Introduction to the R Statistical Computing Environment & RStudio

Measurement, Scaling, and Dimensional Analysis
2019 ICPSR Summer Program
Prof. Adam M. Enders

Objectives

- Learn what R and RStudio are and how they work
- Explore how to complete basic, yet important, tasks in RStudio:
  - Importing and exporting data
  - Data management and manipulation
  - Basic functions
- Consider some basic principles of computer programming (coding)
- Learn how to compute descriptive statistics

What is R?

- R is a language and environment for statistical computing and graphics
- The S language and environment developed at Bell Labs is the direct precursor
  - Really, R is freeware implementation of S, which was commercially implemented in S-PLUS
- Referred to as an "environment" because it's comprehensive
  - Can manage and store data, manipulate data, analyze and graph data, etc.

Why Use R?

- There are a number of programs out there for statistics and data analysis
  - Most common in social sciences: SPSS, Stata, MPlus
- R has many advantages over these programs:
  - It's free!
  - It's flexible: can program your own functions, handle large datasets
  - Lots of public and private entities are beginning to use it
- Some disadvantages:
  - Not quite as easy to interact with as statistics programs built exclusively for data analysis
  - Relatedly, no general user interface (GUI), making getting started a bit harder
- On average, R is worth the trouble you might experience in the very beginning – this is a transferrable skill

What is RStudio?

- A free and open-source integrated development environment (IDE) for R
- What does that mean?
  - Basically, RStudio provides a basic GUI for R that makes getting start with R a bit easier
  - Rather than doing everything from scripts (more on that in a minute) or direct submission to R, this provides some point-and-click-style tools
- Generally speaking, RStudio is set up with four windows or "panes"
  1. Script: this is where you write code to be submitted to R or saved for later
  2. Console: this is where commands are submitted and (non-graphical) output appears
  3. Environment/History: can see loaded datasets and a history of commands issued
  4. Files/Plots/Packages/Help: can see graphical output, packages, and help via documentation

Interacting with RStudio

- Though you can submit commands directly to the console, we will ALWAYS use scripts
- A script is a file where code – commands for carrying out various analyses, calculating particular quantities, or constructing graphs – is written and stored
- I will provide you with the scripts (and data) necessary to carry out (just about) all examples from class
  - This means you will have working code and data necessary to reproduce course material
- When you submit homework involving computing, you must provide two things:
  1. Your own carefully constructed and annotated R script
  2. A written document (presumably using a popular word processor like Microsoft Word) that includes key R output (e.g., graphs, tables) and substantive interpretations of that output

Good Coding Practices

- Congratulations: you're now all computer programmers (of sorts)!
- There are some practices that you should follow in order to keep your life and my life easier
- First, keep lines of code short (no more than about 70 characters)
  - This enhances our ability to quickly and accurately decipher what we've done, and will help troubleshoot errors more quickly
- Second, use lots of comments
  - One of the most important reasons for using R script files is to retain the commands for later use
  - However, it is very easy to forget the details of the analysis over even relatively short time periods

Good Coding Practices

- Beyond helping the researcher, organized R scripts (computing code of any sort) is important for **reproducibility** and **replicability**
  - Science of all types increasingly values being able to reproduce and replicate studies
  - At the *AJPS*, which has a strict reproducibility requirement, only
- Generally speaking, it is also smart to quickly develop good habits and be consistent across projects
  - For example, I always label my party identification variable "pid" and code it range from -3 (Strong Democrat) to 3 (Strong Republican)
  - This is one less thing I need to look up when returning to an old project, or make a decision about when I start a new one

Objects

- R is an "object-oriented" computing environment
- Information of all sorts – whether it be a whole dataset, a single variable, or a single numerical value – is stored in objects
- An object is a data structure having some attributes and methods which act on its attributes
- In plain language, an object is an empty entity which we give meaning to by assigning things to it – we store stuff there
- Datasets are objects, variables within datasets are objects, functions are objects, options are objects...EVERYTHING in R is an object

Objects

- With objects stored within other objects, things can get tricky...
  - Need to specify where variables are located with some functions (graphs, regression)
  - Otherwise, the $ operator addresses a column (variable) *in a data frame* by name
- For example: we have a dataset assigned to object, "anes", and variable, "pid", that we're interested in
  - Many functions will require `anes$pid` in order to locate the "pid" variable
  - Means "grab the 'pid' variable from the 'anes' dataset"
  - Only works if variable/column has name assigned

Some Basic Functions

See R script file called "Intro to R" for examples

Packages

- A "package" is a bundle of code, data, and documentation
- Most importantly for us, packages include functions to carry out specific tasks that were written by others
- Anyone can create a package and upload it to CRAN for free distribution to anyone who wants it
- MOST of what we will be dealing with in our class can be done with the R `base` package
  - This package comes standard with R when you download it, and does not require loading when R/RStudio is opened
- We will, however, be using a package – `lattice` – to construct statistical graphics

Packages

- Obtaining packages
  - Before loading a package for use in a session, we must download them from the CRAN package repository
  - Last time I checked, there were 13,515 available packages
  - How: Packages $\rightarrow$ Install $\rightarrow$ search package and click "Install"
  - Once packages have been downloaded, you'll never have to do this step again (until they update R another version...packages then need to be rebuilt)
- Loading packages
  - Once packages are downloaded, you must load them (i.e., tell RStudio that you're going to use them)
  - Best to do this at the beginning of your session, and top of your script
  - Simply type `library(PACKAGE NAME)`, and submit to the console
  - Must do this each time you reopen RStudio
  - If you go to use a function contained in a package that isn't loaded, you'll receive an error message

Working Directory

- It is useful to set a working directory at the beginning of your session (and, therefore, beginning of code)
  - Note: all of my R scripts begin by loading packages I'll be using and setting the working directory
- The working directory is a hierarchical path of user accounts and files leading to the folder from which one will be procuring data, and where one will be saving output
- For example:

```
``/Users/adamenders/My Courses/Intro to Data Analysis/''
```

- Can see what working directory is with `getwd()`
- Can set working directory with `setwd(``PATH'')`
- After WD is set, you can save files without thinking about where they're going and immediately grab files from that location

Importing Data

- Can import data from several different file types:
  - Other statistical analysis software like Stata (.dta), SPSS (.sav), and SAS (.ssd)
  - Microsoft Excel (either .csv or .xlsx)
  - Tab-delimited files (.txt)
- All of the datasets we will use in this class will be comma-separated values (.csv) files or Stata data files (.dta)
  - Doesn't really matter; I prefer .csv over .xlsx for the flexibility
- Comma-separated values: a comma separates the entries (either textual/numerical) associated with individual columns within a row

Importing Data

- All import functions have the same form: command → file extension → options
- For .txt or .csv: `read.table(...)`
- For .csv: `read.csv(...)`
- For .dta: `read.dta(...)`
- Fill in parentheses with file extension in quotation marks

```
anes <- read.dta("/Users/adamenders/Dropbox/My Courses/
Intro to Data Analysis/Data and Code/anes2016.dta")
```

- Notice that I assign, using the "<-" operator, my dataset to an object called "anes"
- Here's a reasonably good (and up-to-date) tutorial for specific data types, should it be useful

Importing Data

See R script file called "Intro to R" for examples

Exporting Data

- I do this fairly rarely when...
  - A project is complete and the paper is published
  - A particular model takes a long time to estimate and I don't want to wait every time I work on the project (won't have that problem in here)
- Why?
  - No need...I usually run the code from the bottom up on the "raw" data file each time
  - This prevents me from saving over the data or unknowingly altering the data
  - Also ensures that code runs each time and results are reproducible
- That said, it is as simple as importing data, and commands take a similar form

Exporting Data

- Writing data to tab-delimited text file (.txt)
  - `write.table(DATA OBJECT, "FILE EXTENSION.txt", sep="")`
  - This assumes column names, but not row names
  - Can alter defaults using the following options (i.e., appear after a common within the parentheses of the function): `col.names=FALSE` and `row.names=FALSE`
- Writing data to comma-delimited text file (.csv)
  - `write.table(DATA OBJECT, "FILE EXTENSION.csv", sep=",")`
  - `write.csv(DATA OBJECT, ''FILE EXTENSION.csv'')`
- Writing data to Stata data file (.dta)
  - `write.dta(DATA OBJECT, "FILE EXTENSION.dta")`
- Similar functions for .xlsx, .sav, and .ssd

Measures of Central Tendency

- `summary(OBJECT)`: interquartile range, min/max values, mean
- `mean(OBJECT)`: mean
- `median(OBJECT)`: median
- Oddly enough, no built-in function for mode
  - OK, because it's rarely useful, and can almost always be gleaned from visualization

Measures of Dispersion

- `summary(OBJECT)`: interquartile range, min/max values, mean
- `sd(OBJECT)`: standard deviation
- `var(OBJECT)`: variance
- `range(OBJECT)`: range (smallest and largest observations)

Distributions

- `table(OBJECT)`: frequency table
- `stem(OBJECT)`: stem-and-leaf plot
- Will look at graphical functions in the `lattice` package next week
  - Can construct powerful bar charts, histograms, scatterplots, box plots, and violin plots

Data Management

- In class examples and homework assignments will generally be fairly "clean"
  - Most assignments will not include missing data or require you to recode/construct new variables
- Real data – like the data for your final report – will not be so clean
- Some basic data management tasks:
  - Constructing new variables
  - Recoding existing variables
  - Dealing with missing data
- Executing these tasks will require knowledge of lots of other functions useful for subsetting data, targeting particular values of a variable, locating observations with missing data, etc.

Recoding and Creating Variables

- Sometimes the way data comes pre-packaged is...suboptimal in some way
- In particular, I might want to change the way a variable is measured – how numbers are assigned to the unique attributes
  - For example, most measurement begins with the number 0 and proceeds linearly from there (e.g., 1, 2, 3, 4...)
  - I like to code partisan and ideological self-identifications (which have 7 categories each) to range from -3 to 3, rather than 1-7
- Some basic principles of recoding data:
  - Always generate a new variable – do NOT write over an existing one
  - Variable names should be short, yet descriptive
  - Never capitalize variable names or include non-numeric characters (e.g., #, !)
  - Always double check your work to make sure the recode was executed properly

Recoding and Creating Variables

See R script file called "Intro to R" for examples

Dealing with Missing Data

- Generally speaking, when we refer to "missing data" we mean values on some variable are missing for a given observation
  - In other words, there is a "blank" cell in the dataset
- How can we compute the correlation between X and Y if, for a given observation in our dataset, there isn't a value for X?
- Different R functions deal with missing data in different ways
  - Most descriptive statistics functions simply ignore it and produce the quantities of interest
  - Many functions for carrying out bivariate and multivariate analyses require that missing data be omitted
- Usually, we "listwise" delete – if any variable included in our analysis is missing for a given observations, we remove the entire observation

Dealing with Missing Data

- The `is.na(...)` function prints the data with "TRUE" (missing) or "FALSE" (non-missing) in each cell
  - Just tells which cases have missing data and where, doesn't remove them
- The `na.omit(...)` prints the object with observations (i.e., entire rows) with missing data on any variables (columns) suppressed
  - This is useful when interacting with functions that require complete data, but that do not automatically remove observations with missing data for you
  - Can use this function to assign the complete dataset to a new object
  - For example: `data.nomissing <- na.omit(data)`
  - Need to be careful about omitting too many cases – we're usually not interested in all variables in a dataset
- Some statistical functions have an option that will remove observations for missing data for you
  - Usually the option is something like: `na.rm = TRUE`

Subsetting Data

- Beyond removing missing data, we might want to reduce our datasets in order to...
  - Conduct statistical operations on certain cases (e.g., people who are college graduates)
  - Simply make the data more manageable
- `subset(...)` is the most useful function for this purpose
  - Can add multiple conditions within parentheses to subset data by multiple criteria
  - Can assign subsetted data object to a new object
- **See "Intro to R" script for examples**

Other Useful Functions

- For seeing the basic dimensions of a dataset:
  - `head(...)`: prints the first 6 observations (rows)
  - `tail(...)`: prints the last 6 observations (rows)
  - `nrow(...)`: prints the number of rows in the dataset/object
  - `ncol(...)`: prints the number of columns in the dataset/object
- `data.frame(...)`: tells R to treat the object in parentheses as a dataset, and not just a matrix or vector
  - This is particularly when you create a dataset in R, or pull particular variables/observations from an existing dataset
- `rbind(...)`: combines objects (matrices, vectors, or data frames) by row
- `cbind(...)`: combines objects (matrices, vectors, or data frames) by column
- **See the "Reference Card" on Blackboard for more functions that might come in handy!**

Common Error Messages

Errors appear in red in the console output:

- `could not find function "FUNCTION NAME"`
  - Probably means you didn't load the package the function is contained in
- `object 'OBJECT NAME' not found`
  - You failed to define an object you thought you defined (e.g., a dataset, variable)
- `unable to open file: 'No such file or directory'`
  - You misspecified where the data is located on your machine
- `unexpected ')' in "CODE"`
  - There is an extra parenthesis (same for] and }) in your line
- `unexpected ')' in "CODE"`
  - There is an extra parenthesis (same for] and }) in your line
  - You're missing a comma somewhere

Don't become discouraged by errors – you'll be running into them a lot

Tips, Hints, and Words of Encouragement

- Science is iterative – we never get all of right the first time
- Certainly goes for individual elements of the scientific process, such as data analysis
- You're going to screw up all the time – don't worry about it, LEARN from it
- R will feel intimidating and sometimes you'll feel like a total dummy – EVERYONE experiences this
- Use notes, slides, course materials, official documentation, the web, and each other for help
- Persistence is the best predictor of success
  - You'll get plenty of practice with homework assignments
  - Should also run examples at home, conduct analyses on your own data
  - Play with the code to see what does what, dependencies, options, etc.