R for Health Data Science

Week 1: Introduction to Analysis with R

Sam Stewart, PhD

Medical Informatics
Department of CH&E, Faculty of Medicine
Dalhousie University, Halifax, Canada

February 12, 2021

Objectives for Day 1

- Begin using R
- Learn principles of statistical programming in R Studio
- Learn how to read in and begin working with data in R

- You should all have R downloaded and ready to run
  - If you don't go to `https://rstudio.com` and download the free, desktop version of RStudio
  - There are other options for running R, but that would be my suggestion
- You'll also need to download the R software - `https://mirror.its.dal.ca/cran/`
  - This is an independent download from RStudio, you'll need both for this course

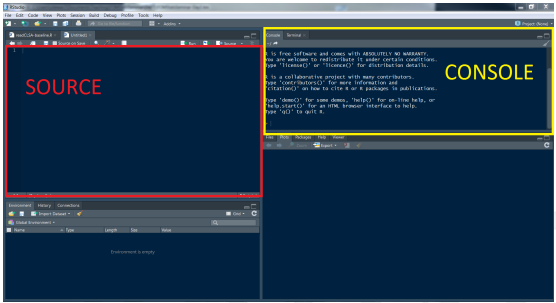
- You should be enrolled in a Brightspace course called "Online Community - Intro to R for Health Data Analysts" where all the course material will be shared
- I will also be making the course material available on github, see `https://github.com/samstewart11/R-for-HDS`
- Class recordings will be shared on the Teams space for this course
  - They will also be uploaded to Youtube, available in this playlist: `https://www.youtube.com/playlist?list=PLJO16lSgnGtgA276LxzOKIG_YXY7s4lRc`

R Statistical Programming Language[1]

- R is a statistical **programming** language, first published in 1993
  - Other popular statistical tools (SAS, SPSS, Stata, ...) are primarily analytic in nature
  - R is the only common statistical language that can function as a true programming tool
  - Is more comparable to Python than SAS
- Is the most powerful of the statistical language
- Has the steepest learning curve of the statistical languages, particularly if you have no programming experience

[1] `https://www.guru99.com/r-tutorial.html`

Rstudio



- RStudio is the best way to program in R
- If you haven't installed R Studio already, do so ASAP
- There are 4 panes here, but two are important

  Source: This is where your *script* will go

  Console: This is where the R terminal is

- Go to Tools > Global Options > Pane Layout to put the source in the top left, console in the top right
- If you got to "Appearance" in Global Options I use the Cobalt theme

Workspace

- R is a workspace language - when you run a command it produces an object that is stored in an active "workspace"
- Your script will be where you write the code needed to run your analysis
- It will execute in the console, and will be saved as the code executes
- Once created objects can be referenced and re-used

```
#starting a line with # makes a comment that won't execute when run
x = 7
y = 'string'
z = runif(100,0,1)
```

Data Types in R

- R is an objected-oriented programming language
  - If you're familiar with real OO languages then prepare to be underwhelmed, but it is loosely OO
- The general process is to create objects and then run *functions* on them
- We'll look at 5 basic data storage types in R
  - Scalars
  - Vectors
  - Matrices and Data Frames
  - Lists

Basic Data Types

- Numbers in R are called **numeric**
- TRUE/FALSE values are called **logicals**
- **Strings** are words, enclosed in either single or double quotes
- These examples are all *scalars*, or single values

```
x = 42.42 #numeric
x2 <- 42 #also numeric
string = 'dolphin' #string
string2 = "another dolphin" #string
t = TRUE #logical
t2 = FALSE
```

Variables

- All objects in R are stored in *variables*, or the left hand side of the equation
- Almost any combination of characters can be used as a variable
  - Can't have spaces
  - Can't *start* with a number
  - Can't have a $, #, probably some others
  - In general use letters, numbers and '.'
- Assignment can be done with = or <−, though I will almost always use =

Vectors

- Vectors are 1-dimensional arrays of numbers, i.e. a sequence of numbers
- The elements of a vector can be referenced using square-brackets, []
- The first element is 1 (and not 0 as is true for some programming languages)

```
x = c(21,2,42)
y = 1:10
z = c("first","second","third","fourth")
x[1] #21
y[3] #3
z[4] #"fourth"
z[5] #NA
```

Simple Operators

- Simple math is done using `+`, `-`, `*`, `/`
- Exponents are performed using either $\wedge$ or `**`
- Comparisons are done using == (equal to), `!=` (not equal to), $>, <$ for greater and less than
- Can combine comparisons using & (AND) and | (OR)

Simple Operators

```
x = 7
y = 5
x+y
x-y
x*y
x/y
x^y
x==y
x!=y
y==5
(x!=y & y==5)
(x!=y & x==y)
(x!=y | x==y)
```

Matrices

- 2-dimensional structures with rows and columns
- Reference the elements using the square bracket notation, but with a comma in between
  - `x[3,5]` gives you the value at the third row, fifth column of `x`
- To create matrices we need the `matrix()` function, which allows us to explore the concept of functions, and help files

```
x = matrix(1:10,nrow=2,byrow=TRUE)
x2 = matrix(c("a","b","c","d"),nrow=2,byrow=FALSE)
```

Functions

- Functions are called to perform tasks and create objects in R
- `matrix(data,nrow,byrow)` is an example of a function, which takes several different arguments
- There are a couple of different ways to find the arguments for a function
  - `help(matrix)` will bring up the help file, which provides all the needed information
  - If you type the name of the function in RStudio and then wait it will give you some advice about function arguments as a hover-over

Factors

- Many of the variables you will work with in your research are not continuous
- Categorical data is data that takes one of a set of unordered levels
- Examples include: Province, hair color, diagnosis, race, ...
- You may have learned about **dummy coding** of categorical variables in previous statistical education
  - Let $x_1 = 1$ if Province=NS, 0 otherwise
  - Let $x_2 = 1$ if Province=NB, 0 otherwise
- There's no need to dummy code variables in R, most libraries and functions will operate on factor variables

Factors in R

```
prov = c('NS','NS','NB','NB','PEI','NS','PEI','NB','NB','NFLD')
prov.factor  = factor(prov)
```

- Factors are very important for your analysis
- Getting the levels right, getting the right first level (i.e. reference level)
- Big part of the data cleaning process, which we will explore shortly

Data Frames

- **Data Frames** are the most natural approximation to spreadsheets in R
- Like matrices (they're 2-dimensional representations of data) where rows represent subjects (experiments, observations, subjects, patients, ...) and columns represents variables/attributes for those subjects
- Different from matrices in that the columns can be different data types
- Your research data will almost always be stored in a data frame, at least when it's read in

Data Frames

```
w = 1:4
x = c("A","B","C","D")
y = prov[c(1,4,5,2)]
z = LETTERS[21:24]
df1 = data.frame(w,x,y,z)
df2 = data.frame(first=w,second=x,third=y,fourth=z,
                 stringsAsFactors=FALSE)
df1
df2
df1[1,]
df1[,2]
df2[1,]
df2[,2]
df1[1:2,]
df2[3:4,1:3]
```

Referencing and Building Data Frames

- The $ operator is used to reference the elements of an object in R, we will use it a lot
- We can reference the columns of a data frame using the dollar sign
- We can add to a data frame with it as well

```
df1$x
df2$third
df2$fifth = c(5,6,7,8)
df2
```

Lists

- Lists are general structures for storing anything, or a mix of things
- Vectors, matrices and data frames have strict structures: fixed numbers of elements in the row/column, or a set data type
- A list is like a "bag" that you can put things in and pull them out later
- Lists become increasingly useful as your programs become more complex
- Many functions will return lists, most notably the `apply` family of functions

lists

```
l1 = list(w,x,y,z)
l2 = list(first=w,second=x,third=y,fourth=z)
l2$fifth=df1
l1[[1]]
l2[[3]]
l3$third
l2$fifth
```

Today's Code

- You should have received a .R file for the course, if not, it's available on the course website, or Github
- Try running the first 86 lines of the script (everything up to read.csv)
- This is all the code that I have shown in the slides so far, make sure you can run it and understand it
- We'll take this time to go over the basics of RStudio - running code, saving files, restarting sessions, general software practice

RStudio Content to Cover

Really just notes for Sam

- Pane contents, pane layout, graphical themes
- Script vs console
- File management (in R and in general)
  - We'll deal with R-projects later in the course, just focus on scripts for now
- Running things from the script (Ctrl+Enter, command buttons)

Analytic Process

- It's important to separate the data pre-processing from the analysis process
- Performing the data cleaning before the analysis begins is essential to ensuring consistent and reproducible results
- R does not produce a lot of output by default, so reporting is a separate step from analysis
  - This allows you more control over how you report your results

Reading in Data[2]

- Your data will typically be stored in external files
- R can read almost any data type, but .csv files are the most common
- `read.csv` or `read.table` are the most common ways to read in your data
- For these sessions we'll be working with a dataset that I've made available on the web, which can be read in directly
  - You should have your own dataset to work with, and I would encourage you to work with that, but if you want to follow along with the examples on the slides this is the dataset I will use

[2] `https://www.datacamp.com/community/tutorials/r-data-import-tutorial`

Reading in Data

```
dat = read.csv("C:\\Users\\sstewar2\\Documents\\Teaching\\HINF6030\\
    Data\\framingham.csv",header=TRUE,na.strings=".",stringsAsFactors
    =FALSE)
#OR
dat = read.csv("Data/framingham.csv",header=TRUE,na.strings=".",
    stringsAsFactors=FALSE)
#OR
dat = read.csv("https://raw.githubusercontent.com/samstewart11/R-for
    -HDS/main/data/framinghamFirst.csv",header=TRUE,na.strings=".",
    stringsAsFactors=FALSE)
head(dat)
```

Other Data Types

Data Source	Library	Function
Excel (xlsx)	`readxl`	`read_excel`
Excel (xls)		
SPSS	`foreign`	`read.spss`
STATA (13)	`readstata13`	`read.dta13`
SAS	`sas7bdat`	`read.sas7bdat`
MySQL	`RMySQL`	`dbConnect, dbGetQuery, ...`
...		

- Almost any data format can be read into R, you just need to find the right function in the right library

Libraries

- The base installation of R has a lot of useful functions, but most times you will need to load additional libraries
- R has over 13,000 libraries available on CRAN for download
- You can download them from CRAN using the command `install.packages` and load them using `library`

```
install.packages("Hmisc")
install.packages("psych")
library(Hmisc)
library(psych)
```

Loading vs Referencing Libraries in R

```
library(psych)
describe(dat)
#OR
psych::describe(dat)
```

- You don't use the `library::function` formatting that often
- Can be helpful if you only want one function from an unrelated library
- Necessary if two libraries contain the same function
  - `psych::describe` and `Hmisc::describe` are both useful summary functions, I usually call them with their library prefix
  - `car::Anova` is a much better function than `base::anova`, so I often call it with the full prefix just to make sure I use the right one

Data Cleaning

- Data cleaning is often the longest and most arduous part of any analysis
- It's hard to come up with a *guide* for data cleaning, as so many things can arise that are problematic
- For R specifically, there are a couple of things that need to be considered
  - Are numbers formatted correctly?
  - Are missing values handled correctly?
  - Are factor variables formatted correctly?
- Data cleaning is also where generated variables should be created - if your data has the results of an SF-36 survey then you should calculate the SF-36 scores before your analysis begins

Exploring Data

- Reading data in is an iterative process, as you find new challenges with your data
- Need ways to quickly explore your data
- There are three good, general-purpose data summary commands that I like for data exploration
  - `str` is a command to quickly summarize the type and contents of each variable
  - `psych::describe` is a good, quick numeric
  - `Hmisc::describe` is a more detailed summary - my favourite, but can take up some space

Exploring Data

```
#describing data
str(dat)#describes variables
psych::describe(dat)#summarizes continuous variables
Hmisc::describe(dat)#describes and summarizes all variables
```

Getting Correct Variables Types

- Want to make sure variables are correct
  - Numbers are numeric
  - Strings are characters
  - Factors are organized correctly

Step 1: Characters vs Factors

- R tries to read in variables as numbers
- If it encounters a non-number field, it reads it in as a character
  - Some functions then convert that character to a factor
- For most projects I would advise against that last conversion - leave non-numeric columns as characters
  - You should setup the factor levels yourself, controlling reference levels
  - Numeric columns that have an incorrect non-numeric entry are easier to solve when they are characters

Converting Numerics

as.numeric

```
d1 = read.csv("data/testData01.csv",header=TRUE)
str(d1)
d1$num1 = as.numeric(d1$num1)
str(d1)

d2 = read.csv("data/testData01.csv",header=TRUE,na.strings = '.')
str(d2)
```

- as.numeric converts characters to strings
- Telling the function what missing is recorded as can save a lot of headache (using the na.strings argument

Formatting Factors

- The `factor()` function takes two arguments that might be of value
  - `levels` is where the levels to be considered are listed
  - `labels` lets you change the labels on the given levels
- Make sure that your factors are correct using the `table()` function

```
dat$SEX = factor(dat$SEX,levels=c(1,2),labels=c("M","F"))
table(dat$SEX)
```

Creating Categories

- Often we want to convert continuous to categorical variables
- Several ways to do this, but the `cut()` function is probably the most efficient

```
dat$BMICat = cut(dat$BMI,breaks=c(0,18.5,25,30,Inf))
table(dat$BMICat)
```

Breakout Activity

- I want you to practice loading and cleaning a dataset
- There's a dataset on the course website called `telecom.csv` that has churn data for a cell phone company
- Download the dataset and then do the following
  - Read it into R
  - Check the variable types
  - Make sure that the variable types are correct
- I'll break you up into groups of 5-6 and we'll take 20-30 minutes (until 10:45) to get that work done
- We'll then re-convene to discuss what steps we took