Data Import and Formatting

http://datascience.tntlab.org

Module 4

Today’s Agenda

- Importing text data
- Basic data visualization
- *tidyverse* vs *data.table*
- Data reshaping and type conversion

Basic Text Data Import

- read.csv(), read.delim(), and read.table() are base-R
- read_csv(), read_tsv(), read_delim(), and read_table() are from *readr*
- fread() is from *data.table*

- 99% of the time, for the type of data we usually look at, you'll use read.csv() (or read.delim())
  - read.csv() is faster than read_csv() for files under about 1MB in size
  - read_csv() is faster than read.csv() for files above 1MB in size (potentially 100x+)
  - fread() is faster than both

- Why would the same process take different amounts of time depending on whether you used read.csv(), read_csv(), or fread()?

Basic Text Data Import

- If the format is unusual (i.e., not a CSV or TSV), definitely start with fread(), which is the *data.table* version of read.table() but automatically detects a lot of formatting features
  - Notice the "auto" value for many of its parameters
  - Note that you can **drop** or **select** variables at import, which is super-useful
    - These parameters take numeric or character vectors

- Important commonly used parameters across functions:
  - header: Boolean
  - stringsAsFactors: Boolean
  - skip: integer

- You can use collectors to pre-specify filetypes, but it's usually easier just to recast them as needed if one the import functions can't figure it out

Process for Data Imports

1. Look at the raw format in a text editor first, if you can
   - On a PC, metapad or notepad++ recommended
   - On either, you can try to open it in RStudio directly
2. Look for the patterns you'll need to develop parameters for the functions
3. Figure out which function is most likely to give you what you want (most likely read.csv() or fread())
4. Write initial code for the import
5. Run that code
6. Revise as needed

Tidy Data

- tidyverse is an R framework
  - Used in multiple packages that all agree upon that same basic structure of functions
  - Represents a data philosophy
  - To use the framework, **library(tidyverse)** is easiest
  - Within tidyverse functions
    - Using the full names of variables in dfs is unnecessary
    - Functions generally return the original df, but changed

  - Core tidyverse: *ggplot2, dplyr, tidyr, readr, purrr, tibble*

Versus data.table…

- data.table is also an R framework
  - Also used in multiple packages with the same caveats
  - Also represents a data philosophy

- Why use one or the other?
  - Familiarity
  - data.table is faster for huge data files because it is more efficient
  - tidyverse is more English-language oriented, so it is easier to learn and read
  - tidyverse has several high-popularity packages

- Because objects in R can have multiple classes, you don’t (exactly) need to choose

Exploring Data

- str(), dim(), colnames() (less necessary when using RStudio)
- glimpse()
- head() and tail()

- Both head() and tail() commonly take two parameters:
  - x, a data frame, matrix, or table
  - n, the number of rows

- Therefore, these are very useful to double-check the file imported as you think it did:
  - head(my_df, 1)
  - tail(my_df, 1)

Common Quick Visualization Tools

- hist() for histograms
- plot() for scatterplots
- barplot() for barcharts, but only on the results of table()
- boxplot()

- These are all in base-R
- If you want presentation-quality visualizations, don't use these functions

Reshaping Data

- Interpretation: gather(data, key, value, cols)
  - Within data, for each variable in cols, convert the name of the column as a value in a new variable called key, and convert each value into a second new variable called value – then return remaining variables from df plus the gathered ones

  - Example
    -

participantID	time1	time2	time3
1	1	2	3
2	3	4	3
3	4	5	2

    - gather(df, time, val, 2:4) # in datacamp, they used a minus as shorthand for "all but"
    -

participantID	time	val
1	time1	2
1	time2	2
1	time3	3
2	time1	3
2	time2	4
2	time3	3
3	time1	4
3	time2	5
3	time3	2

And the Reverse

- Interpretation: spread(df, key, value)
  - Within df, convert the value of col1 (key) into column names and put the values in col2 (value) column as values of those new variables – add that to any other columns already in the dataset
  - Example
    -

participantID	time	val
1	time1	2
1	time2	2
1	time3	3
2	time1	3
2	time2	4
2	time3	3
3	time1	4
3	time2	5
3	time3	2

    - spread(df, time, val)
    -

participantID	time1	time2	time3
1	1	2	3
2	3	4	3
3	4	5	2

Other Reshaping and Conversion Functions

- Multiple values stored in a single variable
  - separate(data, col, into, sep) – converts col from data into length(into) variables called into, split by sep
  - unite(data, col, …, sep) – converts all … into a single col, split by sep
- Printing out locations within a vector
  - which()
- Variables stored as wrong type
  - as.character, as.integer, etc.
  - For date conversions: **lubridate**, including ymd(), dmy(), etc. (but be careful about POSIX)
- String manipulations
  - Use **stringr**: str_trim(), str_pad(), str_detect(), str_replace()

Missing Values Handling

- A fact of life in social scientific research

- Usually represented in R using NA
  - Remember is.na and na.rm

- Useful functions
  - any() : Check if anything in this vector is TRUE
  - complete.cases() : Return a Boolean vector listing complete cases
  - na.omit() : Listwise deletion (you should generally not use this)

Closing Notes

- tidy data is not always what you want
  - Sometimes we want untidy data, i.e., dichotomous representations of variables for use in a later analyses
  - The part of the tidy data that you should internalize is *row independence,* which in social science almost always means *independence of observations*, a common statistical assumption
- Finally, a useful cheat sheet (both this week and next):
  - https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf