Introduction to R and R Markdown

Tami Ren

7/19/2022

Lecture Overview

Goals of the lesson:

- Students will be able to create an R Markdown document and knit it
- Understand the notion of object-oriented programming and run basic commands in R

Instructor Notes:

Lecture Content

R Markdown

R Markdown provides a flexible environment for writing reports and other documents. The beauty of R Markdown is that it allows you integrate R code and output with written analysis *in a single document*. For the computational portions of the last three problem sets, you may submit documents generated by R Markdown.

Why R Markdown?

1. Simplify your workflow
2. Communicate your results
3. Automate repetitive tasks

Geting started

To open a new R Markdown file, click `File` then `New File` then `R Markdown...` in R Studio. At this point, a dialog box will appear. By default, `Document` and `HTML` should be pre-selected in the dialog box. Type in a title and your name then click `OK`.

The new R Markdown document comes pre-loaded with examples. To compare the code to the document it produces, click the `Knit` button.

Anatomy of an R Markdown File

An R Markdown file is the source code for the document that you will generate. You can execute the source code by clicking the `Kint` button in R Studio. When you do this, R will start generating a document. The document will show up in a new window once R is done "knitting" the file.

R Markdown files are also known by their file extension: `.Rmd`.

YAML header

At the top of your `.Rmd` file, you should see some text sandwiched between triple dashes:

```
---
title: "A title"
author: "Your Name"
date: "06/27/2021"
output: html_document
---
```

This text is called the YAML header. It contains basic information about the document that you will create.

The argument `output: html_document` tells R Markdown to produce an HTML document. You can also tell R Markdown to produce other kinds of documents. For example, you will learn how to use R Markdown to produce MS Word documents further below.

Code chunks

Next, you should see some R code sandwiched by a pair of back ticks:

````
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
````

The pair of backticks defines a **code chunk**. A code chunk is a minature R script that lives inside the `.Rmd` file.

The code chunk above is named `setup`. Generally speaking, you don't have to name a code chunk, but names can be useful for helping your future self understand what your code chunk seeks to accomplish.

The argument `include=FALSE` is a code chunk option that instructs R to execute the code chunk without showing the code and the results of the code in the document. This option overides the default seting, which is to show the code and the results in the document.

The code `knitr::opts_chunk$set(echo = TRUE)` specifies default options for all code chunks.

You can add a new code chunk by clicking the `Insert` button then `R`, which results in

````
```{r}
```
````

You can write your R code of inside the code chunk:

```{r}
# Here is a comment inside of a code chunk
library(pacman)
p_load(tidyverse)
random_data <- tibble(
  x = rnorm(1000, mean = 5, sd = 2),
  y = 10 + x + rnorm(1000, mean = 0, sd = 1)
)
mean(random_data$y)
```

Note: In `.Rmd` files, R only executes code typed inside code chunks. If you write code outside the backticks that define a code chunk, then R will render the code as text when you `Knit` the file.

Markdown text

You can write text outside the YAML header and the code chunks.

Most of you write with "what you see is what you get" text editors, like MS Word. When you type text in MS Word, you see the final output as you write it. Markdown is different. To style your text in markdown, you have to use something called markdown syntax.

Markdown syntax:

- `text here` produces text here
- `**text here**` produces **text here**
- `*text here*` produces *text here*
- `***text here***` produces ***text here***
- `$Y_i = \beta_0 + \beta_1 X_i + u_i$` produces $Y_i = \beta_0 + \beta_1 X_i + u_i$

These are just a few examples things you can do with markdown syntax. R Studio's R Markdown Cheat Sheet has a more complete list of examples.

Output

HTML Documents

HTML documents are the default R Markdown output. For more details on the functionality of `output: html_document`, check out chapter 3.1 of R Markdown: The Definitive Guide.

MS Word Documents

You can also generate MS Word documents. As with the HTML documents, making Word documents in R Markdown can save you the hassle of copying and pasting your R code and output into an external document. The difference, however, is that you can edit the Word documents that R Markdown produces in the MS Word editor. This option gives you the best of both worlds. For more information, check out chapter 3.4 of R Markdown: The Definitive Guide.

Saving your work

Before knitting your `.Rmd` file, save it to your EC 320 folder. When you click `Knit`, the output document will show up in the same folder as your `.Rmd` file.

R Basics

R uses object-oriented programming. If you have never used this type of programming before, it can be a bit confusing at first. Essentially, R uses *functions*, which we apply to *objects*. More on this shortly, but if you aren't sure what a function does, or how it works, you can use ? before the function to get the documentation. Ex: `?mean` will bring up the help page for the `mean()` function. Try typing ?mean in the console and looking at the help page.

Lesson 1: Objects

An object is an assignment between a name and a value. You assign values to names using `<-` or `=`. The first assignment symbol consists of a `<` next to a dash `-` to make it look like an arrow.

```
x <- 5 #assign the value of 5 to a variable called x
# notice that this x is now in your global environment
x # print x
```

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

```
y = 10
y
```

```
## [1] 10
```

You can combine objects together as well which lets us do some basic math operations.

```
# create a new object called z that is equal to x*y
z <- x * y
#print z
z
```

```
## [1] 50
```

If you do not create an object, R will not save it to the global environment. If an object is not in the global environment and you try to reference it later, R will not know what you are referring to.

Math Operations

```
a <- 2+3
a
```

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

```
b<-4-5
b
```

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

```r
c<-4*2
c
```

```
## [1] 8
```

```r
d<-6/3
d
```

```
## [1] 2
```

```r
e<-7^2
e
```

```
## [1] 49
```

Vectors

You can create a vector (a list) of items in R.

```r
# create a vector of 1 through 10
vector1 <- 1:10
vector1
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

If we want specific items, we use the `c()` function and separate the items with a comma.

```r
vector2 <- c(1,3,5,7,9)
vector2
```

```
## [1] 1 3 5 7 9
```

Mathematical operations work on vectors too!

```r
vector2^2
```

```
## [1]  1  9 25 49 81
```

Classes

Objects in R have different classes. Check the class of a few objects we have already created:

```r
class(x)
```

```
## [1] "numeric"
```

```
class(vector1)
```

```
## [1] "integer"
```

There are other classes too!

```
# create a string
my_string <- "EC320 is my favorite class!"
class(my_string)
```

```
## [1] "character"
```

```
# logical class
class(2>3)
```

```
## [1] "logical"
```

What happens if we have a vector of characters and numbers?

```
char_vector <- c(1:5, "banana", "apple")
char_vector
```

```
## [1] "1"      "2"      "3"      "4"      "5"      "banana" "apple"
```

```
#cant use mathematical operations on characters
#char_vector^2 (note: this doesn't run; uncomment it and show them it doesn't run)
# why?? because the entire vector is a character class!
class(char_vector)
```

```
## [1] "character"
```

We will be working with data frame classes in this course, but we will cover that next week.

Functions

Functions are operations that can transform your created **object** in a ton of different ways. We have actually already used two functions, `c()` and `class()`. Here are a few other useful ones we will use today:

```
#print the first few objects in vector1
head(vector1)
```

```
## [1] 1 2 3 4 5 6
```

```
#print the first 2 objects in vector1
head(vector1, 2)
```

```
## [1] 1 2
```

```r
#print the last few objects in vector1
tail(vector1)
```

```
## [1]  5  6  7  8  9 10
```

```r
#print last two objects in vector1
tail(vector1, 2)
```

```
## [1]  9 10
```

```r
#find the mean of vector1
mean(vector1)
```

```
## [1] 5.5
```

```r
#median
median(vector1)
```

```
## [1] 5.5
```

```r
#standard deviation
sd(vector1)
```

```
## [1] 3.02765
```

```r
#Summary() prints summary stats
summary(vector1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    3.25    5.50    5.50    7.75   10.00
```

```r
min(vector1)
```

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

```r
max(vector1)
```

```
## [1] 10
```

Exercises

1. Create a new variable called `my.num` that contains 5 numbers
2. Multiply `my.num` by 3
3. Create a second variable called `my.char` that contains 5 character strings
4. Combine the two variables `my.num` and `my.char` into a variable called `both`combined‘
5. What is the length of `combined`? (Hint: use the `length()` function)
6. What class is `combined`? Why?
7. Divide `combined` by 3, what happens?

8. Create a vector with elements 1 2 3 4 5 6 and call it `a`
9. Create another vector with elements 10 20 30 40 50 and call it `b`
10. What happens if you try to add `a` and `b` together? Why?
11. Append the value 60 onto the vector `b` (hint: you can use the c() function)
12. Add `a` and `b` together
13. Multiply `a` and `b` together. Pay attention to how R performs operations on vectors of the same length.