Principles of Data Visualization in R Workshop: Introduction to RMarkdown

Willy Chen

Workshop 6

Click here to return to the menu

RMarkdown

Download my stylesheet here

RMarkdown is file type specific to RStudio that helps you automate document processes. In today's workshop, we will focus on using RMarkdown for PDF and html. Note that RMarkdown can also be used to output MS Word, MS Powerpoint, etc. Markdown (`.md`) is a common way to call text files that describes a document and can be read differently by different compilers. RMarkdown (`.rmd`) is the markdown system specifically designed for RStudio.

When you create a new RMarkdown file in RStudio, it will be created as an example RMarkdown and contains the essential elements in RMarkdown. These elements are `YAML`, `R chunk`, `R inline`, `texts`.

YAML

YAML is always at the beginning of the RMarkdown. It is the top section sectioned off by `---`. Inside your YAML, you can specify the title of this document, the author, the date, the file type, as well as other secondary options like `CSS` for `html` and `preamlbles` for `PDF`. For example, in the RMarkdown I use to make this workshop, my YAML is:

```
---
title: "Principles of Data Visualization in R Workshop: Introduction to RMarkdown"
author: "Willy Chen"
date: "Workshop 6"
output:
  html_document:
    toc: true
    toc_float: true
    css: eye.css
    theme: readable
---
```

`title` specifies the name of this document and `author` specifies the author's name. I put "Workshop 6" for `date` because I did not need it to sepcify a date, but rather just show the index of the workshop. `output` specifies which kind of document I want R to make this `.rmd` into. When this YAML is read by R, it takes these values and outputs the document desired.

Code Chunks

A code chunk is similar to YAML, except it is sectioned off by ` ```{r} ` and ` ``` `. Inside your code chunk, you will have access to your R environemnt. This means you can write a bunch to R code to be run at that specific place. If there is any output from the code, it will be written in the document. This makes it incredibly convenient to make automated reports that is designed and styled in your specified way. For example, I want to output, at this specific spot, a table of the data we used last time, I can write:

```r
require(tidyverse)
require(kableExtra)
data<-read_csv("https://willythewoo.github.io/WillyTheWoo/workshop/data/workshop_data3.csv")
data[1:5,]%>%kable("latex")%>%kable_styling("striped")
```

workshop	age	county	social	cohort	gpa
Yes	24	Orange County, CA	SGGSAC	Anthropology	3.806362
Yes	23	Franklin county, OH	SGGSAC	Economics	3.229651
Yes	23	Franklin county, OH	SGGSAC	Economics	3.338812
Maybe	24	Cook county, IL	SGGSAC	Sociology	3.582800
Yes	23	Knoxville, Tennesse	SGGSAC	Political Science	3.630303

To execute any single line of code in a chunk, you will click on that line and press `command/control + return/enter` on your keyboard. To execrute the entire chunk, you will click on that chunk and press `command/control + shift + return/enter`. To execute the entire document, you will click the `knit` button positioned right below your file tab. When you `knit` a document, your RMarkdown is automatically saved.

Chunk options

When you execute some R code, you generally get 4 types of outputs: `echo`, `warning`, `message`, and `results`. `echo` simply echos your code, you can see your `echo` in the console everytime you execute some code. `warning` and `message` comes from the functions you execute. These are generally things you may want to pay attention to even though they do not affect the output. `results` are the text outputs of your code.

When writing in RMarkdown, you can specify whether you want to include each type of output next to the `r` at the beginning of your chunk. The default is everything is included. If you do not want a specific type of output, you can change them by specifying that type to be `FALSE`. If you are dealing with results though, your specification is only between `asis` and `hide`.

There are other options such as `eval`, `include`, `fig.width`, etc. `eval=FALSE` means this chunk will not be evaluated, but you will still get the echo. `include=FALSE` neglects the existence of that chunk. `fig.__` specify the dimensions of your output if it is a picture.

Global options

Sometimes you might want to specify a specific options for all your chunks. This is when you can use global options. To do global options, you need to create a chunk at the beginning of the document after your `YAML` in the following way:

```r
knitr::opts_chunk$set(echo = TRUE)
```

This chunk, if evaluated, would make all of the chunks `echo=TRUE`. Local specifications will override the global chunk options so you can still specify differently for specific chunks. In addition, if you put pure texts next to the `r` at the beginning of each chunks, you can name that chunk. Naming is especially useful if you have many chunks in a document and your markdown is not knitting because you can see exactly which chunk is giving you troubles.

Inline R Code

If you have a really short code to run and you don't need R to echo it, you can do this inline. This means R will evaluate your code with out it being a chunk and can be much faster for referencing and helps with automation. To use inline code, you will write

```
`r your code here`
```

For example, if I want to reference the second row and thrid collumn of the data frame from earlier, I can write

```
`r data[2,3]`
```

and get Franklin county, OH. You can also use inline code to do simple calculations like

```
`r 7+2`
```

and we get 9

Texts and sectioning

There are acouple of symbols that have specifiv functions in RMArkdown: `#`, `*`, `1.`, `_`, `-`

`#` Let's you declare a section. To do so, you need to make sure there is an empty line, a `#` in the next line, followed by a space and the title you want to call the section. `##` will create a second level of section under the original section. By default, you can have at most 7 levels of sections.

`*` can be used as a bullet point and create an unordered list. It needs to be preceded by an empty line and have a space after it aswell. If you want sub-bullet points, you can use `-` in the next line and indented to be after the `*`.

`1.` A number with a period folloing the same spacing rule will create an ordered list. The sub list item rule still applies.

If you wrap your texts with `*` or `_`, `R` will *italicise* those texts. If you wrap the texts with `**` or `__`, R will make the texts **bold**.

To create a hyper link, you can use `[texts](link)`. If you want to insert a picture, you can use `![](link)`. If you are outputting an html/pdf document, you can write in html/LaTex code as well.

For more symbols like these, you can check out the RMarkdown cheat sheet

PDF document

To make your RMarkdown output a pdf file, you need to specify `output: pdf_document` in your `YAML` header. However, this does require the use of the LaTex language. If you have installed LaTex on your computer before, you can skip next step.

Installing tinytex

To install the LaTex language on your computer, execute the following code chunk

```
install.packages('tinytex')
tinytex::install_tinytex()
```

Once tinytex is installed on your computer, as long as you specify `output: pdf_document`, your RMarkdown will a pdf document. If you need to specify any packages in your LaTex preamble, you need to add it to your `YAML` by doing:

```
---
title: "Title"
author: "Me"
header-includes:
   - \usepackage{packagename}
output:
    pdf_document
---
```

HTML document

To make your RMarkdown output a pdf file, you need to specify `output: html_document` in your `YAML` header. There are many seconndary options that you can specitfy for your html. For example, to create this workshop, I had to make my `YAML`:

```
---
title: "Principles of Data Visualization in R Workshop: Introduction to RMarkdown"
author: "Willy Chen"
date: "Workshop 6"
output:
  html_document:
    toc: true
    toc_float: true
    css: eye.css
    theme: readable
---
```

`toc: true` tells R I want there to be a table of contents. `toc_float:true` make that toc float around. `css` specifies a styling sheet for my webpage. `theme` specifies a theme for the webpage. You can find more secondary options and their values by googling.

Conluding remarks

RMarkdown is super useful both for storing your codes and making documents. When it comes to the usage of RMarkdown, honestly, the sky is the limit. You can check out more of my examples in R on my website. For the last workshop, we will talk about some coding basics that can help you out when you need to specific things in your report/document.

Click here to continue to the next workshop: Coding basics

Click here to return to the menu