R Lab 0: Introduction to R

Contents

Why R?

Now Entering An Excel-Free Zone

While Excel and other spreadsheet software are undoubtedly powerful, ubiquitous tools to quickly manipulate/view tabular data, the data manipulation steps one performs in Excel–be it copying and pasting, sorting the table by a column, fixing that small typo, or accidentally causing that huge transcription error–cannot be tracked and thus *cannot be easily corrected or replicated by other researchers*.

Reproducibility issues in Excel are associated with numerous horror stories*. Moreover, the more complicated statistical analyses you'll likely wish to run are generally not possible in Excel.

Instead, we highly recommend using a statistical programming language while documenting all of the data manipulation and statistical analyses performed programmatically (i.e., no manual manipulation or calculations). The Department of Biostatistics and Informatics here at the Colorado School of Public Health emphasizes two languages: SAS and R. We'll use both during BIOS 6611.

SAS is a reliable and trusted statistical analysis platform that is undisputedly the "industry standard" in many fields related to medicine and public health (namely, the pharmaceutical industry, hospitals/commercial analytics).

However, while SAS does some things exceptionally well and provides a very stable platform for analysis, it has been criticized for not taking advantage of newer developments in software design and statistical approaches. Enter R.

* Notorious spreadsheet mistakes:

- Mathieson, SA. "MI5 makes 1,061 bugging errors." The Guardian, 1 July 2011.
- Baker, Dean. "How Much Unemployment Did Reinhart and Rogoff's Arithmetic Mistake Cause?" The Guardian, 16 Apr. 2013.
- Ziemann, Mark, Yotam Eren, and Assam El-Osta. "Gene name errors are widespread in the scientific literature." Genome biology 17.1 (2016): 177.

Key Features of R (With A Reluctant Comparison to SAS)

"Reluctant" because each statistical package has its own benefits and disadvantages. We also encourage use of other statistical / scientific computing languages–SPSS, STATA, Matlab, Python–with curiosity or need.

- **Free and open-source.** In recent years SAS has released free editions like SAS University Edition for students/non-commercial uses. However, as of summer 2017, SAS's basic analysis package for organizations costs $9,380 per user annually. R is absolutely free to download and use.

- **High customizability and flexibility.** After learning the basics of R, it's relatively easy to create custom functions and data processing pipelines (compare to SAS, in which different programming syntax is used for procedures, IML, and macros). For folks with a background in programming, we should also note that R is a very dynamic programming language, for both better (no need to compile, interesting metaprogramming features) and worse (can sacrifice speed, error-checking for flexibility).
- **Access to new techniques and diverse software.** Due to R's accessibility and flexibility, it's extremely common for academic researchers to distribute their new statistical techniques in freely-available R "packages." There are also many efforts to increase accessibility to traditionally paywalled software or computationally challenging statistical modeling processes (e.g., latent variable modeling, Bayesian hierachial models).
- **Reproducibility Tools.** In response to the issues associated with using Excel, something emphasized by the R development community is creating tools to perform analyses in a reproducible fashion. This includes powerful tools like `Rmarkdown` and `knitr`.
- **Graphics.** R is famous for its highly customizable plotting tools: both the default plots in "base R", as well as `ggplot2` (create complex visualizations using simple syntax) and `shiny` (interactive animations / webapps).

However, there's an obvious flip-side here: although SAS is slower to release the state-of-the-art, their releases are consistently well-documented, tested, and trusted. Be careful when using experimental R packages that aren't well-known in the community.

Getting Started

With all these possible advantages of R being said, R may come with a steep initial learning curve (especially for users with no prior programming experience) in exchange for its diverse features and flexibility. We'll go over some "getting started" basics in this document.

Installing R

Follow the instructions associated with your operating system (Windows, Mac, Linux) under the "download and install" header R at cran.rstudio.com.

"CRAN" stands for the Comprehensive R Archive Network and hosts many of the free packages and documentation contributed by the community.

Installation With RStudio IDE

We *highly* recommend installing RStudio, an Integrated Development Environment (IDE) that makes it much easier to learn R and perform analyses reproducibly.

Rstudio allows you to write and debug R programs, keep track of variables in your environment, read documentation, view graphics, and create reproducible analyses in a single window. The basic version is also open-source and free for individuals.

Download the RStudio Desktop "Open Source Edition" at rstudio.com/products/rstudio, and follow the installation instructions.

Foundations of R

Assignment and Basic Data Types

To tell R to store something, we use the following syntax.

```
# assign value of 7 to variable named "days_per_week"
days_per_week <- 7

# display value of variable
days_per_week
```

```
## [1] 7
```

```
# the variable name is case sensitive:
# R doesn't have this stored in memory
DAYS_PER_WEEK
```

```
## Error in eval(expr, envir, enclos): object 'DAYS_PER_WEEK' not found
```

You can also use `=` instead of `<-` to assign variables. While most style guides recommend the latter, it's more important to be consistent.

Also, you can store two "types" of numeric variables–**integer** or **double**. The latter is used by default. You can also store **logical** values (`TRUE` or `FALSE`). Lastly, for storing text there is the **character** type ("three point one four") or **factors** (categorical variables with some predefined possible values – like how a variable named `voter_affiliation` might have a limited number of values, like "Democrat", "Republican", "Independent").

Vectors

To store multiple values in a given variable name, you can use the concatenate function `c()` to piece different values together.

```
# making a vector
scary_stuff <- c("lions", "tigers", "bears", "oh my")
```

```
# displaying what's stored in vectors
scary_stuff
```

```
## [1] "lions"  "tigers" "bears"  "oh my"
```

```
# what type of data is it?
typeof(scary_stuff)
```

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

```
# more vector examples
fibonacci_series <- c(1, 1, 2, 3, 5, 8)

# what type of data is it?
typeof(fibonacci_series)
```

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

A vector cannot store heterogeneous data types. Check out what happens here:

```
# how does R deal with mixed data in a vector?
test <- c(T, F, 123, "sandwich")

# print vector created with mixed data types
test
```

```
## [1] "TRUE"     "FALSE"    "123"
## [4] "sandwich"
```

```
# print type of vector
typeof(test)
```

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

Vectors can only contain one data type! Because R doesn't know how to convert "sandwich" into a number, it *coerces* the whole vector to be a "character" type to avoid losing information.

Coercion is a feature in R that is sometimes very convenient, but is sometimes very scary–note that we didn't get a warning or error from trying to put in logical, double, and character types together.

Matrices

A two-dimensional vector is known as a "matrix".

```
# wrapping "scary_stuff" vector into a matrix
scary_matrix <- matrix(scary_stuff, c(2, 2))
```

As a matrix is a two-dimensional vector, can still only have one data type here (i.e., coercion happens).

```
# making a matrix that is two rows by four columns long
matrix( c(1, 2, 3, 4, 5, 6, "seven", "eight"), c(2, 4))
```

```
##      [,1] [,2] [,3] [,4]
## [1,] "1"  "3"  "5"  "seven"
## [2,] "2"  "4"  "6"  "eight"
```

Lists and Data Frames

However, there's situations where we might want to collect different data types into a single variable. For this, we need a **list**.

```
# an example of a list
list(T, F, 123, "sandwich")
```

```
## [[1]]
## [1] TRUE
##
## [[2]]
## [1] FALSE
##
## [[3]]
## [1] 123
##
## [[4]]
## [1] "sandwich"
```

This output looks different from the vector, and that only sandwich has quotation marks around it. (Only the fourth item in the list is a character.)

Most of the time, we'll be working with **data frames**. A data frame is a two-dimensional list. Within a data frame, each column is like a vector that will all be the same type (i.e., coercion happens within a column), however across a row you can have multiple data types (i.e., each row is like a list).

```r
# make a data frame
# note that T, F are short for TRUE, FALSE
fibonacci_info <- data.frame(
  numbers = fibonacci_series,
  number_is_even = c(F, F, T, F, F, T)
)

# print the data frame
fibonacci_info
```

```
##   numbers number_is_even
## 1       1          FALSE
## 2       1          FALSE
## 3       2           TRUE
## 4       3          FALSE
## 5       5          FALSE
## 6       8           TRUE
```

```r
# check the type of data in the first column
typeof(fibonacci_info$numbers)
```

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

```r
# how about the second column
typeof(fibonacci_info$number_is_even)
```

```
## [1] "logical"
```

Functions

We've implicitly been using "functions" the whole time. Simply put, a function is something that takes in some user input, then outputs something. The desired input is specified by "arguments".

For example, `typeof()` is a function that has a single argument, `x` (any R object of interest). To see this, type `?typeof` in your R console to pull up the documentation for that function.

Then type `?mean` to see the arguments for the arithmetic mean function.

```r
# a clearer example of a function
# x is argument for mean (try inputting a numeric vector)
mean(x = fibonacci_series)
```

```
## [1] 3.333333
```

How about using `quantile()` to save the 75th quantile of a vector? `x` and `probs` are the arguments we'll used for this.

```
# use quantile function, save to "seventyfifth" variable
seventyfifth <- quantile(x = fibonacci_series, probs = 0.75)

# display variable
seventyfifth
```

```
## 75%
## 4.5
```

Why These Building Blocks Are Important

While learning about data types and structures isn't terribly exciting, they form the fundamental basis underlying all R programming and data analysis in R.

We'll soon see how these basic building blocks are used for data manipulation and analysis.

Self-Study Resources

While you'll practice R throughout this course, here are a few additional resources for R language practice (presented by increasing difficulty).

- **CodeSchool** and **DataCamp.** The websites tryr.coodeschool.com and datacamp.com/courses guide you through R basics (data structures, expressions) interactively in an internet browser.
- **Swirl.** Swirl is a R package full of interactive courses that you can use to learn R within the R environment itself. Follow the instructions at swirlstats.com/students.html to install it and go through some courses, which range from R basics to more advanced techniques (regression, inference, and beyond).
- **Quick-R** and **Cookbook for R**. The tutorials at statmethods.net/r-tutorial and cookbook-r.com are more advanced and begin emphasizing data cleaning and statistical applications of R.
- **R for Data Science.** This textbook, freely available online at r4ds.had.co.nz, is a seminal text for folks who have more programming background or are more comfortable in R. It introduces more advanced modeling problems, as well as advanced tools like the `dplyr` package that are heavily used in industry / by practicing R programmers.