Introduction to R

Chiara Forrester & Teal Potter

6/27/2017

Welcome to R! Before this section, we discussed how to best format your datasets for ease of analysis. As a summary this discussion included best practices for: -labeling columns (short and no spaces), -NAs vs. 0s, formatting (what data are organized in columns vs rows), -keeping datasets clean and simple (i.e. no blank columns or rows within dataset)

Uploading your dataset

Right now, you are reading what is called the "Script". This is where you can write your code and save it in a .R file for future use. If you are in RStudio (which we hope you are), you will see the "Console" below. If you type code into that box, you can run it (like hitting an equal sign on a calculator) but it will not save. Instead, you can write code in the script, run it in the script, and see the output in the Console. Whenever you run a line of code, the output will be shown in this window as well as a copy of the code you just ran. In the top right of R studio, you will see your "Global Environment", which will become a list of datasets and other objects (defined later) you have created as you write your code. To view a dataset, you can simply click the spreadsheet icon on the far right of the row. The blue arrow icon on the far left will give you a list of the variables included in that dataset, as well as the format R believes that column to be (e.g. number, integer, factor).

BTW, to run a line of code on press the "command" key on your keyboard and the "return" key at the same time (note that it won't work if you hit "return" before "command". Ctr + enter for Windows

Below, we upload a dataset into R and call it "mydat". Run the line of code and navigate to the file on your computer. This is the most common data import method.

You can see that the dataset is now in your global environment (top right of R studio).

Any time you see "<-": the left side is what you are naming the object (could be a dataset, statistical model, etc) and the right side is the content of that object.

Important note! Your data should be saved as a ".csv" file (Comma Separated Values) for R to read it. Just save your excel file as a .csv and you're good to go!

```
mydat <- read.csv("file_name.csv") #replace file_name with the name you gave your .csv spreadsheet
```

if you are struggling to get line of code to work properly there are a couple of alternative methods that are more user friendly. First:

```
mydat<-read.csv(file.choose())
```

A second option for uploading a dataset would be to go under the "Files" tab in the box on the bottom right of your RStudio console. You can then navigate to the dataset, click on it and ask it to import the file. It will likely prompt you to install the "readr" package.

FYI, now that you have saved your dataset to a name you chose (mydat), this dataset is now referred to as an "object". The entities that R creates and manipulates are known as objects. Objects can be whole datasets, lists of numbers, individual numbers etc. Each of these types of data are described in the glossary at the bottom of this script.

Some best practices for working in R

Set working directory

Your working directory is the location on your computer that R will go to find files when you upload them using a name (read.csv("file_name.csv")) as oppposed to using the file.choose() function. Edit the code below to navigate to your own preferred working directory (e.g. your desktop or a specific folder). R will also default to saving your script and other objects to this location.

```
setwd("file_path")  #e.g.  setwd("/Users/tealpotter/Desktop")
##Save your script often
```

If you just want to check what your working directory is set to, use:

```
getwd()
```

Annotate/comment your code.

While hashtags are super in right now, that's not why we're using them all over this script! Anything with a # at the beginning of the line denotes a comment. R will recognize anything folling # is not code and will not attempt to run it as code, allowing you to run your entire script with comments in it. We reccomend that you comment your code to be able to refer back and remember what your code actually does! It is often helpful to comment what test you are using, and the biological question you are asking. In addition to helping you organize your thoughts, it's also very helpful when sharing code between people.

Example of how to organize your data using comments (put a # in front of the below lines): QUESTION – Does chocolate actually make people happier? Test – T-test comparing levels of happiness between people that eat chocolate and those who don't Code for test below

Set preferences in R studio

There are many ways to personalize your R studio experience. Too see a full menu of preferences go to the R Studio tab on the bar at the very top of your desktop (title bar). Navigate to "preferences" in the dropbox menu. Here, you'll find settings including fun options like color/font preferences (under "appearance") and practical ones like soft-wrap (under "code", makes it so that your code will automatically start a new line to fit in the script window).
tools -> global options for windows

Functions

What is a function? You've already used some! "setwd" and "read.cvs" are functions. Functions are like verbs in R, just like objects are the nouns. Functions do something specific to your data (to an object). Some basic functions are "mean" and "sum". To use a function type it out and include the object name you want the funtion to compute with in parantheses after the fuction name. You can also sometimes use numbers instead of an object that stores the numbers in the parentheses. See these two examples that acomplish the same thing:

```
sum(2,8,5)
```

```
## [1] 15
```

```
myvector <- c(2,8,5)  #actually using another function "c" here to save this ordered list/string of num
sum(myvector)
```

```
## [1] 15
```

Housekeeping functions

The command "head()" shows the first several rows of the object. This helps to remind you exactly what you named each column. This is important because you need to copy this exactly when writing the code – for example, if you asked it to analyze the relationship between "bears" and "beets" but they were named "Bears" and "Beets" in your dataset, it would tell you that those data don't exist.

```
head(mydat)
tail(mydat)  #gives the last couple rows
```

More functions

```
names(mydat)   # gives the headers in your dataset
length(mydat)  #gives the number of columns in your dataset
dim(mydat)     #gives the number of rows and number of columns in that order
colSums(mydat) #gives the sum of all the numbers within each column if all are numeric
```

The command "str()" shows the structure of the object (the quantity and type of data is stored in rows and columns) This also tells you what it is reading each variable in your dataset as. For example, you might have number of flowers, which is a continuous variable. If R is reading this column in as the correct format, it would call it "int" (integer) or "num" (number). On the other hand, you might have presence absence data where the cells either have a "0" or "1" in the dataset, indicating presence or absence. Because R just sees numbers, it may think that this column should be read as a number or integer, when in reality it should be a factor because there are only two levels.

```
str(mydat)
```

Referring to data in columns by header names

If I wanted to look at the data in one column of my dataset you can't just type the header name since R doesn't know which column of which dataset (go ahead and try it). Users often have multiple datasets that they are working on in R and names of headers are often reused (e.g. I have a column in all my datasets called "site"). There are a few different options for how to refer to a header (aka column name) of a dataset to see the data in that column. Here is one method that is commonly used:

```
mydat$name   #you have to replace "name" with an actual column name in the dataset
```

Help?

If you don't remember or know what a function does, you can ask R to show you the help documention on a function by typing ?(function_name). If you run the example below, info about that function and how to use it will pop up in the lower right window in RStudio.

```
?str
```

Changing the type of data in an object

If presence-absence data are being read in as a number/integer, you can run the line of code below. This line tells R to read that column as a factor.

```
mydat$ColumnName <- as.factor(mydat$ColumnName)
```

Alternatively, if R is reading something as a factor that should be a number, you can run:

```
mydat$ColumnName <- as.number(mydat$ColumnName)
```

Downloading packages

The basic version of R you are using has many useful functions that you can use anytime. However, if you want to do anything at all complicated you will most likely want to use functions that are not supplied in base R. To access the sets of functions that people have written and published online, you will have to install the package that conatains the functions you need (to run an advanced statistical test for example). A "package" is the code that you save in R (the code is hidden) so that you can use new functions like you normally would in base R. To install a package:

```
install.packages("package_name")
```

Alternatively, you can do the same thing by clicking the "Install" tab in the bottom right window in RStudio. Type in the name of the package you want to install and click Install.

Once a package is installed, you will still need to "attach" the package sot that it is active in R and you can use the functions it contains. Every time you open R and want to use functions in a package you will run the following line of code to be able to use the functions.

```
library(package_name)
```

Saving Files you make/modify in R

R is an extemely useful tool for reformatting data to make new versions of your dataset as well as making beautiful graphs. If you create new objects in R that you want to save to use in a paper or presentation or future data analysis, there are several ways to save these files somewhere on your computer.

To save a new version of your dataset you can use the code:

```
write.csv(object_name, "file_name_that_you_want_to_use.csv")
```

To save a graph as a pdf file (pdfs are good quality images)

```
pdf("file_name_that_you_want_to_use.pdf", width = 7, height = 5)
```

Note that these files will be automatically saved to your working directory. Remember how to check what folder on your computer your working directory is set to? Alternatively you can specify the full "path" to put this file in a specfic folder on your computer.

Alternatively you can save a plot that is visible in the bottom right window in RStudio by clicking on the "Export" tab.

Glosassary for basic R terminology

script: the file you write and save your code in, file name ends in .R

console: this is where you'll see lines of code that you've run, outputs from that code, and errors

global environment: this is where all of your objects are stored (viewed in a window in RStudio)

object: any data that you assign a name, typically data structures such as numbers, vectors, matrices and dataframes

function: a variable name in R or another package that is associated with hidden code that does something specific to your data. Functions are followed by parantheses containing the data and optional arguments to tell the function what data to operate on

argument: an argument is kind of like a function, but it is used inside the parantheses of a function and allows you to customize what the function does. Many arguments are optional. Arguments are separated by commas and followed by "=" instead of parantheses.

working directory: your working directory is an electronic folder where R looks for datafiles to load and deposits andy graphs or datasets you create and save while working in R. You can designate any folder to be your working directory.

path: To assign a working directory you need to tell R the path so that it can find that folder which is typically nested in folders. The order of nested folders backing out to you as a user on your operating system is the path.

package: a stand alone, published set of functions that someone has made available for you to use to have access to functions that are not otherwise available in base R (the main program you use when you open R/RStudio)

Names of different types of data/data structures:

integer whole numbers (no decimals) (e.g. 5)

number any real number (can have decimals or not) (e.g. = 5.43)

factor data that represent categories, often represented as letters or words (e.g. low, medium, and high)

character data that are represented as text in R. In the example above, "low", "med", and "high" are each a different character. When they are used together and assigned an order (R saves a hidden number series like 0,1,2 to remember the factor order) they are also factors!

vector a data series of the same type of data. (e.g. 4,5,3,4 or a column in a dataset)

matrix a dataset (has columns and rows) that only contains numbers

dataframe a dataset (has columns and rows) that contains different types of data such as numbers and text (called factors)