Beginning with R/RStudio 1

Neil J. Hatfield

1/11/2021

Welcome! Whether this is your first time using R/RStudio or not, I hope that this document will be helpful for getting started.

Installing R and RStudio

If you have not already done so, you will need to install both R and RStudio from the following links:

- Download R–Use version 4.0.1+ (current is 4.0.3)
- Download RStudio–Use version 1.4+

While you can use R without RStudio, you won't have as nice of an environment as you could if you use RStudio. RStudio is an Integrated Development Environment (IDE) that help you keep track of R code, output, and documents all in one place. You can't really use RStudio without R.

Getting R Ready to Go

If you already have R/RStudio installed, your next step is to make sure you have all of the packages you'll need. To assist with this, I've written a script that you can use. Simply copy the two lines of code below and then paste them in our Console of R/RStudio.

```
source("https://raw.github.com/neilhatfield/STAT461/master/rScripts/checkSetup.R")
checkSetup()
```

The script will run, occasionally printing messages to the console for you. Once that is finished, you're ready to start exploring.

Decision Time

When you're ready to start working in R/RStudio, you need to make a decision: which production tool kit are you going to use? You can choose any of the following:

- Use the Console directly
  - If you are just wanting to something quickly and don't care about saving your work, you can choose to use the Console directly.
- Write R scripts (*.R)
  - If you want to save your code for later use
- Write an R Markdown (*.Rmd)
  - If you want to create a document that contains narrative in addition to your code and can convert into a PDF/Word document quickly.

I typically use either an R file or R Markdown. I use R Markdown any time I need to weave statistical outputs (values of statistics, data visualizations, model summaries, etc.) with written narrative and explanations. This guide is written in R Markdown.

Create a New R File

To create a new R file in RStudio, click on the File menu and then mouse over New File. In the list that appears click on R Script. A new blank document will appear in the upper left pane of RStudio (unless you have rearranged the panes).

Create a New R Markdown File

To create a new R Markdown file in RStudio, click on the File menu, mouse over New File and click on R Markdown... This will launch the wizard you can add a title, author, and choose the default output types. I would recommend that you choose PDF or Word for your output. (These are the formats that can be uploaded for your assignments.)

Using the Homework Template

I have also made a homework template which you can use. This is an R Markdown file. I would download the template and then make copies for each new file. (Be sure to change the file name so you can keep track of which one is which.) Then double click on the file to open it in RStudio.

R Markdown Formatting

If you are not using R Markdown you can skip this section.

Formatting your text in R Markdown is not quite the same as in Word. If you are new to coding, look for the compass icon just above the editor window (close to where the editor meets the Environment pane). Click that icon to get a WYSIWYG editor that includes a formatting tool bar.

Otherwise, here is a quick rundown of formatting in R Markdown

- Headings -Use the # to create a heading; the more pound signs you use, higher the heading level. H1 is the highest heading; use H5 and H6 sparingly.
- Bold and Italics
  - Use * on both sides of a word/phrase to italicize the word/phrase. For example, *sample arithemtic mean* becomes *sample arithmetic mean*.
  - Use two underscores ___ on both sides of a word/phrase to for boldface. For example, ___bold___ becomes **bold**.
- Mathematics
  - R Markdown will use LaTeX syntax and display mathematical formulas for you. You can choose for math to be inline or display style (on its own line and centered)
  - Inline: Use \ (mathematical expression\ ) to create an inline mathematical display. For example, \ (y=x^2+\frac{3}{4}\ ) will become $y = x^2 + \frac{3}{4}$. (No space between the backslashes and the parentheses.)
  - Display: Replace the parentheses with square brackets. (No space between the backslashes and the square brackets.) Thus, \ [y=x^2+\frac{3}{4}\ ] will become
$$y = x^2 + \frac{3}{4}$$
  - For multiline mathematical expressions, you'll want to use the align environment. For example,

```
\ begin{align*}
H_0: y = \mu_{\cdot\cdot} + \epsilon_{ij}\\
H_1: y = \mu_{\cdot\cdot} + \alpha_{i} + \epsilon_{ij}
\ end{align*}
```

will become

$$\begin{aligned} H_0 : y &= \mu_{\cdot\cdot} + \epsilon_{ij} \\ H_1 : y &= \mu_{\cdot\cdot} + \alpha_i + \epsilon_{ij} \end{aligned}$$

There's plenty more you can explore. Here are few resources which are helpful:

- Xie, Allaire, and Grolemund's R Markdown: The Definitive Guide
- Xie's Kintr Options
- Ismay and Kennedy's Getting Used to R, RStudio, and R Markdown

Writing Code

There are two ways in which you can write code in an R Markdown file.

Inline Code

Inline code is great for reporting a value that you have stored in the environment, to do a quick calculation, or something else that is **simple**. To create some inline code, you'll need to start with a grave, ‘, the symbol located on the key just to the left of the 1 key. This will start your code; you'll close the code with another instance of the grave, ‘. You'll need to type a lower case r right after the first grave to state what language your code is supposed to be (in this case R). For example, ‘ r 1 + 2 ‘ will yield 3. Notice that inline code is evaluated and the result is displayed.

Code Chunks

Code chunks are much larger blocks where you can do more complicated things. For example, you can use code chunks to import a picture, load and clean data, create data visualizations, tables, and run models, etc.

You can add a code chunk manually by typing three graves in a row on new line. When you're finished with the chunk, you type another three graves. All of your code should be on lines in between these two lines with graves. Chunk options, including the language should be on the first line with graves enclosed in curly braces: {r chunkName [options]}

You can also use the insert code chunk button, and select R to get a code chunk.

Importing Data

There are functions and tools that you can use to load data.

Import Wizard

You can use the Import Dataset wizard by clicking on the icon next to the words “Import Dataset” in the Environment tab. This will load the data into your current environment and display the necessary code in the Console. Be sure to copy that code and place inside a code chunk of your R Markdown document. (When you knit R Markdown, a new temporary environment is created; if you don't have the code to the load the data, your knit will fail.)

Write Your Own Code

Another way (and a good practice to get into) is to write the code yourself. This is actually pretty easy. Most data files I provide you will either be *.CSV, *.DAT, or *.Rdata. Further, you can use this method to read data in from a URL.

```
# Basic Structure for reading in data
# CSV and DAT files
objectName <- read.table(
  file = "path/to/file.csv or URL",
  header = TRUE,
  sep = ","
)
```

You'll have to know where you have saved your file. For example, if I have a CSV saved to my desktop that I want to load, I would have to use `~/Desktop/mydata.csv`. You can simplify the file path if you set your working directory (which will be covered in the next section). Here's an example using a provided URL

```
# Importing via URL
oreoData <- read.table(
  file = "https://raw.github.com/neilhatfield/STAT461/master/dataFiles/classDemoOreo.dat",
  header = TRUE,
  sep = ","
)
```

If you are loading an RData file, all you need to do is the following:

```
load(file = "path/to/file.Rdata")
```

The data frames inside should immediately load to the environment.

Some Useful Functions

R has many functions built in. Each library/package you load will give you even more. However, I want to mention some basic functions that are incredibly useful.

`setwd` and `getwd`

A working directory (computer folder) is the file path that software uses to look for data and write output to. In R, you can type `getwd()` in the console to see where your R session is currently look. As I write this, my working directory is /Users/neilhatfield/Documents/GitHub/STAT461/demoFiles.

If you want to change the working directory, you can use the `setwd` function. For example, `setwd("/Users/neilhatfield/Desktop/")` would change my working directory to my desktop.

I recommend making an R Project for this class and then work in there. This will help you keep all of your files centralized.

`str`

The structure function, `str` allows you to see the structure of any object in your R session. This is useful when you want to see how R is thinking about an object rather than viewing all of the things inside of the object. Here's an example of `str` in action:

```
load(file = "../dataFiles/onewayExplorations.Rdata")
str(fabric)
```

```
## 'data.frame':    16 obs. of  2 variables:
##  $ loom    : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 2 2 2 2 3 3 ...
##  $ strength: num  98 97 99 96 91 90 93 92 96 95 ...
```

The `fabric` object is a data frame (`data.frame`) with 16 observations on two attributes. The variables to access those attributes are `fabric$loom` (a factor with 4 levels) and `fabric$strength` (a numeric value).

`View`

The `View` function will open up a window that shows you the contents of the object. This is particularly useful when you want to look at a data table or list. **This should only be used in the Console.** Using the `View` function inside code chunk in R Markdown tends to cause problems varying from extremely slow knitting or errors that prevent knitting.

In your console you would type `View(objectName)`.

`names`

Many objects in R will have other objects inside of them. The one that we'll be using the most are data frames. Each column has a name. You can call individual columns for analysis by `dataFrameName$columnName`. However, you have to know what the names of the columns are. You can see what these are in the `str` output but you can also use the `names` function to get a list of the column names. For example `names(fabric)` will return: loom, strength.

If a data frame has row names, you can use `row.names` to see what those names are. For the fabric data frame, the rows are named 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16.

Calling Individual Cells

While not a function per se, you can use row names and column names to grab individual elements from a data frame. For example, `fabric["4","strength"]` will return the strength value for the object named "4" in the data set; in this case, 96. Notice the use of quotation marks. If you were to use 4 instead of "4", you'll end up with the 4th *row* rather than the named row you would want.

You can also call individual cells via numeric indices. For example `fabric[3, 1]` would return the value in the third row, first column. Notice this isn't has helpful for knowing what you're getting as using names.

If you want an entire row, omit the column value but include the comma. For example, `fabric["4",]` or by index `fabric[3,]`).

If you want an entire column and you know the name, use `fabric$strength` format. If you don't know the name, but you know the index, you can use `fabric[,1]`. However, using the name is much better (and safer).

Formatting Numbers

There are two functions that can help you format numbers to make them look nice.

round

The first is `round`. This function takes two arguments: `x` the number/vector/data frame that you want to round, and `digits` which is the number of digits *after* the decimal you wish to keep. Notice that you can apply `round` to multiple values at time IF they are part of the same vector or data frame.

```
round(x = 124.512, digits = 2)
```

```
## [1] 124.51
```

```
round(x = 124.512, digits = 0)
```

```
## [1] 125
```

```
round(x = 124.512, digits = -1)
```

```
## [1] 120
```

```
head(x = round(x = fabric$strength, digits = 0), n = 2)
```

```
## [1] 98 97
```

Notice that you can use negative values for `digits`, which will shift the rounding to places left of the decimal; -1 will be to the nearest 10, -2 to the nearest 100, etc.

The last example line shows applying the `round` function to a vector. (The function `head` will limit the display to the first `n` observations.)

prettyNum

The other useful number formatting function is `prettyNum`. As the function name suggests, this makes your numbers look "pretty". By default, R does not make numbers easy to read when printed. For example, 1.2345679×10^9. However, we can add in commas AND round using `prettyNum`. The code `prettyNum(x = 12345.54321, digits = 7, big.mark = ",")` will yield 12,345.54–much easier to read.

The three important arguments of `prettyNum` are `x` the number/vector you want to manipulate, `digits` refers to the total number of *significant* digits that you want (negative values **NOT** allowed here), and `big.mark` which sets the symbol that gets used between three digit chunks.

Getting Help in R

There are a couple of ways that you can get help with a function in R.

In the lower right pane of RStudio, there should be a Help tab. There is a search box that will help you search through the packages of R for the function you want help with.

Additionally, you can type `?functionName` into the console to bring up the help page for a function you know the name of.

Further, you can place your cursor inside of function name which is in a inline code piece, a code chunk, R script file, or your current console line and then press the F1 key (Mac Users: fn + F1) and the help window will update to the appropriate documentation.

That wraps up this document for R/RStudio.