

`group_by` and `summarise`: a way to collapse data by category and generate summary statistics

Example:

- Group by gender
- Generate a summary statistic of not cheating's importance on that grouped data

```r
gender_group <- group_by(addh, gender)
summarise(gender_group,
          meannocheat = mean(nocheating))
```

```
# A tibble: 2 × 2
  gender meannocheat
  <chr>        <dbl>
1 female        9.85
2 male          9.61
```

`group_by` and `summarise`: additional verbs...

- n(): count the elements in a group
- n_distinct(): count the distinct elements in a group
- first: list the first element (would usually use in combo with arrange)
- last: list the lest element (same as above)

Example: Find

1. the number of females and males by debt status
2. the percentage in each debt x gender category as a fraction of all observations
3. the number of distinct ratings of love’s importance in each of these debt x gender categories

Example: Find

1. the number of females and males by debt status
2. the percentage in each debt x gender category as a fraction of all observations
3. the number of distinct ratings of love's importance in each of these debt x gender categories

```r
genderdebt <- group_by(addh, gender, debt)
summarise(genderdebt,
          count = n(),
          percent = n()/nrow(addh),
          distinctlove = n_distinct(love))
```

```
# A tibble: 4 × 5
# Groups:   gender [2]
  gender debt    count percent distinctlove
  <chr>  <chr>   <int>   <dbl>        <int>
1 female nodebt    828   0.271            8
2 female yesdebt   747   0.245            7
3 male   nodebt    907   0.297           10
4 male   yesdebt   568   0.186            8
```

Rename

- Rename: you can use rename() as a function to modify names of columns.
- You can rename numerous columns by using c() to produce a 1-D array to pass to the replace position (more details later)

```r
# let's use one of the built in R datasets, mtcars
head(mtcars, 1)
```

```
          mpg cyl disp  hp drat   wt  qsec vs am gear carb
Mazda RX4  21   6  160 110  3.9 2.62 16.46  0  1    4    4
```

```r
# what if we want the names to be more informative
mtcars2 <- rename(mtcars,
                  c("displacement" = "disp",
                    "milespergal" = "mpg"))
```

```r
head(mtcars2, 1)
```

```
          milespergal cyl displacement  hp drat   wt  qsec vs am gear carb
Mazda RX4          21   6          160 110  3.9 2.62 16.46  0  1    4    4
```

Combining multiple verbs with piping

- You’ll notice that the example on the previous slides combines multiple actions
- Pipes provide a way to chain together multiple verbs in a specified order
  - Pipes (%>%) comes from the package with two aims: to decrease development time and to improve readability and maintainability of code
  - This operator %>% allow you to pipe a value forward into an expression or function call; something along the lines of , rather than . It might be helpful to think of this as …then…

Piping Functional Sequence

The basic (pseudo) usage of the pipe operator goes something like this:

```
awesome_data <-
  raw_interesting_data %>%
  transform(somehow) %>%
  filter(the_good_parts) %>%
  finalize
```

This takes an input, an output, and a sequence transformations. That's suprisingly close to the definition of a function, so `magrittr` is really just a convenient way of of defining and applying a function.

Example of combining multiple verbs with piping

Example:

- Group the data by gender and debt status
- Find the average rating of love, no cheating, and money’s importance for a relationship in each group
- Arrange the groups by their rating of money’s importance to a relationship from the highest to rating to the lowest rating

Implementing in R with pipes

```r
addh %>%
  group_by(gender, debt) %>%
  summarise(nocheatavg = mean(nocheating),
            loveavg = mean(love),
            moneyavg = mean(money)) %>%
  arrange(desc(moneyavg)) %>%
  rename(c("No cheating average" = "nocheatavg",
           "Love average" = "loveavg",
           "Money average" = "moneyavg"))
```

```
# A tibble: 4 × 5
# Groups:   gender [2]
  gender debt    `No cheating average` `Love average` `Money average`
  <chr>  <chr>                   <dbl>          <dbl>           <dbl>
1 male   nodebt                   9.60           2.24            6.40
2 female nodebt                   9.84           2.26            6.40
3 female yesdebt                  9.87           2.28            6.38
4 male   yesdebt                  9.62           2.25            6.25
```

Summing up

In this lecture, we've reviewed:

- Practices for organizing research code
- Basics of data exploration and working with variables
- dplyr and pipes as a tool for data manipulation

Pair exercises!

Exercise:

1. We will randomize you into new pairs!
2. Move to the “Pair exercises” section of your `day1_morning_codealong.qmd` document for the exercises.
3. Practice working with the Add Health data.