**Introduction**

If the [practical tips for R Markdown](https://jozefhajnala.gitlab.io/r/r909-rmarkdown-tips/) post we talked briefly about how we can easily create professional reports directly from R scripts, without the need for converting them manually to Rmd and creating code chunks. In this one, we will provide useful tips on advanced options for styling, using themes and producing light-weight HTML reports directly from R scripts. We will also provide a repository with example R script and rendering code to get different styled and sized outputs easily.

**Contents**

1. [Creating reports directly from R scripts](https://jozefhajnala.gitlab.io/r/r913-spin-with-style/#creating-reports-directly-from-r-scripts)
2. [Using knitr’s spin directly](https://jozefhajnala.gitlab.io/r/r913-spin-with-style/#using-knitrs-spin-directly)
3. [Using rmarkdown’s render](https://jozefhajnala.gitlab.io/r/r913-spin-with-style/#using-rmarkdowns-render)
4. [TL;DR: Just show me the examples](https://jozefhajnala.gitlab.io/r/r913-spin-with-style/#tldr-just-show-me-the-examples)
5. [References](https://jozefhajnala.gitlab.io/r/r913-spin-with-style/#references)

**Creating reports directly from R scripts**

For an introduction on creating nice reports directly from R scripts, look into the [relevant section](https://jozefhajnala.gitlab.io/r/r909-rmarkdown-tips/#creating-beautiful-multi-format-reports-directly-from-r-scripts) of the previous blog post. In one sentence, we can just call one of the following:

# with knitr directly

knitr::spin("path-to-r-script.R")

# or with rmarkdown

rmakdown::render("path-to-r-script.R")

to create a report from an R script directly. Both spin() and render() provide a default style that will be used to render an R script to html. The same is true from RStudio’s built-in File -> Compile report... functionality, which will call render() in the background when used.

We might, however, be interested in using different styles other than the default one when rendering our R scripts into HTML reports, and there are multiple ways to achieve this.

**Including styles the quick, dirty and risky way**

The fastest way to include a custom css stored in a file is to simply include a line like the following at the beginning of the R script that we are using spin() on:

'#

This simple approach however has many caveats, as the line is just inserted into the body of the document within a paragraph, completely oblivious to what else was inserted. Unless there is a very good reason, we should use one of the safer and more robust approaches mentioned below.

**Using knitr’s spin directly**

**Under the spin’s hood**

Under the hood, spin() calls knit2html(), which passes many useful arguments to markdownToHTML(), the function that actually converts a markdown file to the final HTML format. Unfortunately, many of those useful arguments are not exposed via spin().

Bearing this in mind, we have a few ways to access and provide them with the desired values:

1. Changing the options that govern the default values and just call spin() as before
2. Perform the spinning in 2 steps

**Changing the options that govern the default values and just call spin() as before**

As mentioned above, spin() does not expose the arguments of markdownToHTML() directly, so what happens in practice is that the default values for those arguments are used when spin() is called. Some of the interesting arguments are by default selected in the following way:

options = getOption("markdown.HTML.options"),

extensions = getOption("markdown.extensions")

stylesheet = getOption("markdown.HTML.stylesheet")

template = getOption("markdown.HTML.template")

Let’s have a look at some interesting default options’ values:

library(markdown)

options()[c(

"markdown.HTML.options",

"markdown.extensions",

"markdown.HTML.stylesheet"

)]

## $markdown.HTML.options

## [1] "use\_xhtml" "smartypants" "base64\_images" "mathjax"

## [5] "highlight\_code"

##

## $markdown.extensions

## [1] "no\_intra\_emphasis" "tables" "fenced\_code"

## [4] "autolink" "strikethrough" "lax\_spacing"

## [7] "space\_headers" "superscript" "latex\_math"

##

## $markdown.HTML.stylesheet

## [1] "/usr/local/lib/R/site-library/markdown/resources/markdown.css"

If we want to keep the spinning in one step, we can simply update those options before calling spin (and ideally change them back afterwards). For a somewhat minimalistic HTML output still keeping images self-contained, we can do:

options(

markdown.extensions = "fenced\_code",

markdown.HTML.options = "base64\_images",

markdown.HTML.stylesheet = "{}"

)

knitr::spin("spin\_exaple.R")

To use a custom css stylesheet instead of the one [provided by default](https://github.com/rstudio/markdown/blob/master/inst/resources/markdown.css) with the markdown package:

options(markdown.HTML.stylesheet = "path\_to\_custom.css")

knitr::spin("path-to-r-script.R")

**Perform the report creation in 2 steps**

The method above works but can seem quite workaround-ish. The method that could be considered more proper is to actually split the production of the final output into 2 steps:

1. Generate an intermediate .Rmd file via spin(), using spin(..., knit = FALSE)
2. Run knit2html() on the created .Rmd file with the desired options directly specified as arguments

This allows us to provide additional arguments extensions, stylesheet, header, template and encoding in the second step, instead of relying on the changed options to be passed as defaults.

The below example will embed styles present in path\_to\_custom.css into the resulting HTML:

# Creates the intermediate path-to-r-script.Rmd

knitr::spin("path-to-r-script.R", knit = FALSE)

# Now create the final HTML output from

# path-to-r-script.Rmd, with desired options

knitr::knit2html(

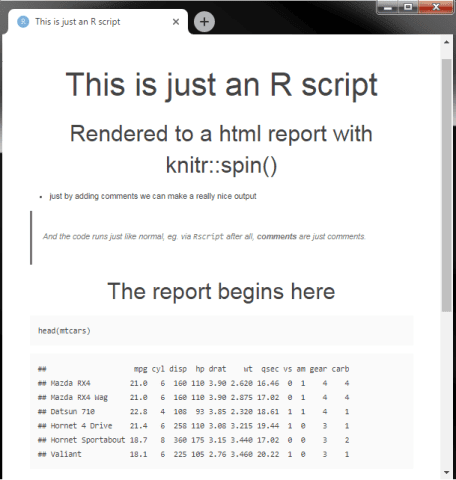
input = "path-to-r-script.Rmd",

stylesheet = "path\_to\_custom.css"

)

Using both of the above options will actually embed the css directly into the HTML output that is produced, making the output larger in size.

Note that the arguments we are looking to provide to knit2html() are implemented as part of ..., so we will have to name them. To look at the details, study the [documentation of markdownToHTML()](https://www.rdocumentation.org/packages/markdown/versions/0.9/topics/markdownToHTML?tap_a=5644-dce66f&tap_s=10907-287229), to which those arguments get passed.



spin with custom [air.css](https://github.com/markdowncss/air)

**Using rmarkdown’s render()**

To produce an HTML report from an R script we can also use rmarkdown::render() on an R script file. This will create a report with slight differences to the default knit() output, one notable for HTML output is that render() will by default include inline base64 representations of fonts and JavaScript sources. It will also include some potentially useful metadata, such as the author’s name and the date of rendering.

**The output\_format powerhouse**

The output of render() is governed mainly by the output\_format argument. Most of the time users will pass on just the name of the format, such as "html\_document", as most of the options are governed by the yaml metadata present at the beginning of our Rmd files.

For R scripts we usually do not use the yaml metadata. In this case, we can take full advantage of the flexibility of that argument, passing a call to rmarkdown::html\_document() with the desired parameters as output\_format.

**Minimalistic output with render()**

To produce a minimalistic HTML output from our path-to-r-script.R script, we can for example specify the following as output\_format:

rmarkdown::render(

"path-to-r-script.R",

output\_format = rmarkdown::html\_document(

theme = NULL,

mathjax = NULL,

highlight = NULL

)

)

**Custom css with render()**

Including a custom css stylesheet is equally simple, just provide a css argument with the css file path to the html\_document() function:

rmarkdown::render(

"path-to-r-script.R",

output\_format = rmarkdown::html\_document(

theme = NULL,

mathjax = NULL,

highlight = NULL,

css = "path\_to\_custom.css"

)

)

An interesting property of including custom css styles is that by default the argument self\_contained is set to TRUE, meaning that the full stylesheet will be embedded into the output HTML file, including all the external css imported into the one we are using. This means that if your stylesheets import external fonts such as the following, those will also be pasted directly into the output:

@import url(http://fonts.googleapis.com/css?family=Open+Sans:300italic,300);

This behavior is different for spin(), which will paste the @import clause into the output as-is, instead of parsing and pasting the actual content of the provided url.

**TL;DR: Just show me the examples**

If instead of reading about it you would like to just test it yourself, I created a very simple R project showcasing the mentioned methods and some more [available via a GitLab repo](https://gitlab.com/jozefhajnala/gists/tree/master/rmarkdown/spin).

The project has the following files:

* src/path-to-r-script.R – an R script with custom formatted comments to be used as the source for creating reports with knitr::spin() and rmarkdown::render()
* rendering\_render.R – an R script that uses rmarkdown::render() to create multiple different output reports based on path-to-r-script.R and save them to outputs/
* rendering\_spin.R – an R script that uses knitr::spin() to create multiple different output reports based on path-to-r-script.R and save them to outputs/
* outputs/ – HTML reports generated from the content of path-to-r-script.R by running rendering\_spin.R and rendering\_render.R
* css/ – Example css used for creating outputs/ex\_04\_spin\_air\_css.html, all credit for the air.css goes to <https://github.com/markdowncss/air>

**References**

* [HTML document](https://bookdown.org/yihui/rmarkdown/html-document.html) chapter of the R Markdown: The Definitive Guide book
* [Create R Markdown reports and presentations even better with these 3 practical tips](https://jozefhajnala.gitlab.io/r/r909-rmarkdown-tips/)
* [air.css](https://github.com/markdowncss/air) style used to create the report on the screenshot above