R, RStudio, and R Markdown

Adam Kuczynski

UNIVERISTY OF WASHINGTON

Course Objectives

Overarching goal: to develop a foundational knowledge in R

You will:

- Learn basic programming skills
- Develop data management and visualization skills in R
- Learn how to manipulate and analyze data in R
- Be prepared to master programming tasks you encounter in your coursework and research

Required Software

- R (Version 4.1+)
- RStudio (Version 1.4+)

- R is the programming language
  - Comes with its own GUI on Windows and Mac, but it's not great!
- RStudio is software to help you efficiently write R code
  - Called an **integrated development environment** (IDE)

Course Structure

Lecture

- Every Tuesday
- Focus on covering foundational knowledge in R
- Please ask questions throughout lecture!

Lab

- Every Thursday
- Focus on implementing the R skills covered in lecture
- Sometimes I will cover additional material in lab
- Mostly a space to work with each other on novel programming tasks, homework, and to get your questions answered

Assignments

- Weekly (due on Tuesdays before class)
- Designed to be challenging (depending on your experience, expect a to spend a couple of hours each week on HW)

A note about assignments

My hope is that these assignments will prepare you as much as possible for the "real world" of statistical computing. In approaching data tasks, it is often helpful to have examples of code that you have written in the past to perform a similar task. These assignments will likely function in that way. However, if you are facing a task "in real life" that is similar to a homework question (e.g., creating the same/similar figure for your manuscript), please get in touch with me and we can figure out a way to work that into your weekly assignment instead.

Logistics

Location

- Lecture: Zoom (932 9410 5848) on Tuesdays, 1:10-2:40pm
- Lab: Zoom (932 9410 5848) on Thursdays, 1:10-2:40pm
- Office Hours: by appointment
- Slack: click to join

Materials: on the course Canvas page

Grading

Getting help in Slack

- I'll monitor it as much as I can, but it's even better if you help each other!
- Put single lines of code between backticks (same key as '~')
- Put multiple lines of code after 3 backticks to create a code chunk. This makes it much easier to read and copy & paste your code
- Describe your problem clearly, including what you have tried already
- Include all the code that is needed to respond to your question

Don't ask questions like this:

> I tried lm(y~x) but it isn't working, how do I fix it?

Getting help in Slack

Instead, ask like this:

```
y <- seq(1:10) + rnorm(10)
x <- seq(0:10)
model <- lm(y ~ x)
```

Running the code above gives me the following error. Anyone know whats going on?

> ```
> Error in model.frame.default(formula = y ~ x,
> drop.unused.levels = TRUE) : variable lengths differ
> (found for 'x')
> ```

Lecture Plan

1. R, Rstudio, Rmarkdown, and Packages
2. Fundamentals of R
3. Control structures (`if`/`else`, `switch`, `for`, `while`, `repeat`, `break`, `next`), classes and methods, and vectorization
4. Functions, user-defined functions, and the `apply` family of functions
5. Importing/exporting data, data cleaning, working with strings (regular expressions)
6. Data manupulation and summarizing
7. Data visualization in base R
8. Data visualization in `ggplot2`
9. Intro to statistical computing in R (formulas, functions, and packages) and writing reproducible code (git, `renv`)

Slide Formatting

`Code` represents R code you type into the editor or console, for example: "you can use the `mean()` function to find the average of a vector of numbers. Sometimes `median()` is a better summary, though."

Code chunks that span the page represent *actual R code embedded in the slides*.

```
# Sometimes important lines of code will be highlighted!
10*9
```

```
## [1] 90
```

The lines preceded by `##` represent the output, or result, of running the code in the code chunk

R and RStudio

Why R?

R is a programming language built for statistical computing

Why use R if you are already familiar with other statistical software?

- R is *free*, so you don't need to connect to a terminal server or shell out tons of money
- R has a very large community, so there are tons of packages and it's easy to find help
- R is a programming language, so you can do almost anything you want with it (you don't have to rely on whatever is implemented in the software you use)
- R can handle almost any data format
- R makes it easy to create reproducible analysis
- R is increasingly becoming the norm in psychological science
- Find a job! R skills transfer well to other programming languages, and many stats-related jobs require proficiency in R

R Studio

R Studio is an **integrated development environment (IDE)** for R that will make your life *much easier*

Features of RStudio include:

- Auto-complete code, format code, and highlight syntax
- Organize your code, output, plots, and objects into one window
- View data and objects in readable and searchable format
- Seamless creation of RMarkdown documents

RStudio can also...

- Interact with git and github
- Run interactive tutorials
- Help you write in other languages like python, javascript, C++, and more!

Getting Started

To make your RStudio look like mine...

1. *Tools* > *Global Options* > *Pane Layout* and make top right your *Console*
2. *Tools* > *Global Options* > *Appearance* and select *Chaos* as your *Editor theme*
3. *Tools* > *Global Options* > *Code* > *Display* and select:
    - *Highlight selected line*
    - *Allow scroll past end of document*
    - *Highlight R function calls*

Open up RStudio now and choose *File* > *New File* > *R Script*.

Then, let's get oriented with the interface:

- *Top left*: Code **editor** pane, data viewer (browse with tabs)
- *Top right*: **Console** for running code (`>` prompt)
- *Bottom left*: List of objects in **environment**, code **history** tab.
- *Bottom right*: Browsing files, viewing plots, managing packages, and viewing help files.

Editing and Running Code

There are several ways to run R code in RStudio:

- Highlight lines in the **editor** window and click *Run* at the top or hit `Ctrl`+`Enter` or `⌘`+`Enter` to run them all
  - You can change these (other other hot keys) in *Tools* > *Global Options* > *Code* > *Modify Keyboard Shortcuts*
- With your cursor on a line you want to run, hit `Ctrl`+`Enter` or `⌘`+`Enter`. Your cursor moves to the next line, so you can run code sequentially with repeated presses
- Type individual lines in the **console** and press `Enter`
- In R Markdown documents, click within a code chunk and click the green arrow to run the chunk. The button beside that runs *all prior chunks*.
  - You can also type `Ctrl` + `Shift` + `Enter` when your cursor is inside a code chunk to run just that chunk.

The **console** will show the lines you ran followed by any printed output.

Incomplete Code

If you try to run incomplete code (e.g. leave off a parenthesis), R might show a `+` sign prompting you to finish the command:

```
> (11-2
+
```

Finish the command by typing `)` and hitting `Enter` or hit `Esc` to cancel code execution

Arithmetic operations in R

There are several arithmetic operators in R that are *extremely* useful:

- `+` addition, `-` subtraction, `*` multiplication, `/` division
- `^` (less commonly `**`) exponentiation
- `%*%` matrix multiplication
- `%/%` integer division (`9 %/% 2` is 4)
- `%%` modulo (returns the remainder of the left number divided by the right)
  - e.g., To check that a number is even, you can do `number %% 2`. If the return value is 0 it is even, otherwise it is odd.

```
2^2
```

```
## [1] 4
```

```
2**2
```

```
## [1] 4
```

```
4 %% 2
```

```
## [1] 0
```

```
5 %% 2
```

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

Now in your blank R document in the **editor**, typing the line `sqrt(25)` and either clicking *Run* or hitting `Ctrl` + `Enter` or `⌘` + `Enter`.

```
sqrt(25)
```

```
## [1] 5
```

R uses the PEMDAS order of operations:

```
# 2*3 then / 2 then + 25
25 + 2 * 3 / 2
```

```
## [1] 28
```

You can tell R to evaluate chunks of the equation together by using parentheses:

```
# 25 + 2 then *3 then /2
(25 + 2) * 3 / 2
```

```
## [1] 40.5
```

Functions and Help

`sqrt()` is an example of a **function** in R.

If we didn't have a good guess as to what `sqrt()` will do, we can type `?sqrt` in the console (`help(sqrt)` will also work) and look at the **Help** panel on the bottom right.

```
?sqrt
```

Arguments are the *inputs* to a function. In this case, the only argument to `sqrt()` is `x` which can be a number or a vector of numbers.

Help files provide documentation on how to use functions and what output functions produce

Creating Objects

R stores everything as an **object**, including data, functions, models, and output.

Creating an object can be done using the **assignment operator**: `<-`

```
luckynumber <- 13
```

Operators like `<-` are functions that look like symbols but typically sit between their arguments (e.g. numbers or objects) instead of having them inside `()` like in `sqrt(x)`

We can actually call operators like other functions by placing them between backticks:

```
`<-`(luckynumber, 100)
print(luckynumber)
```

```
## [1] 100
```

```
# 1 + 5
`+`(1, 5)
```

```
## [1] 6
```

```
5 %% 2
```

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

```
`%%`(5, 2)
```

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

You can even create your own operators (we'll get back to this in Week 4)

```
500 %largerthan% 50
```

```
## Yes,  500  is larger than  50
```

```
50 %largerthan% 500
```

```
## No,  50  is not larger than  500
```

Calling Objects

You can **call** (i.e., use) an object simply by using its name:

```
luckynumber <- 13
luckynumber + 1
```

```
## [1] 14
```

Object names should include letters, numbers, and underscores. They *can* include other characters (even spaces) if you surround the object name in backticks, but this is considered poor style and will likely lead to errors.

Object names have to start with a letter (unless surrounded by backticks)

Good

```
myobject <- "My New Object!"
```

Bad

```
1stobjectever! <- "My first object ever!"
```

Overwriting Objects

If you name an object the same name as an existing object, *it will overwrite it*:

```
age <- 30
print(age)
```

```
## [1] 30
```

```
age <- 40
print(age)
```

```
## [1] 40
```

You can create a copy of the object by assigning it a new name

```
object1 <- 100
object2 <- object1
print(object1)
```

```
## [1] 100
```

```
print(object2)
```

```
## [1] 100
```

This does *not* clone the object (i.e., changing one does not change the other)

```
object2 <- object2 + 1
print(object1)
```

```
## [1] 100
```

```
print(object2)
```

```
## [1] 101
```

Creating Vectors

A **vector** is a series of **elements**, such as numbers.

You can create a vector and store it as an object in the same way as we just did. To do this, use the function `c()` which stands for "**c**ombine"

```
myvector <- c(4, 9, 16, 25, 36)
print(myvector)
```

```
## [1]  4  9 16 25 36
```

You can provide a vector as an argument for many functions (more on this later in the course)

```
mean(myvector)
```

```
## [1] 18
```

More Complex Objects

The same principles can be used to create more complex objects like **matrices**, **arrays**, **lists**, and **dataframes** (lists which look like matrices but can hold multiple data types at once).

Most data sets you will work with will be read into R and stored as a **dataframe**, so this course will mainly focus on manipulating and visualizing these objects.

Before we get into these, let's revisit R Markdown.

R Markdown

R Markdown

Markdown is a simple language for creating formatted text using a plain-text editor. Markdown can be compiled (i.e., translated into) into several formats including HTML, PDF, and Word Docs!

R Markdown is an extension of regular vanilla markdown that allows you to embed R code and output inside your document

R Markdown Documents

To create a R Markdown file (.Rmd):

1. Choose *File* > *New File* > *R Markdown…*
2. Make sure *HTML Output* is selected and click OK
3. Save the file somewhere, call it `my_first_markdown.Rmd`
4. Click the *Knit HTML* button
5. Knitting progress is displayed in the *R Markdown* pane at the top right

You may also open up the file in your computer's browser if you so desire, using the *Open in Browser* button at the top of the preview window.

R Markdown Headers

The header of an .Rmd file is a YAML (YAML Ain't Markup Language) code block, and everything else is part of the main document.

```
---
title: "Untitled"
author: "Adam Kuczynski"
date: "June 21, 2021"
output: html_document
---
```

To control global output, you can modify the header[1].

```
output:
  html_document:
    theme: darkly
```

[1] YAML is space-sensitive! Newlines and indents matter!

R Markdown Syntax

Output	Syntax
bold/strong emphasis	`**bold/strong emphasis**`
italic/normal emphasis	`*italic/normal emphasis*`
# Header	`# Header`
## Subheader	`## Subheader`
### Subsubheader	`### Subsubheader`
> Block quote from famous person	`> Block quote from` `> famous person`

R Markdown Syntax

Output

1. Ordered lists
2. Second item
    1. With sublists
    2. Second subitem

- Unordered lists
- Are simple!
    - Even with subitems

UW Website

Syntax

```
1. Ordered lists
2. Second item
  1. With sublists
  2. Second subitem


- Unordered lists
- Are simple!
  - Even with subitems

[UW Website](http://www.uw.edu

![UW Logo](http://depts.washin
```

Formulae and Syntax

Output

You can put some math $y = \left(\frac{2}{3} \right)^2$ in your document

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Or a sentence with `code-looking font`.

Or a block of code:

```r
y <- 1:5
z <- y^2
```

Syntax

````
You can put some math $y= \left(
\frac{2}{3} \right)^2$ in you docum

`$$\bar{x}_n=\frac{1}{n}\sum_{i=1}^
x_i$$`

Or a sentence with `code-looking fo

Or a block of code:

    ```r
    y <- 1:5
    z <- y^2
    ```
````

R Markdown Tinkering

R Markdown documents can be modified in many ways. Visit these links for more information:

- Ways to modify the overall document appearance
- Ways to format parts of your document
- R Markdown: The Definitive Guide

R Code in R Markdown

Inside RMarkdown, lines of R code are called **chunks**. Code is placed between sets of three backticks and `{r}`.

This chunk of code:

````
```{r}
summary(cars)
```
````

Produces this output in your document:

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

Chunk Options

Chunks have options that control what happens with their code, such as:

- `echo=FALSE`: Keeps R code from being shown in the document
- `eval=FALSE`: Shows R code in the document without evaluating it
- `include=FALSE`: Hides all output but still runs code (good for `setup` chunks where you load packages!)
- `results='hide'`: Hides R's (non-plot) output from the document
- `cache=TRUE`: Saves results of running that chunk so if it takes a while, you won't have to re-run it each time you re-knit the document
- `fig.height=5, fig.width=5`: modify the dimensions of any plots that are generated in the chunk (units are in inches)

Some of these can be modified using the gear-shaped *Modify Chunk Options* button in each chunk. There are a *lot* of other options, however.

Chunk Options and Naming

Try adding or changing the chunk options (separated by commas) for the two chunks in `my_first_markdown.Rmd` and re-knitting to check what happens.

You can also name your chunks by putting something after the `r` before the chunk options.

````
```{r summarizing, echo=FALSE}
summary(cars)
```
````

After you name your chunks, look what happens in the dropdown on the bottom left of your editor pane.

Naming chunks allows you to browse through an RMarkdown document by named chunks.

You can also browse by sections named using headers and subheaders.

In-Line R code

Sometimes we want to insert a value directly into our text. We do that using code in single backticks starting off with `r`.

```
Today's date is `r Sys.Date()`
```

Today's date is 2021-06-22

You can also reference an object that you created in a chunk:

```r
age <- 30
```

```
This year he is `r age` years old.
```

This year he is 30 years old.

```
Next year he will be `r age + 1` years old.
```

Next year he will be 31 years old.

R Markdown is really powerful!

Having R input values directly into your document makes sure you are reporting your findings accurately:

- Never wonder where a quantity came from. Just look at your formula in your .Rmd file!
- Consistency! No "find/replace" mishaps; reference a variable in-line throughout your document without manually updating if the calculation changes (e.g. reporting sample sizes).
- You are more likely to make a typo in a "hard-coded" number than you are to write R code that somehow runs but gives you the wrong thing.

Example: Dynamic Dates

In your YAML header, make the date come from R's `Sys.Date()` function by changing:

```
date: "March 30, 2016"
```

to...

```
date: "`r Sys.Date()`"
```

Or even better, Format the output of `Sys.Date()`[1]:

```
date: "`r format(Sys.Date(), format = '%B %d, %Y')`"
```

June 21, 2021

[1] `format(Sys.Date(), format='%B %d, %Y')` says "format system date as month name (`%B`), day-of-month (`%d`), and four-digit year (`%Y`): June 21, 2021. See `?strptime` for these format codes.

R Markdown Tables

Tables and data.frames in R markdown print just like the all other output:

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear c
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3
```

But sometimes you want tables that look a little nicer in your output...

```
# Load the rmarkdown package (do this once in your session, at the
library(rmarkdown)
paged_table(mtcars)
```

	mpg <dbl>	cyl <dbl>	disp <dbl>	hp <dbl>	drat <dbl>	wt <dbl>	qsec <dbl>	vs <dbl>
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1

1-10 of 32 rows | 1-9 of 12 columns Previous **1** 2 3 4 Next

Packages

A lot of R's abilities come packaged with your base R installation, and you will use many of these in your work. Examples of these abilities include performing basic arithmetic, running linear models, and visualizing your data. However, you will also need to do things that aren't included in Base R's default functionality. For example, run structural equation models, estimate inter-rater reliability, and scrape data from the web. You *could* write all these functions yourself, but this is difficult and time consuming.

Packages are bundles of R functions that help you perform some kind of task. For example, `lavaan` is a package used for structural equation modeling, `psych` is a package used for estimating inter-rater reliability, and `rvest` is a package used for scraping data from the web.[1]

Packages are installed to your local computer. To see where R is looking for your currently installed packages, type `.libPaths()` in your console

[1] These packages are also capable of performing other tasks. Check them out online!

Installing packages is easy:

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

You only need to install a package *once* per R installation.

Use the **console** to install a package (*don't put this in your script*)

You can install several packages at once:

```
install.packages(c("psych", "lavaan", "rvest"))
```

To use a package in your script, use the `library()` function:

```
# The packages name can be in quotes, but doesn't have to be
library(psych)
```

You can also use a function from a package without loading that package by prepending the function call with the package name + `::`. For example:

```
psych::ICC()
```

library vs. require

We just learned that `library()` is used to load packages at the beginning of your script. You can also accomplish this with `require()`. However, this is unusual, slower, and has slightly different behavior.

The only thing `library()` does for you is load your packages. If you don't have the package installed, you will get an error. Most of the time this is what you want!

`require()` is designed to use inside R functions that you create yourself. If you don't have the package installed, you will get a warning[1], not an error, and the function will return `FALSE`. If the package *is* installed, the function will return `TRUE`. Most of the time this is extraneous behavior and makes reading your code more difficult.

Example of proper use of require:

```
if(!require("package")) install.packages("package")
```

Package Repositories

Most of the packages you will use are stored on CRAN (the Comprehensive R Archive Network), which is maintained by the folks at the R Project. By default, `install.packages()` will look at a CRAN mirror to download and install the requested packages.

```
getOption("repos")
```

```
##                             CRAN
## "https://cloud.r-project.org"
```

You can install packages from other sources as well. For example, to install a package from github:

```
library(devtools)
# install_github("DeveloperName/RepoName")
install_github("hadley/dplyr")
```

To install a package from your local computer:

```
install.packages(path_to_file, repos = NULL, type="source")
```