# R Training

Lesson 6

*Stefanie Molin*

*April 19, 2017*

## I. RMarkdown

Have you wondered how I made all these pretty and super-helpful training notes? Surely you thought I spent multiple hours copying, pasting, and formatting, but, actually, I made everything using R; I never had to leave RStudio! RMarkdown allows you to run repeatable analysis and generate high-quality reports (PDFs, HTML, slides, etc.) all in one place.

### A. Setup

Using RMarkdown requires a little more setup than the other packages we have seen. You have already installed `knitr` and `rmarkdown` in the first lesson, but we also need to download LaTeX for producing PDFs. You can download LaTeX here: https://miktex.org/download (it can be a very lengthy process). If you now open a new RMarkdown (.Rmd) document in RStudio, it should have a "Knit" button which you can use to change the output type and to compile the document; note that you won't need to call `library(knitr)` or `library(rmarkdown)` to do so. *If you are missing any packages at this point, RStudio will prompt you and ask if you want to install them.*

### B. Elements of RMarkdown Documents

When you create your first RMarkdown document, RStudio provides you with some sample code to generate a mock RMarkdown document and give you an idea of the capabilites. There are 5 elements I want to go over:

#### 1. YAML header

At the top of the document, between 2 sets of `"---"`, you will have a YAML header which `knitr` uses for the header of the file. YAML is a markup language of key-value pairs like JSON, but easier to read. Here is the header for this lesson's notes.

```
---
title: "R Training"
subtitle: "Lesson 6"
author: "Stefanie Molin"
date: "`r format(Sys.time(), '%B %d, %Y')`"
output: pdf_document
---
```

It's pretty straightforward right? The entry for `date` is R inline code; each time I compile this document, it calculates the date rather than using a fixed value.

**2. R inline code**

As you saw in the YAML above, you can write R code inline as:

`` `r code_goes_here` ``

and that text will be replaced by the evaluation of that piece of code in the document. For example, I put

`` `r Sys.time()` ``

at the end of this sentence: the date-time is 2017-04-19 08:47:03.

**3. R chunks**

Inline code is great for evaluating short code snippets, or something where the implementation isn't really important; however, most of what you are documenting in RMarkdown is more than one line of code. For this reason, we can use R chunks. This allows us to put multiple lines of code in a code block and set multiple options for each chunk. A chunk looks like this:

~~~
```{r chunkName, echo=FALSE}
print('Hello World!')
```
~~~

Some useful chunk options include:

- `echo`: whether or not to display the code in the document (boolean)
- `eval`: whether or not to evaluate the code in the document (boolean)
- `warning`: whether or not to print warnings to the document (boolean)
- `error`: whether or not to let the document compile if there are errors raised in that chunk (boolean)
- `message`: whether or not to print messages to the document (boolean)
- `fig.height`: change height of graphs in document
- `comment`: what to print in front of the results of the evaluated code (defaults to "##", `NA` will print without anything in front)

These can all be set on the document level or on the chunk level. Chunk-level settings will override global settings. Multiple settings are comma-separated. You can also name chunks (optional) to easily repeat code throughout the document without having to copy and paste; names (if specified) will be the first word after `r` and inside `{}`.

**4. Text**

This one is pretty simple, just type the text you want, where you want it into the script. There is no need to put quotes around the text, unless you want them to show up in your output.

**5. Special symbols for formatting**

For any special formatting you want to do in your documents, you will need to use special characters so RMarkdown knows how to format the text. The below are a few that you will probably need, but more can be found here: https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf (along with RMarkdown tips in general).

- **italics**
- ****bold****
- *: unordered list
- 1.: ordered list
- ##: header level 2 (levels 1-6 available with 6 being the smallest)

## C. Additional Utilities

The aforementioned information gives you all the tools you need to get started, but of course RMarkdown is capable of more than what is presented here. A few other features you will find useful are formatted tables and the ability to turn R scripts into RMarkdown documents without having to rewrite everything.

### 1. Formatted Tables with `knitr::kable()`

As you have seen from prior lessons, the default output for tables isn't too pretty. If you want to make your tables look better, you can use `kable()`. The below example is taken from the RStudio guide to RMarkdown, which you may find helpful for further info: http://rmarkdown.rstudio.com/lesson-1.html

Printed dataframe **without** using `knitr::kable()`:

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
```

Using `knitr::kable()`:

| | mpg | cyl | disp | hp | drat | wt | qsec | vs | am | gear | carb |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Mazda RX4 | 21.0 | 6 | 160 | 110 | 3.90 | 2.620 | 16.46 | 0 | 1 | 4 | 4 |
| Mazda RX4 Wag | 21.0 | 6 | 160 | 110 | 3.90 | 2.875 | 17.02 | 0 | 1 | 4 | 4 |
| Datsun 710 | 22.8 | 4 | 108 | 93 | 3.85 | 2.320 | 18.61 | 1 | 1 | 4 | 1 |
| Hornet 4 Drive | 21.4 | 6 | 258 | 110 | 3.08 | 3.215 | 19.44 | 1 | 0 | 3 | 1 |
| Hornet Sportabout | 18.7 | 8 | 360 | 175 | 3.15 | 3.440 | 17.02 | 0 | 0 | 3 | 2 |

### 2. Turning R scripts into RMarkdown

Rather than rewriting your R scripts into RMarkdown, you can use the `spin()` or `stitch()` functions from `knitr` or `rmarkdown::render()` to do this for you. Note that comments in your script need to be in `roxygen2` format, i.e. `#'` before each line. More info here: http://rmarkdown.rstudio.com/articles_report_from_r_script.html

```r
# not run

# using render, make a PDF (this looks the best)
rmarkdown::render("analysis.R", "pdf_document")

# using stitch makes a PDF by default
knitr::stitch("analysis.R")

# using spin makes a HTML by default
knitr::spin("analysis.R")
```

### 3. Turning RMarkdown into an R script

The `purl()` function from `knitr` pulls out all R chunks from a RMarkdown file and creates an R script.

```r
# not run
knitr::purl("R_training.Rmd")
```

## II. Exercises

Let's do some exercises to practice what you have learned and review prior lessons.

1. Create a new RMarkdown PDF document in RStudio and click the "Knit" button. Let RStudio install any missing LaTeX packages. Once you are able to generate a PDF successfully, try `html_document` and `slidy_presentation` output formats by clicking the drop down on the "Knit" button or changing the value for `output` in the YAML header.

*No written solution.*

2. Add `roxygen2` comments to one of your scripts from the exercises of the previous lessons and using the method of your choice, generate a RMarkdown HTML.

```r
library(rmarkdown)

# this script already has roxygen2 comments
render("R scripts/retail_chord_diagram.R")
```

3. Prepare a well-formatted PDF document with a specific analysis embedded. You should query Vertica, use `dplyr` and `ggplot2` to produce pivoted data and a graph. Include explanations of the analysis and how you did it. Keep in mind that you don't need all R chunks to be visible; some that are strictly for setup don't have to be in the document. RMarkdown also can't read variables you have defined in your environment unless they are created in when compiling the document; therefore, you will need to use `getPass::getPass()` to prompt for a password.

   If you can't think of an analysis, look at the percent of transactions that are coming in without an internal product ID by day for [redacted] (last 30 days).

*For a possible solution, see the file "R Training Exercise 6.3 Solution.Rmd" covering* [redacted].