Beautifying RMarkdown

Diego Dulanto

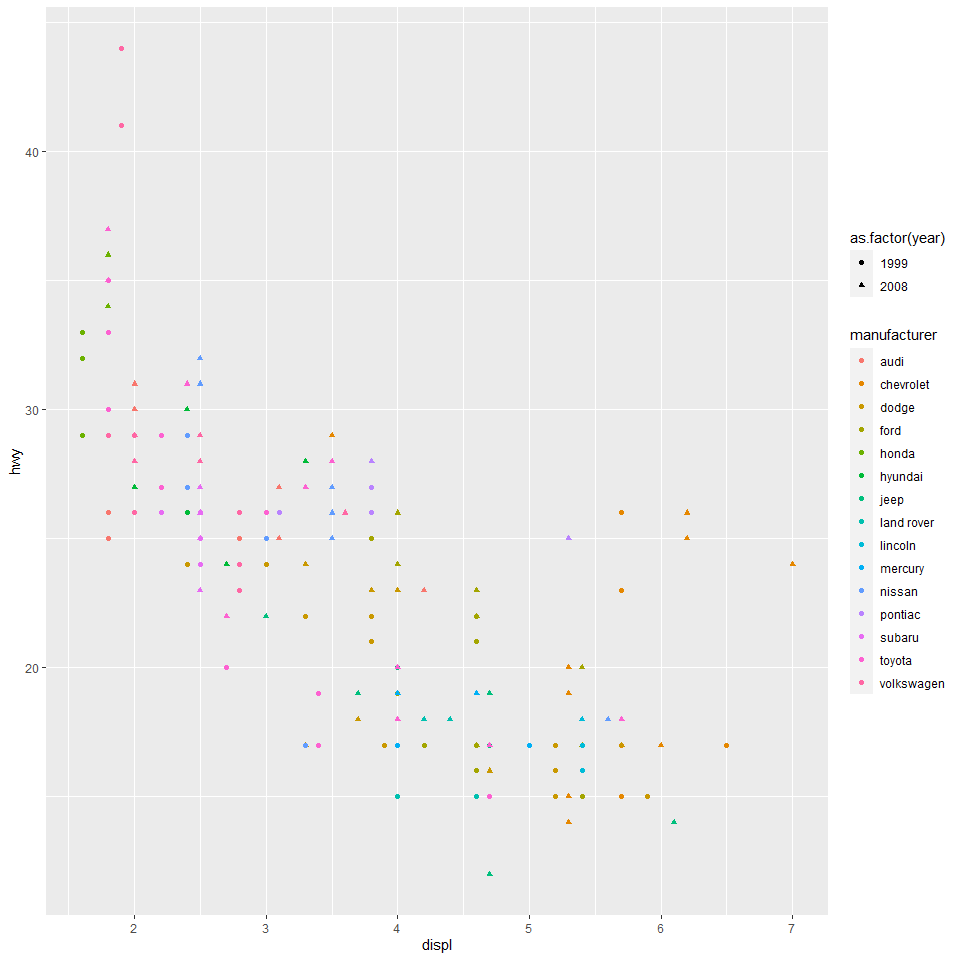
Mar 13 2021

Table of Contents

## The Surprising Truth about mpg

### mpg: The Figure

Add a code chunk that uses the ggplot2::mpg dataset to plot displ against hwy (highway mpg), with points colored by manufacturer and shaped by year.



The overall relationship between highway and displ seems to be negative. The chevrolet manufacturer seems to have an outlier for the year 2008.

### mpg: The Table

Add a second code chunk that gives a summary table with the **mean** and **sd** for displ and hwy for each year.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| year | displ\_mean | displ\_sd | hwy\_mean | hwy\_sd |
| 1999 | 3.281197 | 1.260058 | 23.42735 | 6.084628 |
| 2008 | 3.662393 | 1.300645 | 23.45299 | 5.847923 |

The mean and standard deviation for displ increased from 1999 to 2008. The mean highway had a slight increase and the standard deviation a slight decrease from 1999-2008.

Around each piece of output, add a brief description.

We will examine how various options change the rendered HTML output.

## Resources

Here are two general RMarkdown how-to books that are chock-full of great info:

* <https://bookdown.org/yihui/rmarkdown/>
* <https://bookdown.org/yihui/rmarkdown-cookbook/>

## Themes

Themes control the general styling of text, headings, etc. RMarkdown comes with several built-in theme options:

* default, cerulean, journal, flatly, darkly, readable, spacelab, united, cosmo, lumen, paper, sandstone, simplex, yeti

You can see how these (and other) themes look [on this page](https://www.datadreaming.org/post/r-markdown-theme-gallery/)

Several packages provide additional themes. Here are a few themes I like:

* hrbrthemes::ipsum
* prettydoc::leonids
* rmdformats::material
* tufte::tufte\_html

Change the theme for the document in the YAML header and try out some different themes to see what they look like.

## Code Highlighting

In the RStudio script editor, code syntax is highlighted to make it easier to follow. You can enable this in your knitted output by adding the highlight option under the output format.

Available highlighting color schemes include:

* default, tango, pygments, kate, monochrome, espresso, zenburn, haddock, breezedark, textmate, null
* I like haddock and zenburn
* null is the same as omitting highlight (disables it)

Add the highlight option to the document YAML and try out different styles.

## Code Folding

So far in class, we have always shown both your code and output. This can make it easy to review the code, but makes it hard to read the actual report.

You can have the best of both worlds with the code\_folding option.

Add the option code\_folding under your html\_document output format and set it to hide. Knit and see what happens.

You can set all of the code chunks to show initially but still by hideable by setting the option to show.

## Table of Contents

You can add a table of contents sidebar to your HTML output by adding the toc: true option under your html\_document output format.

You can set the table of contents to float with the page as the reader scrolls with the toc\_float: true option. You can control how many levels of headings (#, ##, ###, etc.) to include in the table of contents with the toc\_depth option.

Add a floating table of contents to your document and have it show 3 levels of headings. Change the depth to 2 and knit again to see the difference.

## Running R Code in YAML

You can include R code in your YAML that will be evaluated when you knit the document. Do that by writing (in quotes), for example:

* ‘3’ where 1 + 2 is the R code you want to run
* (if you are looking at this in the knit HTML, that’s '`r 1 + 2`')

Change the date field in YAML to be dynamically generated with today’s date by running this R expression in the date field:

format(Sys.Date(), '%b %d %Y')

**Note!** For some parts of your YAML, this syntax doesn’t work. In those cases, use this alternative format:

* '!expr <expression>'

See the section on Word document output below for an example. (I don’t really get when you have to use one versus the other, either. You can default to the '`r `' format and then switch if that doesn’t work.)

## Chunk Options

YAML options let you control output features for the whole document. You can also specify options to control output of each code chunk. For example, you can control the size of images or set a chunk not to run.

Set chunk options like this:

{r chunk name, option=TRUE, option="choice"}

We will practice a few useful chunk options. For more details, see <https://yihui.org/knitr/options/>.

### Chunk Names

You can give your chunks names. This is helpful if you want to keep track of where in your document the knitting process is as it runs (useful if you have a long script that takes some time to run). Anything after the r in the chunk heading before the first comma will be the chunk name.

Chunk names have to be unique (not duplicated) by default, so be careful when copying and pasting chunks in your document.

You should always have a chunk at the top of your script called setup. In this script, you should have:

* All of your library() calls or other package checks
* Other options or “setup” bits (e.g., vectors of item names for scales).
* Potentially, put your data import calls here.

RStudio treats the setup chunk a little specially. When you run chunks interactively (by clicking the Run buttons or typing Ctrl+Enter or Cmd+Return), RStudio will run the setup chunk first.

### eval, echo, include

These chunk options control whether a chunk is run and whether the code and output are shown.

* eval: Should the code be run (evaluated)? Set to FALSE to not run the code
  + e.g., use this if you include an install.packages() call in your script
* echo: Should the code be included? Set to FALSE to omit the code.
  + Note that if you have code folding enabled, echo=FALSE will still remove the code entirely.
* include: Should the output be included. Set to FALSE to omit the output
  + This is useful, for example, to hide output of library() calls from your setup chunk.

Try these out!

### fig.width, fig.height

You can control height and width of figures in a chunk (in inches) with fig.width and fig.height.

Change your plot chunk’s figure dimensions and see what happens.

### Graphic Device and DPI

By default, RMarkdown will save images as PNG (a bitmap format). You can change this to use SVG or PDF using the dev option: dev="svglite" or dev="pdf"

For bitmap devices, you can specify the resolution (DPI) for images with the dpi argument. For example, dpi=300. This mostly doesn’t matter for HTML output, but if you output to Word or PDF, you will want to specify a DPI of 300.

Try changing the graphics device for your plot chunk to "svglite".

### Caching

When you click the Knit button, R will run your entire script in a new session. That’s usually good! The results are fresh and new each time! But if you have a long script with slow operations, you might want to **cache** the results of chunks so that they don’t have to be re-run unless the code in the chunk changes.

Do that by setting the cache=TRUE option. I’ve already done that globally for this document (see the setup chunk).

Be careful with caching! This approach only checks if the code in a specific chunk has changed. It won’t work correctly if, e.g., you have changed a data file that is read in or if you change a variable from a previous chunk that the current chunk relies on.

You almost always want to set caching on specific chunks, rather than globally. We will explore some more robust caching methods in a future class.

### Allowing Errors

Normally, if your R code produces an error, knitting will stop. If you want to allow knitting to continue even if there’s an error, set the error=TRUE option.

Try it out! Change this chunk’s eval=FALSE to error=TRUE.

stop("This is a scary error!")

### Collapsing Results Together

Normally, if both echo and include are TRUE, if a chunk has multiple outputs, the knitted document will alternate code 1, output 1, code 2, output 2, etc. If you want to have to show all of the code together, then all of the outputs together, set the collapse chunk option to TRUE.

### result=‘asis’

Some packages generate output text that is already in HMTL, LaTeX, or Markdown format and doesn’t need any more processing. To render such output, set the results option to 'asis'.

Try it out! Add results='asis' to this chunk and see how the output changes.

cat("$$  
\\operatorname{mpg} = \\alpha + \\beta\_{1}(\\operatorname{cyl}) + \\beta\_{2}(\\operatorname{disp}) + \\epsilon  
$$")

## $$  
## \operatorname{mpg} = \alpha + \beta\_{1}(\operatorname{cyl}) + \beta\_{2}(\operatorname{disp}) + \epsilon  
## $$

### Global vs Specific Chunk Options

You can set an option for all the chunks in a document using the knitr::opts\_chunk$set() function. See the setup chunk at the top of the document for an example.

If you set a chunk both globally and on a specific chunk, the specific setting will override the global setting for that chunk.

## A Few More Awesome Things

Let’s look at a few more awesome RMarkdown features we can explore.

### params

You can set parameters for your document that you can call later. Do this by adding a params option to the first-indentation level of your document YAML. Add parameters as options indented under params like this:

title: "Your document title"  
params:  
 manufacturer: "audi"

You can refer to a parameter with the code params$paramter (e.g., params$manufacturer).

Try it! Add manufacturer: "audi" as a parameter to your document. Then, filter the mpg data to only show cars with this manufacturer before creating the plot.

### Tables

By default, if you just print a data frame, it will be formatted in your document the same way it is in the R console. This isn’t ideal.

You can format the document in a nicer way with the knitr::kable() function. For very long tables in HTML output, the DT::datatable() function is better.

We will also explore more advanced table design packages in a future class.

Try it! Change your table output above to use the knitr::kable() function. Then, knit and see the difference.

### Tabbed Results

If you have a long document, it can be hard to navigate. One way you can help organize your document is with tabs.

You can specify a section to be tabbed by adding {.tabset} at the end of the heading line. Then, all of the sub-headings on the next level below that one will be formatted as tabs.

### Hidden <details>

You can hide a secontion of your document behind a clickable button (like code folding) with the HTML <details> tag. When something is inside a <details> </details> block, it won’t show until the button is clicked.

You can add a title describing what’s behind the button with the <summary> tag.

For example:

<details>  
 <summary>This is the title of the hidden code.</summary>   
 This text doesn't show until the button is clicked.  
</details>

One really useful use of <details> is to provide the information about your computer system and the versions of packages you used at the end of your document with devtools::session\_info().

Try it! Add a <details> block to the end of the document that contains a code chunk with devtools::session\_info() in it. Provide an appropriate <summary> for the <details>.

### Other Child Scripts and Documents

You can offload some parts of your script to different documents. For example, you could move your data cleaning steps to a separate .R document. You could also move different chapters of your document (e.g., of your thesis) to separate .Rmd documents. We will explore this in a future class.

## Word documents

We are usally working with HTML documents. HTML is a great format because it is small and handles page formatting for you. It’s a great default format, especially if readers will read on screen.

Sometimes you might want to work with Word, though. For example, I often write my main document in Word, but write the methods and results sections and tables and figures with RMarkdown. Then, I copy-paste those in one go into the Word document.

Let’s try out some Word knitting options.

Add word\_document: default as an output format for your document and knit it. Look at the resulting Word file.

### Template Files

The default Word template styles used are pretty ugly in my opinion. You can provide RMarkdown with a different template file.

Place the reference\_docx.docx file in your repository in a folder called "templates". Then

### Page Breaks

In HTML, you don’t need to worry about page breaks. In Word or PDF, you might want to force a page break so that some content (e.g., the next table) starts on a new page.

To do that, add \newpage to your Markdown text on its own line (with a blank line before and after).

Try it! Add a page break between your plot and your table output. Then, knit and see what happens.

### Options Based on Output Format

You might want to change your chunk options based on your output format. For example, you might want to include folded code chunks if your document is knit to HTML but omit them if going to Word (where code folding doesn’t exist).

To do that, you can use the knitr::is\_html\_output() or knitr::pandoc\_to() functions. knitr::is\_html\_output() will return TRUE if knitting to an HTML-like format (e.g., HTML or Markdown) and FALSE otherwise. knitr::pandoc\_to() takes a character vector of file extensions (e.g., "docx" for Word) and checks if the document is knitted to that format.

Try it! Change the echo option for your chunks to be echo=knitr::is\_html\_output(), then knit to Word and see the result!