Lesson 2: Getting cozy with R Markdown

Patrick Mathias

Why integrate your analysis and documentation in one place?

The short answer is that it will be easier for you to understand what you did and easier for anyone else to understand what you did when you analyzed your data. This aligns nicely with the principles of reproducible research and is arguably just as important for any analysis that occurs in a clinical laboratory for operational or test validation purposes. The analysis and the explanation of the analysis live in one place so if you or someone else signs off on the work, what was done is very clear.

The more philosophical answer to this question lies in the principles of literate programming, where code is written to align with the programmer's flow of thinking. This is expected to produce better code because the program is considering and writing out logic while they are writing the code. So the advantages lie in both communication of code to others, and that communication is expected to produce better programming (analysis of data in our case).

There is another advantage of using this framework with the tools we discuss below: the output that you generate from your analysis can be very flexible. You can choose to show others the code you ran for the analysis or you can show them only text, figures, and tables. You can produce a webpage, a pdf, a Word document, or even a set of slides from the same analysis or chunks of code.

Basics of knitr and rmarkdown

The theme of the course so far is "there's a package for that!" and this of course is no exception. The knitr package and closely related rmarkdown package were built to make it easier for users to generate reports with integrated R code. The package documentation is very detailed but the good news is that RStudio inherently utilizes knitr and rmarkdown to "knit" documents and allows for a simple, streamlined workflow to create these documents.

There are 3 components of a typical R Markdown document:

- header
- text
- code chunks

Header

The header includes metadata about the document that can help populate useful information such as title and author. This information is included in a YAML (originally *Yet Another Markup Language*, now *YAML Ain't Markup Language*) format that is pretty easy to read. For example, the header for this document is:

```
---
title: 'Lesson 2: Getting cozy with R Markdown'
author: "Patrick Mathias"
output: html_document
---
```

The output field dictates the output once the document is knit, and users can add other data such as the date or even parameters for a report.

Text

Text is written in whitespace sections using R Markdown syntax, which is a variant of a simple formatting language called markdown that makes it easy to format text using a plain text syntax. For example, asterisks can be used to *italicize* (`*italicize*`) or **bold** (`**bold**`) text and hyphens can be used to create bullet points: - point 1 - point 2 - point 3

```
- point 1
- point 2
- point 3
```

Code chunks

Interspersed within your text you can integrate "chunks" of R code, and each code chunk can be named. You can supply certain parameters to instruct R what to do with each code chunk. The formatting used to separate a code chunk from text uses a rarely utilized character called the backtick ‘ that typically can be found on the very top left of your keyboard. The formatting for a code chunk includes 3 backticks to open or close a chunk and curly brackets with the opening backticks to supply information about the chunk. Here is the general formatting, including the backticks and the curly braces that indicate the code should be evaluated in R:

````
```r
mean(c(10,20,30))
```


```
## [1] 20
```
````

And this is how the code chunk looks by default:

```r
mean(c(10,20,30))
```

There are shortcuts for adding chunks rather than typing out backticks: the `Insert` button near the top right of your script window or the `Ctrl+Alt+i`/`Command+Option+i`(Windows/Mac) shortcut.

In addition code can be integrated within text by using a single backtick to open and close the integrated code, and listing "r" at the beginning of the code (to indicate the language to be evaluated): 20.

Flexibility in reporting: types of knitr output

Under the hood, the knitting functionality in RStudio takes advantage of a universal document coverter called Pandoc that has considerable flexibility in producing different types of output. The 3 most common output formats are .html, .pdf, and Microsoft Word .docx, but there is additional flexibility in the document formatting. For example, rather than creating a pdf or html file in a typical text report format, you can create slides for a presentation.

There is additional functionality in RStudio that allows you to create an R Notebook, which is a useful variant of an R Markdown document. Traditionally you might put together an R Markdown document, with all its glorious text + code, and then knit the entire document to produce some output. The R Notebook is a special execution mode that allows you to run individual code chunks separately and interactively. This allows you to rapidly interact with your code and see the output without having to run all the code in the entire document. As with inserting a chunk, there are multiple options for running a chunk: the `Run` button near the top right of your script window or the `Ctrl+Shift+Enter`/`Command+Shift+Enter` (Windows/Mac) shortcut. Within a code chunk, if you just want to run an individual line of code, the `Ctrl+Enter`/`Command+Enter` (Windows/Mac) shortcut while run only the line your cursor is currently on.

Exercise 1

Let's use the built-in functionality in RStudio to create an R Markdown document. 1. Add a file by selecting the add file button on the top left of your screen 1. Select R Markdown... as the file type 1. Title the document "Sample R Markdown Document" and select OK 1. Put the cursor in the "cars" code chunk (should be the 2nd chunk) and hit `Ctrl+Shift+Enter/Command+Shift+Enter`. What happened? 1. Insert a code chunk under the cars code chunk by using the `Ctrl+Alt+i/Command+Option+i`(Windows/Mac) shortcut 1. Create output for the first lines of the cars data frame using the `head(cars)` command and execute the code chunk

RStudio sets up the document to be run as an R Notebook so you can interactively run chunks separately and immediately view the output.

RStudio also already provides you with an outline of a useful document, including interspersed code chunks. The header is completed based on the data that was entered into the document creation wizard. The first code chunk below the header is a useful practice to adopt: use your first code chunk as a setup chunk to set output options and load packages you will use in the rest of the document. The `knitr::opts_chunk$set(echo = TRUE)` command in the setup chunk tells R to display (or echo) the source code you write in your output document. A detailed list of various options can be found under the R Markdown cheatsheet here: https://www.rstudio.com/resources/cheatsheets/.

Now let's knit this file and create some output.

Exercise 2

1. Click the **Knit** button
2. You are being prompted to save the .Rmd file. Choose the "src" folder of your project and name the file sample_markdown_document
3. RStudio should produce output in .html format and display
4. Click the Open in Browser window and the same output should open in your default internet browser
5. If you find the folder you saved the .Rmd file there should also be a .html file you can open as well
6. Now, instead of hitting the **Knit** button, select the down arrow adjacent to it and click Knit to PDF
7. Repeat the previous step but knit to a Word document

The add file options also allow you to create a presentation in R Markdown. This can be a handy alternative to Powerpoint, especially if you want to share code and/or many figures within a presentation. You can find more information about these presentations and the syntax used to set up slides at the RStudio site on Authoring R Presentations.

Creating an R Markdown document that integrates best practices in reproducible research

Let's review the components of a project according to the take home points from Lesson 1: - a directory for each project - a folder structure the separates raw data, output, documents, etc. - clear file naming - code written with a style convention - code portability/reproducibility with help from here and checkpoint packages

How can we integrate R Markdown documents within the context of these requirements? We already created a project and associated directories for this course so most of our work is done. Now we are going to add our R Markdown file to the project and set it up so we can be efficient and reproducible.

Exercise 3

1. Close the RStudio window that has your "sample-project-structure" project open (refer to the upper right of your RStudio window to see which active project is open).
2. Find your RStudio window with your "MSACL-intermediate-R-course" project (look in upper right). If you cannot find it, within RStudio, click the arrow to the right of the project name (it will say Project: (None) if you are not in a project) and select your "MSACL-intermediate-R-course" project.
3. From within the course directory ("MSACL-intermediate-R-course"), create a new directory called "coursework" & set your working directory to there
4. Load the here package from within your console and use the `set_here()` command to set your "coursework" directory to be the primary point of reference for your project. (This creates an empty file named ".here", which is the first thing the here package looks for when trying to set its point of reference.)
5. Restart your R session by navigating to the Session menu and selecting Restart R. This is required to reinitialize the here package and point it to the .here file you just created in your "coursework" directory
6. Create the directory structure we discussed in the last lesson, **with the exception of the data directory**
7. Go to your operating system and copy the "class_data" directory you unzipped prior to the course (per the pre-course instructions) into your "coursework" directory
8. At this point we will want to include our individual lesson .Rmd files in the "src" directory. For now, copy your lesson3.Rmd from the lesson3 folder into your "src" directory under coursework.

These steps have set up your directory structure for future lessons. We have pre-made lesson files for future lessons, but it is also helpful to create an independent R Markdown file for any additional code you might want to write outside of the lesson. Plus, we want you to get some experience building your own R Markdown file within your project environment from scratch.

Exercise 4

1. Before we jump into creating our R Markdown file, we need to create a file that will allow us to run checkpoint together with knitr - click the Add File button and select "R Script"
2. Add the following code to your R script (minus the code chunk delimiters)

```
library("formatR")
library("htmltools")
library("caTools")
library("bitops")
library("base64enc")
library("rprojroot")
library("rmarkdown")
library("evaluate")
library("stringi")
```

3. Save as "knitr-packages.R" in the src folder of your project (explanation of why you did this below)
4. Now add a new R Markdown file, title it "Data Science 201", and leave as the default .html ouptut selection
5. Go ahead and save the .Rmd by clicking the Save button (disk button) or selecing Save under the File menu: save the file to your src folder within your project. This will be an extra place to write code outside of the lesson files we provide. Name according to the principles we disucssed in lesson 1 (think of it as a scratchpad).
6. In the setup code chunk for the document, load the checkpoint package
7. Add the checkpoint command to load packages from today: `checkpoint("2018-01-21")`
8. Load the following packages in the setup chunk: tidyverse, readxl, lubridate, janitor, lattice, magrittr, modelr, xml2, jsonlite

9. Save the file and then go ahead and run your setup chunk

So why did you have to go through the first 3 steps before starting your R Markdown document? When RStudio knits a document, it calls certain packages behind the scenes to create the document. However, when you are using checkpoint, the package only looks for packages within your project folders and does not see those hidden RStudio folders. So adding knitr-packages.R is a workaround to allow checkpoint to see those packages and knit the document.

The R Markdown document you created will be your scratchpad if you want to experiment with differnt functions or explore data in our dataset outside of the context of the structured lessons.

Now we're ready to jump into lesson 3!

Summary

- Integrating code and documentation in one place produces clearer, more reproducible code
- RStudio provides useful built-in functionality for "knitting" documents into a variety of output formats
- R Markdown documents can be integrated within a recommended project structure to create a reproducible analysis