Building LaTeX Templates for R Markdown to Produce Branded PDF Reports

CSP 2020 Short Course

Ben Barnard

February 20, 2019

Contents

Course Outline

Abstract

Branded reports give a clean, clear and consistent message for data science teams in an organization. We walk through the process of building a latex template distributed through an R package. We begin with a short introduction to rmarkdown and some motivating examples for using branded reports. Then, we demonstrate from scratch how one can build a minimal latex template, and distribute in a R package. We describe some best practices for branding and highlight use of ggplot2 themes to match document branding. Finally, we walk through some further uses such as parameterized reports, using the template for bookdown, and recommendation for deploying the R package at your company.

Learning Objectives

The student should be able to walk away from this class with:

1. a general understanding of rmarkdown,
2. why it is important to have branded reports,
3. a R package with a latex template that uses their companies branding,
4. understanding of best practices in branding,
5. use of ggplot2 themes,
6. and some possible further uses for the using and distributing the template.

About the Instructor

Ben Barnard is a Data Scientist at Wells Fargo in the Team Member Insights group. Ben has a PhD from Baylor University in Statistics.

Jeff Idle is an Analytic Manager at Wells Fargo in the Team Member Insights group. Jeff leads the HR Advanced Analytics & Architecture team. Jeff is currently pursuing a MBA from the University of Minnesota's Carlson School of Management.

Relevance to Conference Goals

We stress using branded reports to communicate clean, clear and consistent messages to your audience. Communication is the most important part of Data Science since decision makers are rarely analytic experts. Branded reports bring a certain professionalism that will be greatly appreciated by administration. Building the latex templates saves time and makes sure every report comes out looking the same. Consistently branded reports allows your team to be recognized immediately by your work product.

Getting Started

Introduction to R Markdown - 45 mins

We discuss the R Markdown document format developed by RStudio. R Markdown is a document format, and `rmarkdown` is an R package. There can be confusion when talking about the two, and in this course we are usually talking about the R Markdown document format. Most of this portion of the class uses references from the Rstudio R Markdown website, the `rmarkdown` package website and the R Markdown book. We will walk through creating an R Markdown document that we will use for the initial template of our branded document.

What is R Markdown

R Markdown is a document format built to embed R code chunks in Markdown documents. R Markdown documents remove most of the formatting aspects of report generation to a "back-end." R Markdown can be used to generate many different output some of which are websites, PDFs, presentations, shiny, and Word documents. In this course we are going to focus on PDFs documents, but in general the same R Markdown document can be used to produce all of the above with minimal changes.

Why should I use R Markdown

R markdown provides a medium to attach and run code inside the content for dashboards, articles, reports, websites, PowerPoint and books. This medium provides a reproducible and portable format that works with text based version control systems. These characteristics generally make R markdown documents easy to put into production systems

How to start an R Markdown document

As previously mentioned R markdown plays well with text based version control systems, and that is because at its core an R markdown document is just a text file. R markdown files have .rmd extensions but you could potentially render any text file no matter the extension. If you expect the RStudio IDE to recognize an R markdown document it needs the .rmd extension. Let us take the easy route and create an R markdown document from the RStudio IDE menu.

Structure of an R Markdown document

````
---
title: "Untitled"
author: "Name"
date: "Date"
output: pdf_document
---

```{r}
1 + 1
```
````

PDF documents

R markdown can be used to build PDFs documents using latex. We can usually tell what R markdown documents are going to render as PDFs by output specified in the yaml header.

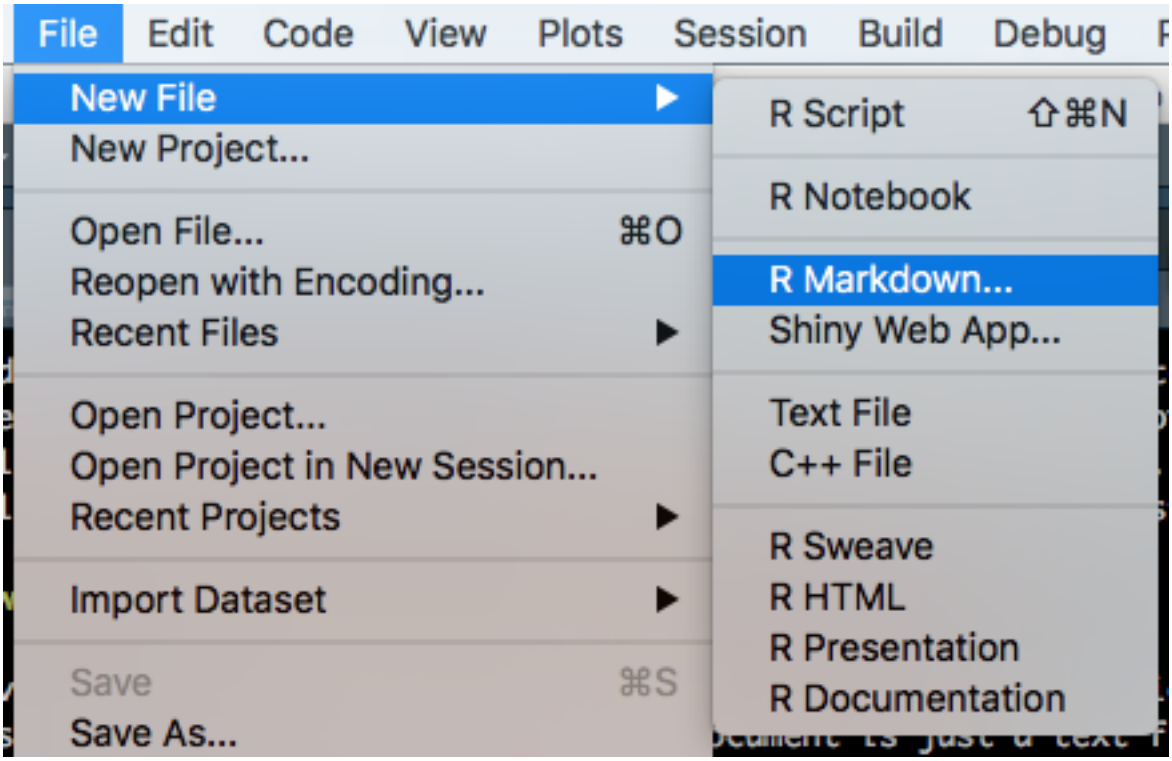

Figure 1:

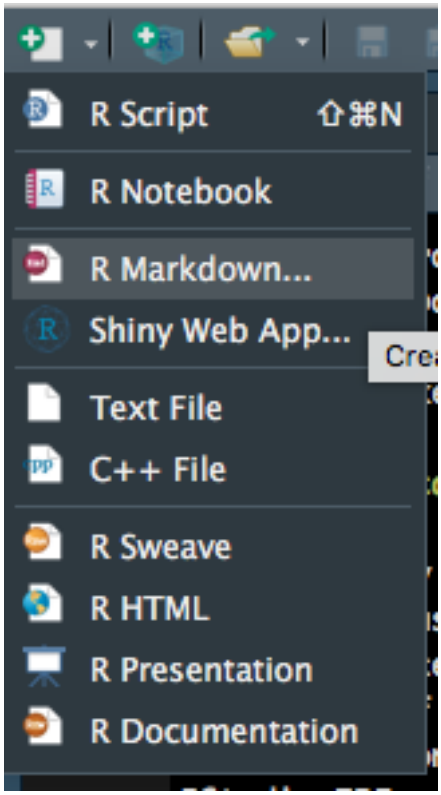

Figure 2:

```
---
title: "Untitled"
author: "Name"
date: "Date"
output: pdf_document
---
```

The PDFs_document output option actually references a function. We are going to create our own output function that wraps the PDFs_document function. This will end up being one of the few functions in the R package we will build. We will discuss further in the R package portion of the course. The PDFs_document function has some options that we can specify in the yaml of our R markdown document. These next subsections discuss some of these options.

Table of contents

For longer reports and documents it is useful to have a Table of Contents. The PDFs documents generated from R markdown can automatically build the table of contents for you. We can build the table of contents by setting `toc: true` in the yaml options under the `pdf_document` output. You might also notice that `toc` is a function call option in `pdf_document`, and we should expect it to be listed under `pdf_document` in the yaml.

```
---
title: "Untitled"
author: "Name"
date: "Date"
output:
  pdf_document:
    toc: true
    toc_depth: 3
---
```

Another option under `pdf_document` was added with `toc`. The yaml `toc_depth` should always be added with `toc: true`. It isn't that you can't specify it, but there really isn't much of a reason to specify it. The `toc_depth` or table of contents depth sets the depth of sections/subsections listed in the table of contents. The number specified by `toc_depth` corresponds to the number of # preceding your section/subsection name in the R markdown document. If you wonder why the subsection isn't showing up in the table of contents then you should count the #s and see if that count matches the `toc_depth`.

Figure options

The `pdf_document` function has several figure options you can put in the r markdown yaml header. The options `fig_width` and `fig_height` specify the default width and height in inches for figures in the document. You might notice that `fig_width` and `fig_height` have default values in the `pdf_document` function as 6.5 and 4.5 inches respectively. The `fig_crop` options controls whether the `pdfcrop` utility is used. The `pdfcrop` utility trims white-space or fixed borders from PDFs graphics. Most people wouldn't notice anything is being done, but it is recommended by the developers to use the `fig_crop` option. The `fig_crop` option is defaulted to `TRUE`. The `fig_caption` option controls whether captions should be rendered with figures. Finally, the `dev` option is for the graphics device used to produce figure output. The `dev` option is `pdf` by default but this could potentially be `png` or `jpeg`. I only ever find it worth while to change the `fig_height` and `fig_width` options to maximize the figure size on the viewable page.

```
title: "Unititled"
output:
  pdf_document:
    fig_width: 7
```

```
    fig_height: 6
```

Data frame printing

The `df_print` option lets us change the display of data frames in our output. Options include `default`, `kable`, and `tibble`. I never really just print data frames in my output, and there is generally some formatting before they are presentation ready. I usually leave this option as it is and build my presentation tables with the `xtable` package.

```
---
title: "Untitled"
output:
  pdf_document:
    df_print: kable
---
```

Syntax highlighting

The `highlight` option specifies how code chunks are displayed in the document. Some options you can use are `default`, `espresso`, `haddock`, `kate`, `monochrome`, `pygments`, `tango`, and `zenburn`. Please don't ever using any of the dark themes.

Latex options

Aside form the options we see in `pdf_document` there are also options that are specified in the top-level YAML header. These top-level options can also be specified within the `render` function, but this is only important if you want to change the defaults.

Table 1: Top-level LaTex YAML options.

Variable	Description
lang	Document language code
fontsize	Font size (e.g., `10pt`, `11pt`, or `12pt`)
documentclass	LaTex document class (e.g., `article`)
classoption	Options for documentclass (e.g., `oneside`)
geometry	Options for geometry class (e.g., `margin=1in`)
mainfont, sansfont, monofont, mathfont	Document fonts (works only with `xelatex` and `lualatex`)
linkcolor, urlcolor, citecolor	Color for internal, external, and citation links

We are going to simplify our LaTex templates significantly and remove many of these options, but these are available in the default LaTex template shipped with `rmarkdown`.

Latex packages for citations

The developers suggest using a LaTex citation package might produce better results than the default `pando-citeproc`. The LaTex packages `natbib` and `biblatex` are two options. I don't see much reason to change this option even with PDF documents. To use citations you will need to specify a bibliography with the top-level YAML `bibliography` or use inline references. I do not recommend inline references. My suggestion would be to use a bibliography manager and export you bibliography in one of the acceptable formats.

Table 2: Bibliography formats

Format	File extension
MODS	.mods
BibLaTeX	.bib
BibTeX	.bibtex
RIS	.ris
EndNote	.enl
EndNote XML	.xml
ISI	.wos
MEDLINE	.medline
Copac	.copac
JSON citeproc	.json

Advanced customization

There a a few options that allow us to do some more customization and debugging of the underlying LaTeX code. We can specify the `latex_engine` used such as `pdflatex`, `xelatex`, and `lualatex`. We can also keep the intermediate LaTeX code using `keep_tex: true`, and include snippets of LaTeX code in the document. The option `includes` allows us to put code `in_header`, `before_body`, and `after_body`. What wwe are going to end up using and is the power behind document templates is the `template` option, but we aren't going to specify our template in document YAML header.

```
---
title: "Habits"
output:
  pdf_document:
    latex_engin: pdflatex
    keep_tex: true
    includes:
      in_header: preamble.tex
      before_body: doc-prefix.tex
      after_body: doc-suffix.tex
    template: custom-template.tex
---
```

Lets Start building a report

Now that we have seen all these options lets start building a rmarkdown report on Iris Petal and Sepal Length. We are using the Iris data set as a minimal example to base our future template. We will end up renaming some things later on. Lets start by starting a new r markdown document by going to File > New File > R Markdown... You can fill out the title and author to anything you want because we will change these later on. Make sure you choose PDF, but it isn't the end of the world if you don't.

Building the header

Our header should look something like this:

```
---
title: "Iris Petal and Sepal Length"
author: "Ben Barnard"
date: "February 15, 2020"
```

```
output: pdf_document
---
```

I plan to run this report many times and I would like the date to update each time I run the report.

Running R code in the yaml

To run R code in the YAML header we just need to add `!r` before the R code we wish to run. In this case I want to run `Sys.Date()` to use the run date for the report date.

```
---
title: "Iris Petal and Sepal Length"
author: "Ben Barnard"
date: !r Sys.Date()
output: pdf_document
---
```

I think thats the important stuff for the header right now. Let us start working on the body.

R markdown body

Notice the first code chunk in document. The `knitr::opts_chunk$set(echo = TRUE)` sets the code chunk options for the rest of the code chunks in the document. Depending on the audience we might want to set `echo = FALSE`, and we might also want to set `message = FALSE` and `warning = FALSE`.

```
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
```

Rendering R Markdown documents

Knit

Render function

Engines

Python

Shell Scripts

SQL

Rcpp

Stan

JavaScript

CSS

Julia

C

FORTRAN

Introduction to R packages - 35 mins

What is an R package

Why should I build R packages

Distribution

Testing

Reproducibility

Documentation

How to build your first R package

GitHub

usethis

Ways to distribute R packages

Source and binary

GitHub

drat

CRAN

Introduction to LaTex - 60 mins

What is LaTex

Why should I use LaTex

Reproducibility

Automation

Distribution

Not a Microsoft product

How to start your first LaTex document

Bringing It all together - 30 mins

Possible extensions - 35 mins