Session 1

Data Wrangling in R

Youth Impact R Workshop Series

Zezhen Wu

Disclaimer

Part of this presentation is borrowed from the public GitHub repository for the NYU Global TIES R workshop coauthored by Patrick Anker and Zezhen Wu.

Agenda

By the end of this workshop, you should be able to understand the following code:

```
library(tidyverse)

dat <- read_csv(here::here("data/RCT.csv"))

dat %>%
  filter(age <= 10) %>%
  group_by(sch_id) %>%
  summarize(avg_scores = mean(std_scores)) %>%
  mutate(transformed_scores = as.numeric(scale(avg_scores))) %>%
  select(-avg_scores)
```

We'll be covering

- Basic R syntax
- Data loading
- Data inspection
- Data manipulation

Stata vs. R (*ref*)

Stata	R
Stata is a proprietary, licensed software and can be expensive	R is open-source and free
Only one dataset can be in memory at a time	Multiple datasets can be manipulated and used simultaneously
Limited to built-in and user-created .ado files for extending functionality	A vast ecosystem of packages available through CRAN and GitHub
Graphic capabilities, while adequate, are not as flexible or customizable	Advanced graphical packages (e.g., ggplot2) offer high-quality and customizable visualizations
Stata's community, while active, is smaller and less diverse	Larger and more active user community leading to more resources, solutions, and online help
Limited advanced programming capabilities	Offers advanced programming constructs like functions, lists, and apply functions for better automation

Learning tools

- R for Data Science by Grolemund and Wickham
- ChatGPT (especially GPT4)
- Stackoverflow
- There is usually an R book for almost everything (see here).

Type of R files

R Projects (.Rproj)

- Organize work in a structured manner with relative path.
- Provide a dedicated workspace for each project.

R Script (.R)

- Plain text files containing R code.
- Executed directly or sourced within other scripts or documents.

R Markdown (.Rmd)

- Combine R code, text, and output.
- Create dynamic documents.

We will talk about data workflow management in a future workshop.

Let's try to create these files by hand!

Loading your data

Understanding R syntax

R is built around functions, bits of atomic code that accept inputs and return an output.

A function looks like this:

```
function_name(argument_1, argument_2, parameter = TRUE)
```

Functions can be bundled together and shared using packages. To access a function from a specific package, we use the `::` operator.

```
package.name::function_name(argument_1, argument_2, parameter = TRUE)
```

Understanding R syntax

It may be cumbersome to write `::` each time you use a function. Therefore, R allows you to load those packages in your current session using the `library()` function so you can just refer to the function name without `::`.

```
library(package.name)

# Refers to `function_name` in `package.name`
function_name(argument_1, argument_2, parameter = TRUE)
```

In Stata, this would be similar to:

```
// function_name argument_1 argument_2, parameter 1
```

For instance,

```
# Stata code
gen new_var = old_var * 2
```

```
# R code
df <- dplyr::mutate(df, new_var = old_var * 2)
```

Tools for loading data

While there are lots of options, I recommend loading data using the following functions:

```
readr::read_csv("data.csv")
haven::read_dta("data.dta")
openxlsx::read.xlsx("data.xlsx")
```

readr and haven are both part of the "*tidyverse*", a bundle of packages that contain the most essential functions for data wrangling.

openxlsx is a standalone package that deals with reading and writing Excel files.

In Stata, the equivalent commands are:

```
import delimited using "data.csv"
use "data.dta"
import excel using "data.xlsx", firstrow
```

Tools for loading data

Some packages come with the installation of R, some don't. This is because R is an open-source software, and anyone in the world can contribute their packages to the R community.

To use these packages, you need to install them first by running `install.packages("package.name")`:

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

Note: Only run these commands once. Once they're installed, you don't need to run that command again since the package will be on your computer.

Finding help on packages

R packages are typically published on CRAN (The Comprehensive R Archive Network).

Developer versions of R packages are typically published on GitHub (for instance, `readr`).

When you encounter a function that you don't know or understand for some reason, consult the R help system!

```
# search help doc for a package
?readr

# search help doc for a function in a package
?readr::read_csv

# If you're not certain what a function name is or you're looking for a
??readr::read_table
```

Let's load our data!

The simulated RCT dataset is located in the `data` directory (folder) at the root of this project. I'll use the here package to create paths that are *relative to the project root*.

I'll attach the packages we need for now:

```
library(readr)
library(here)
```

```
## here() starts at /Users/michaelfive/Desktop/R Directory/R_Workshop
```

Note how here tells us where it thinks the project root is! It recursively searches upward for a directory with a `DESCRIPTION` or `.Rproj` file.

here is very handy utility for reproducibility since it *localizes* paths to the project -- and not expanding to the full path on *your* computer.

Let's load our data!

Time to load the data:

```
dat <- read_csv(here("data/RCT.csv"))
```

```
##
```

```
Rows: 1000 Columns: 6
##  [36mi [39m Rendering  ]8;;file:///Users/michaelfive/Desktop/R Directory/R_Wo
##
```

```
##  [36mi [39m Rendering  ]8;;file:///Users/michaelfive/Desktop/R Directory/R_Wo
```

```
── Column specification ────────────────────────────────────────────────────────
##  [36mi [39m Rendering  ]8;;file:///Users/michaelfive/Desktop/R Directory/R_Wo
```

```
Delimiter: ","
## dbl (6): sch_id, st_id, treated, age, female, std_scores
##
##  [36mi [39m Rendering  ]8;;file:///Users/michaelfive/Desktop/R Directory/R_Wo
##
```

```
##
```

Two things to note

R uses the `<-` operator to *assign* values to symbols (variables).

```
x <- (1+1) * 2^3 / 4 - 2
```

In our data's case, the results of reading the CSV file are assigned to the `dat` symbol.

In STATA, the closest action is to generate a variable called `x` in the current dataset. But what the R code does is to create x as a vector of 1 instead of a data frame.

```
# Stata code
gen x = (1+1) * 2^3 / 4 - 2
```

Unlike Stata, you can have multiple data frames in one R session.

```
dat1 <- dat
dat2 <- dat
```

This is very helpful in cross-checking and merging datasets, as well as interacting with various R objects (e.g., vectors, matrices, lists, data frames, etc.).

Looking at our data

After loading our data, it's useful to high-level view of its content to make sure that it loaded correctly.

We'll use the `str()` and `print()` functions here, both part of base R.

Looking at our data

```
str(dat)
```

```
## spc_tbl_ [1,000 × 6] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ sch_id    : num [1:1000] 1 1 1 1 1 1 1 1 1 1 ...
##  $ st_id     : num [1:1000] 1 2 3 4 5 6 7 8 9 10 ...
##  $ treated   : num [1:1000] 0 1 0 1 1 1 0 1 1 1 ...
##  $ age       : num [1:1000] 8 10 12 7 11 11 9 9 9 9 ...
##  $ female    : num [1:1000] 0 0 0 0 0 0 0 0 1 0 ...
##  $ std_scores: num [1:1000] -0.159 -0.578 -0.807 0.163 -0.621 ...
##  - attr(*, "problems")=<externalptr>
```

```
class(dat)
```

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

Looking at our data

The *class* of this `dat` object is a `data.frame`. `data.frame`s are R's representation of tabular data. `dat` is also a `tbl_df` ("tibble"), which is the Tidyverse's version of a `data.frame` that is a little bit more stringent (i.e. safe) than base R's version, and it has a prettier `print()` method:

```
print(dat)
```

```
## # A tibble: 1,000 × 6
##    sch_id st_id treated   age female std_scores
##     <dbl> <dbl>   <dbl> <dbl>  <dbl>      <dbl>
##  1      1     1       0     8      0     -0.159
##  2      1     2       1    10      0     -0.578
##  3      1     3       0    12      0     -0.807
##  4      1     4       1     7      0      0.163
##  5      1     5       1    11      0     -0.621
##  6      1     6       1    11      0     -0.643
##  7      1     7       0     9      0     -0.337
##  8      1     8       1     9      0     -0.345
##  9      1     9       1     9      1      0.160
## 10      1    10       1     9      0     -0.248
## # ℹ 990 more rows
```

Looking at our data

If you'd like to see *all* of your data in a pretty Excel-like spreadsheet, use the `View()` function which has a great display in RStudio.

```
View(dat)
```

Note: Or you can click on the data frame in the Environment tab. Or hold command/ctrl and click `dat` on the R script.

Data types in R

You may have noticed a bunch of three-letter abbreviations next to the variable names in `print(dat)` and the variable content in `str(dat)`. These are the *type signatures* for each variable:

- Double (`dbl` / `num`): All real numbers, like 50, 23.193, or 1.24×10^{-8}
- Character (`chr`): Text/strings, written with single- or double-quotes (e.g. `"a string"`)

There are a couple more that you will encounter regularly:

- Integer (`int`): Integers and counting numbers. No decimals and scientific notation here!
- Logical (`lgl`): Boolean data, written as `TRUE` or `FALSE`.
- Factor (`fct`): Factors, R's representation of categorical data. They look like character data with `print()`, but under the hood they are integers. These should only exist in cleaned, ready-for-analysis data!

Quiz

How do I create a variable `x` that refers to the sentence "Hello, world!" ?

Use 1, 2, 3, or 4 to refer to the code line number.

```
x "Hello, world!"
"x" <- Hello, world!
x <- "Hello, world!"
x <- Hello, world!
```

Quiz

How do I create a variable `x` that refers to the sentence "Hello, world!" ?

Use 1, 2, 3, or 4 to refer to the code line number.

```
x "Hello, world!"
"x" <- Hello, world!
x <- "Hello, world!"
x <- Hello, world!
```

Working with your data

Working with your data

What we'll be discussing today is basic *data wrangling*.

> Data wrangling (n.): The process of transforming data from one raw form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics.
>
> —*Wikipedia*

Basic Data Wrangling

Most data wrangling comes down to five basic operations:

- Selecting: Selecting or deselcting variables
- Filtering: Filtering by row
- Arranging: Reordering by variables
- Summarizing: Aggregating data for summary statistics
- Mutating: Creating, editing, or deleting variables

You can also Group these operations by variables. For example, finding the average math score in each class would be achieved by *grouping* by class and then *summarizing* by taking the mean.

Basic Data Wrangling

In this workshop, we recommend using the dplyr package in tidyverse, but we can also achieve the same thing in base R:

Operation	Tidyverse	Base R
Selecting	`select(dat, treated)`	`dat[,"treated"]`
Filtering	`filter(dat, treated == 1)`	`dat[dat$treated == 1", ]`
Arranging	`arrange(dat, std_scores)`	`dat[order(dat$std_scores), ]`
Summarizing	`summarize(dat, avg_scores = mean(std_scores))`	`data.frame(avg_scores = mean(dat$std_scores))`
Mutating	`mutate(dat, failed = std_scores < 0)`	`dat$failed <- dat$std_scores < 0`

It's apparent that tidyverse has a much more consistent and readable syntax than base R.

The equivalent commands in Stata are:

```
keep treated
keep if treated == 1
sort std_scores
collapse (mean) avg_scores = std_scores
gen failed = std_scores < 0
```

Basic Data Wrangling

All of these tidyverse functions are provided by the dplyr package.

```
library(dplyr)
```

Selecting

Selects or deselects one or multiple variables.

```
select(dat, treated)
```

```
## # A tibble: 1 × 1
##   treated
##     <dbl>
## 1       0
```

```
select(dat, -treated)
```

```
## # A tibble: 1 × 5
##   sch_id st_id   age female std_scores
##    <dbl> <dbl> <dbl>  <dbl>      <dbl>
## 1      1     1     8      0     -0.159
```

```
select(dat, sch_id:female)
```

```
## # A tibble: 1 × 5
##   sch_id st_id treated   age female
##    <dbl> <dbl>   <dbl> <dbl>  <dbl>
## 1      1     1       0     8      0
```

Filtering

Looks for rows with a provided comparison, usually with `<`, `>`, `<=`, `>=`, or `==` (note: two equals!).

```
filter(dat, treated == 1)
```

```
## # A tibble: 499 × 6
##    sch_id st_id treated   age female std_scores
##     <dbl> <dbl>   <dbl> <dbl>  <dbl>      <dbl>
##  1      1     2       1    10      0     -0.578
##  2      1     4       1     7      0      0.163
##  3      1     5       1    11      0     -0.621
##  4      1     6       1    11      0     -0.643
##  5      1     8       1     9      0     -0.345
##  6      1     9       1     9      1      0.160
##  7      1    10       1     9      0     -0.248
##  8      1    11       1     9      0     -0.365
##  9      1    13       1     9      1      0.220
## 10      1    15       1    11      0     -0.771
## # ℹ 489 more rows
```

Homework: Figure out how to choose row-wise those with age greater than the average age.

Arranging

Sorts a dataset by a `variable` in *ascending* order with `arrange(dat, variable)`.

```
arrange(dat, std_scores)
```

```
## # A tibble: 6 × 6
##   sch_id st_id treated   age female std_scores
##    <dbl> <dbl>   <dbl> <dbl>  <dbl>      <dbl>
## 1      5   486       1    15      0      -1.40
## 2      2   178       0    14      0      -1.36
## 3      3   227       1    14      0      -1.33
## 4      8   771       0    14      0      -1.27
## 5      1    62       0    14      0      -1.22
## 6     10   973       1    13      0      -1.19
```

Homework: Figure out how to order by *descending* order.

Summarizing

Calculate some summary function on variables, like `mean()`, `median()`, or `sd()`:

```
summarize(
  dat,
  avg_score = mean(std_scores),
  sd_score = sd(std_scores)
)
```

```
## # A tibble: 1 × 2
##   avg_score sd_score
##       <dbl>    <dbl>
## 1    -0.352    0.357
```

Homework: Is there a difference between summarize() and summarise()?

Mutating

Create, edit, or delete variables in a dataset.

```
mutate(
  dat,
  # Changes condition to an index variable with character values
  condition = ifelse(treated == 1, "Treatment", "Control")
)
```

```
## # A tibble: 2 × 7
##   sch_id st_id treated   age female std_scores condition
##    <dbl> <dbl>   <dbl> <dbl>  <dbl>      <dbl> <chr>
## 1      1     1       0     8      0     -0.159 Control
## 2      1     2       1    10      0     -0.578 Treatment
```

Homework: Create a variable in **`dat`** called age_gt_10, which takes the value of 1 when age > 10, otherwise 0.

Tidyverse Caution!

The tidyverse (mainly dplyr, but a couple other packages) refers to variables in datasets with "naked" symbols.[1]

[1]: In computer science lingo, these are technically called *quoted symbols*, but that can be confusing.

Tidyverse Caution!

But *outside* of the tidyverse functions, these variables are not defined.

```
mean(std_scores)
```

```
## Error in mean(std_scores): object 'std_scores' not found
```

Tidyverse Caution!

To access dataset variables outside of tidyverse functions, you must use the **$** ("extract") operator in conjunction with the symbol for the parent dataset.

Just keep this in mind!

Combining Operations

These "verbs" (as dplyr likes to call them) are composable meaning you can chain these commands together, allowing you to create complex data transformations or summaries all in one step.

```
summarize(
  filter(dat, treated == 1),
  avg_score = mean(std_scores)
)
```

```
## # A tibble: 1 × 1
##   avg_score
##       <dbl>
## 1    -0.347
```

This is ugly. Thankfully, there is a better way of chaining these steps together!

Introducing...

The Pipe!

%>%

The pipe `%>%` is an operator provided by magrittr[1] that makes writing nested functions *much* easier.

The shortcut is `command + shift + M` on Mac, or `ctrl + shift + M` on Windows.

[1]: Starting from R 4.1.0, there is the native pipe |> in R, which does the same thing without loading `tidyverse`.

The Pipe

Instead of writing

```
h(g(f(x)))
```

you can write

```
x %>%
  f() %>%
  g() %>%
  h()
```

The pipe works by inserting the result of the left hand side into the first argument of the right hand side.

Combining Operations

Revisiting our first chained example,

```
summarize(
  filter(dat, treated == 1),
  avg_scores = mean(std_scores)
)
```

we can now use

```
dat %>%
  filter(treated == 1) %>%
  summarize(avg_scores = mean(std_scores))
```

```
## # A tibble: 1 × 1
##   avg_scores
##        <dbl>
## 1     -0.347
```

Grouping

Now let's add grouping into the mix, which you can easily do with `group_by(variable)`:

```
dat %>%
  group_by(treated) %>%
  summarize(avg_scores = mean(std_scores))
```

```
## # A tibble: 2 × 2
##   treated avg_scores
##     <dbl>      <dbl>
## 1       0     -0.357
## 2       1     -0.347
```

Grouping

You can group by multiple variables as well. Just add more variables to `group_by()`:

```
dat %>%
  group_by(treated, female) %>%
  summarize(avg_scores = mean(std_scores))
```

```
## # A tibble: 4 × 3
## # Groups:   treated [2]
##   treated female avg_scores
##     <dbl>  <dbl>      <dbl>
## 1       0      0    -0.504
## 2       0      1     0.0169
## 3       1      0    -0.473
## 4       1      1    -0.0218
```

Recap

How would I find the median scores by treatment and gender?

```
# 1.
dat %>%
  filter(treated, female) %>%
  summarize(
    med_scores = median(std_scores)
  )

# 2.
dat %>%
  summarize(
    med_scores = median(std_scores)
  ) %>%
  group_by(treated, female)

# 3.
dat %>%
  group_by(treated, female) %>%
  summarize(
    med_scores = median(std_scores)
  )

# 4.
dat %>%
  summarize(
    med_scores = median(std_scores)
  ) %>%
  filter(treated, female)
```

Recap

How would I find the median scores by treatment and gender?

```
# 1.
dat %>%
  filter(treated, female) %>%
  summarize(
    med_scores = median(std_scores)
  )

# 2.
dat %>%
  summarize(
    med_scores = median(std_scores)
  ) %>%
  group_by(treated, female)

# 3.
dat %>%
  group_by(treated, female) %>%
  summarize(
    med_scores = median(std_scores)
  )

# 4.
dat %>%
  summarize(
    med_scores = median(std_scores)
  ) %>%
  filter(treated, female)
```

Bonus Point

You can do Tableau-style plots in R using just one command:

```
install.packages("esquisse")
library(esquisse)

# Open Tableau-like dashboard in Rstudio
esquisser()
```

Note: We can talk about summary statistics and visualization in a future workshop.

Exercise

Use the functions you learned to answer the following question:

1. What is the average age for boys and girls?
2. Which `st_id` is the oldest students in school 8?
3. Which age group has the most number of students?
4. What is the average `std_scores` for girls aged 7 or 8 in the treatment group?
5. Create a data.frame with the name `dat_s` with the following rules:

- Deselect `st_id`
- Filter out students younger than 10 years old
- Create a new variable called `transformed_scores` and assign it the value of 99 times the `std_scores` if age >= 10, else assign it the value 101 times the `std_scores`

Consider using the `|` (read: or) operator to combine multiple filter statements into one (e.g. filtering for `this` or `that` would be `filter(dat, this | that)`).