**R Markdown: The Definitive Guide**

*Yihui Xie, J. J. Allaire, Garrett Grolemund*

*2019-11-20*

Preface

**Note**: This book has been published by [Chapman & Hall/CRC](https://www.crcpress.com/p/book/9781138359338). The online version of this book is free to read here (thanks to Chapman & Hall/CRC), and licensed under the [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](http://creativecommons.org/licenses/by-nc-sa/4.0/).

The document format “R Markdown” was first introduced in the **knitr** package (Xie [2015](https://bookdown.org/yihui/rmarkdown/#ref-xie2015), [2019c](https://bookdown.org/yihui/rmarkdown/#ref-R-knitr)) in early 2012. The idea was to embed code chunks (of R or other languages) in Markdown documents. In fact, **knitr** supported several authoring languages from the beginning in addition to Markdown, including LaTeX, HTML, AsciiDoc, reStructuredText, and Textile. Looking back over the five years, it seems to be fair to say that Markdown has become the most popular document format, which is what we expected. The simplicity of Markdown clearly stands out among these document formats.

However, the original version of Markdown [invented by John Gruber](https://en.wikipedia.org/wiki/Markdown) was often found overly simple and not suitable to write highly technical documents. For example, there was no syntax for tables, footnotes, math expressions, or citations. Fortunately, John MacFarlane created a wonderful package named Pandoc ([http://pandoc.org](http://pandoc.org/)) to convert Markdown documents (and many other types of documents) to a large variety of output formats. More importantly, the Markdown syntax was significantly enriched. Now we can write more types of elements with Markdown while still enjoying its simplicity.

In a nutshell, R Markdown stands on the shoulders of **knitr** and Pandoc. The former executes the computer code embedded in Markdown, and converts R Markdown to Markdown. The latter renders Markdown to the output format you want (such as PDF, HTML, Word, and so on).

The **rmarkdown** package (J. Allaire, Xie, McPherson, et al. [2019](https://bookdown.org/yihui/rmarkdown/#ref-R-rmarkdown)) was first created in early 2014. During the past four years, it has steadily evolved into a relatively complete ecosystem for authoring documents, so it is a good time for us to provide a definitive guide to this ecosystem now. At this point, there are a large number of tasks that you could do with R Markdown:

* Compile a single R Markdown document to a report in different formats, such as PDF, HTML, or Word.
* Create notebooks in which you can directly run code chunks interactively.
* Make slides for presentations (HTML5, LaTeX Beamer, or PowerPoint).
* Produce dashboards with flexible, interactive, and attractive layouts.
* Build interactive applications based on Shiny.
* Write journal articles.
* Author books of multiple chapters.
* Generate websites and blogs.

There is a fundamental assumption underneath R Markdown that users should be aware of: we assume it suffices that only a limited number of features are supported in Markdown. By “features”, we mean the types of elements you can create with native Markdown. The limitation is a great feature, not a bug. R Markdown may not be the right format for you if you find these elements not enough for your writing: paragraphs, (section) headers, block quotations, code blocks, (numbered and unnumbered) lists, horizontal rules, tables, inline formatting (emphasis, strikeout, superscripts, subscripts, verbatim, and small caps text), LaTeX math expressions, equations, links, images, footnotes, citations, theorems, proofs, and examples. We believe this list of elements suffice for most technical and non-technical documents. It may not be impossible to support other types of elements in R Markdown, but you may start to lose the simplicity of Markdown if you wish to go that far.

Epictetus once said, “*Wealth consists not in having great possessions, but in having few wants.*” The spirit is also reflected in Markdown. If you can control your preoccupation with pursuing typesetting features, you should be much more efficient in writing the content and can become a prolific author. It is entirely possible to succeed with simplicity. Jung Jae-sung was a legendary badminton player with a remarkably simple playing style: he did not look like a talented player and was very short compared to other players, so most of the time you would just see him jump three feet off the ground and smash like thunder over and over again in the back court until he beats his opponents.

Please do not underestimate the customizability of R Markdown because of the simplicity of its syntax. In particular, Pandoc templates can be surprisingly powerful, as long as you understand the underlying technologies such as LaTeX and CSS, and are willing to invest time in the appearance of your output documents (reports, books, presentations, and/or websites). As one example, you may check out the [PDF report](http://files.kff.org/attachment/Report-Employer-Health-Benefits-Annual-Survey-2017) of the [2017 Employer Health Benefits Survey](https://www.kff.org/health-costs/report/2017-employer-health-benefits-survey/). It looks fairly sophisticated, but was actually produced via **bookdown** (Xie [2016](https://bookdown.org/yihui/rmarkdown/#ref-xie2016)), which is an R Markdown extension. A custom LaTeX template and a lot of LaTeX tricks were used to generate this report. Not surprisingly, this very book that you are reading right now was also written in R Markdown, and its full source is publicly available in the GitHub repository <https://github.com/rstudio/rmarkdown-book>.

R Markdown documents are often portable in the sense that they can be compiled to multiple types of output formats. Again, this is mainly due to the simplified syntax of the authoring language, Markdown. The simpler the elements in your document are, the more likely that the document can be converted to different formats. Similarly, if you heavily tailor R Markdown to a specific output format (e.g., LaTeX), you are likely to lose the portability, because not all features in one format work in another format.

Last but not least, your computing results will be more likely to be reproducible if you use R Markdown (or other **knitr**-based source documents), compared to the manual cut-and-paste approach. This is because the results are dynamically generated from computer source code. If anything goes wrong or needs to be updated, you can simply fix or update the source code, compile the document again, and the results will automatically updated. You can enjoy reproducibility and convenience at the same time.

References

Allaire, JJ, Yihui Xie, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, Hadley Wickham, Joe Cheng, Winston Chang, and Richard Iannone. 2019. *Rmarkdown: Dynamic Documents for R*. <https://github.com/rstudio/rmarkdown>.

Xie, Yihui. 2015. *Dynamic Documents with R and Knitr*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. <https://yihui.name/knitr/>.

Xie, Yihui. 2016. *Bookdown: Authoring Books and Technical Documents with R Markdown*. Boca Raton, Florida: Chapman; Hall/CRC. <https://github.com/rstudio/bookdown>.

Xie, Yihui. 2019c. *Knitr: A General-Purpose Package for Dynamic Report Generation in R*. <https://yihui.org/knitr/>.

## **How to read this book**

This book may serve you better as a reference book than a textbook. It contains a large number of technical details, and we do not expect you to read it from beginning to end, since you may easily feel overwhelmed. Instead, think about your background and what you want to do first, and go to the relevant chapters or sections. For example:

* I just want to finish my course homework (Chapter [2](https://bookdown.org/yihui/rmarkdown/basics.html#basics) should be more than enough for you).
* I know this is an R Markdown book, but I use Python more than R (Go to Section [2.7.1](https://bookdown.org/yihui/rmarkdown/language-engines.html#python)).
* I want to embed interactive plots in my reports, or want my readers to be able change my model parameters interactively and see results on the fly (Check out Section [2.8](https://bookdown.org/yihui/rmarkdown/interactive-documents.html#interactive-documents)).
* I know the output format I want to use, and I want to customize its appearance (Check out the documentation of the specific output format in Chapter [3](https://bookdown.org/yihui/rmarkdown/documents.html#documents) or Chapter [4](https://bookdown.org/yihui/rmarkdown/presentations.html#presentations)). For example, I want to customize the template for my PowerPoint presentation (Go to Section [4.4.1](https://bookdown.org/yihui/rmarkdown/powerpoint-presentation.html#ppt-templates)).
* I want to build a business dashboard highlighting some key figures and indicators (Go to Chapter [5](https://bookdown.org/yihui/rmarkdown/dashboards.html#dashboards)).
* I heard about yolo = TRUE from a friend, and I’m curious what that means in the **xaringan** package (Go to Chapter [7](https://bookdown.org/yihui/rmarkdown/xaringan.html#xaringan)).
* I want to build a personal website (Go to Chapter [10](https://bookdown.org/yihui/rmarkdown/websites.html#websites)), or write a book (Go to Chapter [12](https://bookdown.org/yihui/rmarkdown/books.html#books)).
* I want to write a paper and submit to the Journal of Statistical Software (Go to Chapter [13](https://bookdown.org/yihui/rmarkdown/journals.html#journals)).
* I want to build an interactive tutorial with exercises for my students to learn a topic (Go to Chapter [14](https://bookdown.org/yihui/rmarkdown/learnr.html#learnr)).
* I’m familiar with R Markdown now, and I want to generate personalized reports for all my customers using the same R Markdown template (Try parameterized reports in Chapter [15](https://bookdown.org/yihui/rmarkdown/parameterized-reports.html#parameterized-reports)).
* I know some JavaScript, and want to build an interface in R to call an interested JavaScript library from R (Learn how to develop HTML widgets in Chapter [16](https://bookdown.org/yihui/rmarkdown/html-widgets.html#html-widgets)).
* I want to build future reports with a company branded template that shows our logo and uses our unique color theme (Go to Chapter [17](https://bookdown.org/yihui/rmarkdown/document-templates.html#document-templates)).

If you are not familiar with R Markdown, we recommend that you read at least Chapter [2](https://bookdown.org/yihui/rmarkdown/basics.html#basics) to learn the basics. All the rest of the chapters in this book can be read in any order you desire. They are pretty much orthogonal to each other. However, to become familiar with R Markdown output formats, you may want to thumb through the HTML document format in Section [3.1](https://bookdown.org/yihui/rmarkdown/html-document.html#html-document), because many other formats share the same options as this format.

## **Structure of the book**

This book consists of four parts. Part I covers the basics: Chapter [1](https://bookdown.org/yihui/rmarkdown/installation.html#installation) introduces how to install the relevant packages, and Chapter [2](https://bookdown.org/yihui/rmarkdown/basics.html#basics) is an overview of R Markdown, including the possible output formats, the Markdown syntax, the R code chunk syntax, and how to use other languages in R Markdown.

Part II is the detailed documentation of built-in output formats in the **rmarkdown** package, including document formats and presentation formats.

Part III lists about ten R Markdown extensions that enable you to build different applications or generate output documents with different styles. Chapter [5](https://bookdown.org/yihui/rmarkdown/dashboards.html#dashboards) introduces the basics of building flexible dashboards with the R package **flexdashboard**. Chapter [6](https://bookdown.org/yihui/rmarkdown/tufte-handouts.html#tufte-handouts) documents the **tufte** package, which provides a unique document style used by Edward Tufte. Chapter [7](https://bookdown.org/yihui/rmarkdown/xaringan.html#xaringan) introduces the **xaringan** package for another highly flexible and customizable HTML5 presentation format based on the JavaScript library remark.js. Chapter [8](https://bookdown.org/yihui/rmarkdown/revealjs.html#revealjs) documents the **revealjs** package, which provides yet another appealing HTML5 presentation format based on the JavaScript library reveal.js. Chapter [9](https://bookdown.org/yihui/rmarkdown/community.html#community) introduces a few output formats created by the R community, such as the **prettydoc** package, which features lightweight HTML document formats. Chapter [10](https://bookdown.org/yihui/rmarkdown/websites.html#websites) teaches you how to build websites using either the **blogdown** package or **rmarkdown**’s built-in site generator. Chapter [11](https://bookdown.org/yihui/rmarkdown/pkgdown.html#pkgdown) explains the basics of the **pkgdown** package, which can be used to quickly build documentation websites for R packages. Chapter [12](https://bookdown.org/yihui/rmarkdown/books.html#books) introduces how to write and publish books with the **bookdown** package. Chapter [13](https://bookdown.org/yihui/rmarkdown/journals.html#journals) is an overview of the **rticles** package for authoring journal articles. Chapter [14](https://bookdown.org/yihui/rmarkdown/learnr.html#learnr) introduces how to build interactive tutorials with exercises and/or quiz questions.

Part IV covers other topics about R Markdown, and some of them are advanced (in particular, Chapter [16](https://bookdown.org/yihui/rmarkdown/html-widgets.html#html-widgets)). Chapter [15](https://bookdown.org/yihui/rmarkdown/parameterized-reports.html#parameterized-reports) introduces how to generate different reports with the same R Markdown source document and different parameters. Chapter [16](https://bookdown.org/yihui/rmarkdown/html-widgets.html#html-widgets) teaches developers how to build their own HTML widgets for interactive visualization and applications with JavaScript libraries. Chapter [17](https://bookdown.org/yihui/rmarkdown/document-templates.html#document-templates) shows how to create custom R Markdown and Pandoc templates so that you can fully customize the appearance and style of your output document. Chapter [18](https://bookdown.org/yihui/rmarkdown/new-formats.html#new-formats) explains how to create your own output formats if the existing formats do not meet your need. Chapter [19](https://bookdown.org/yihui/rmarkdown/shiny-documents.html#shiny-documents) shows how to combine the Shiny framework with R Markdown, so that your readers can interact with the reports by changing the values of certain input widgets and seeing updated results immediately.

Note that this book is intended to be a guide instead of the comprehensive documentation of all topics related to R Markdown. Some chapters are only overviews, and you may need to consult the full documentation elsewhere (often freely available online). Such examples include Chapters [5](https://bookdown.org/yihui/rmarkdown/dashboards.html#dashboards), [10](https://bookdown.org/yihui/rmarkdown/websites.html#websites), [11](https://bookdown.org/yihui/rmarkdown/pkgdown.html#pkgdown), [12](https://bookdown.org/yihui/rmarkdown/books.html#books), and [14](https://bookdown.org/yihui/rmarkdown/learnr.html#learnr).

## **Software information and conventions**

The R session information when compiling this book is shown below:

xfun::**session\_info**(**c**(

'blogdown', 'bookdown', 'knitr', 'rmarkdown', 'htmltools',

'reticulate', 'rticles', 'flexdashboard', 'learnr', 'shiny',

'revealjs', 'pkgdown', 'tinytex', 'xaringan', 'tufte'

), dependencies = FALSE)

## R version 3.6.1 (2019-07-05)

## Platform: x86\_64-apple-darwin15.6.0 (64-bit)

## Running under: macOS Catalina 10.15.1

##

## Locale: en\_US.UTF-8 / en\_US.UTF-8 / en\_US.UTF-8 / C / en\_US.UTF-8 / en\_US.UTF-8

##

## Package version:

## blogdown\_0.17 bookdown\_0.15.2

## flexdashboard\_0.5.1.1 htmltools\_0.4.0

## knitr\_1.26.1 learnr\_0.10.0

## pkgdown\_1.4.1 reticulate\_1.13

## revealjs\_0.9 rmarkdown\_1.17.3

## rticles\_0.12 shiny\_1.4.0

## tinytex\_0.17 tufte\_0.5

## xaringan\_0.13

##

## Pandoc version: 2.3.1

We do not add prompts (> and +) to R source code in this book, and we comment out the text output with two hashes ## by default, as you can see from the R session information above. This is for your convenience when you want to copy and run the code (the text output will be ignored since it is commented out). Package names are in bold text (e.g., **rmarkdown**), and inline code and filenames are formatted in a typewriter font (e.g., knitr::knit('foo.Rmd')). Function names are followed by parentheses (e.g., blogdown::serve\_site()). The double-colon operator :: means accessing an object from a package.

“Rmd” is the filename extension of R Markdown files, and also an abbreviation of R Markdown in this book.

# Chapter 1 Installation

We assume you have already installed R ([https://www.r-project.org](https://www.r-project.org/)) (R Core Team [2019](https://bookdown.org/yihui/rmarkdown/installation.html#ref-R-base)) and the RStudio IDE ([https://www.rstudio.com](https://www.rstudio.com/)). RStudio is not required but recommended, because it makes it easier for an average user to work with R Markdown. If you do not have RStudio IDE installed, you will have to install Pandoc ([http://pandoc.org](http://pandoc.org/)), otherwise there is no need to install Pandoc separately because RStudio has bundled it. Next you can install the **rmarkdown** package in R:

*# Install from CRAN*

**install.packages**('rmarkdown')

*# Or if you want to test the development version,*

*# install from GitHub*

**if** (!**requireNamespace**("devtools"))

**install.packages**('devtools')

devtools::**install\_github**('rstudio/rmarkdown')

If you want to generate PDF output, you will need to install LaTeX. For R Markdown users who have not installed LaTeX before, we recommend that you install TinyTeX (<https://yihui.name/tinytex/>):

**install.packages**("tinytex")

tinytex::**install\_tinytex**() *# install TinyTeX*

TinyTeX is a lightweight, portable, cross-platform, and easy-to-maintain LaTeX distribution. The R companion package **tinytex** (Xie [2019e](https://bookdown.org/yihui/rmarkdown/installation.html#ref-R-tinytex)) can help you automatically install missing LaTeX packages when compiling LaTeX or R Markdown documents to PDF, and also ensures a LaTeX document is compiled for the correct number of times to resolve all cross-references. If you do not understand what these two things mean, you should probably follow our recommendation to install TinyTeX, because these details are often not worth your time or attention.

With the **rmarkdown** package, RStudio/Pandoc, and LaTeX, you should be able to compile most R Markdown documents. In some cases, you may need other software packages, and we will mention them when necessary.

### References

R Core Team. 2019. R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing. [https://www.R-project.org/](https://www.r-project.org/).

Xie, Yihui. 2019e. Tinytex: Helper Functions to Install and Maintain Tex Live, and Compile Latex Documents. [https://CRAN.R-project.org/package=tinytex](https://cran.r-project.org/package=tinytex).

Chapter 2 Basics

R Markdown provides an authoring framework for data science. You can use a single R Markdown file to both

* save and execute code, and
* generate high quality reports that can be shared with an audience.

R Markdown was designed for easier reproducibility, since both the computing code and narratives are in the same document, and results are automatically generated from the source code. R Markdown supports dozens of static and dynamic/interactive output formats.

If you prefer a video introduction to R Markdown, we recommend that you check out the website [https://rmarkdown.rstudio.com](https://rmarkdown.rstudio.com/), and watch the videos in the “Get Started” section, which cover the basics of R Markdown.

Below is a minimal R Markdown document, which should be a plain-text file, with the conventional extension .Rmd:

---

title: "Hello R Markdown"

author: "Awesome Me"

date: "2018-02-14"

output: html\_document

---

This is a paragraph in an R Markdown document.

Below is a code chunk:

```{r}

fit = lm(dist ~ speed, data = cars)

b = coef(fit)

plot(cars)

abline(fit)

```

The slope of the regression is `r b[1]`.

```

You can create such a text file with any editor (including but not limited to RStudio). If you use RStudio, you can create a new Rmd file from the menu File -> New File -> R Markdown.

There are three basic components of an R Markdown document: the metadata, text, and code. The metadata is written between the pair of three dashes ---. The syntax for the metadata is YAML (YAML Ain’t Markup Language, <https://en.wikipedia.org/wiki/YAML>), so sometimes it is also called the YAML metadata or the YAML frontmatter. Before it bites you hard, we want to warn you in advance that indentation matters in YAML, so do not forget to indent the sub-fields of a top field properly. See the [Appendix B.2](https://bookdown.org/yihui/bookdown/r-markdown.html) of Xie ([2016](https://bookdown.org/yihui/rmarkdown/basics.html#ref-xie2016)) for a few simple examples that show the YAML syntax.

The body of a document follows the metadata. The syntax for text (also known as prose or narratives) is Markdown, which is introduced in Section [2.5](https://bookdown.org/yihui/rmarkdown/markdown-syntax.html#markdown-syntax). There are two types of computer code, which are explained in detail in Section [2.6](https://bookdown.org/yihui/rmarkdown/r-code.html#r-code):

* A code chunk starts with three backticks like ```{r} where r indicates the language name,[1](https://bookdown.org/yihui/rmarkdown/basics.html#fn1) and ends with three backticks. You can write chunk options in the curly braces (e.g., set the figure height to 5 inches: ```{r, fig.height=5}).
* An inline R code expression starts with `r and ends with a backtick `.

Figure [2.1](https://bookdown.org/yihui/rmarkdown/basics.html#fig:hello-rmd) shows the above example in the RStudio IDE. You can click the Knit button to compile the document (to an HTML page). Figure [2.2](https://bookdown.org/yihui/rmarkdown/basics.html#fig:hello-rmd-out) shows the output in the RStudio Viewer.

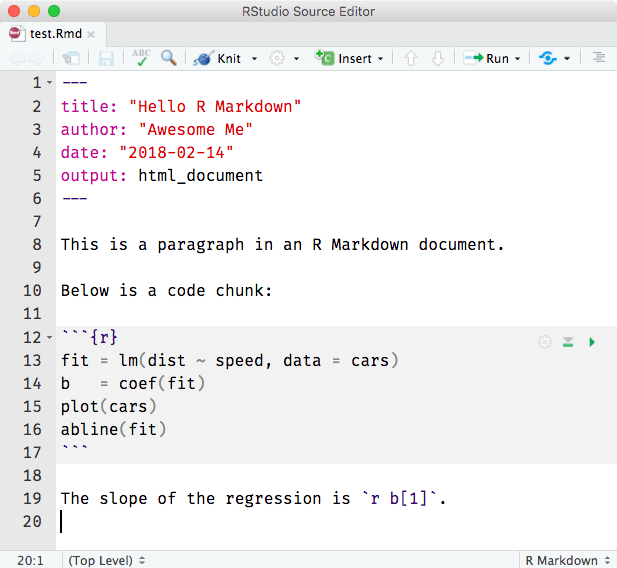


FIGURE 2.1: A minimal R Markdown example in RStudio.

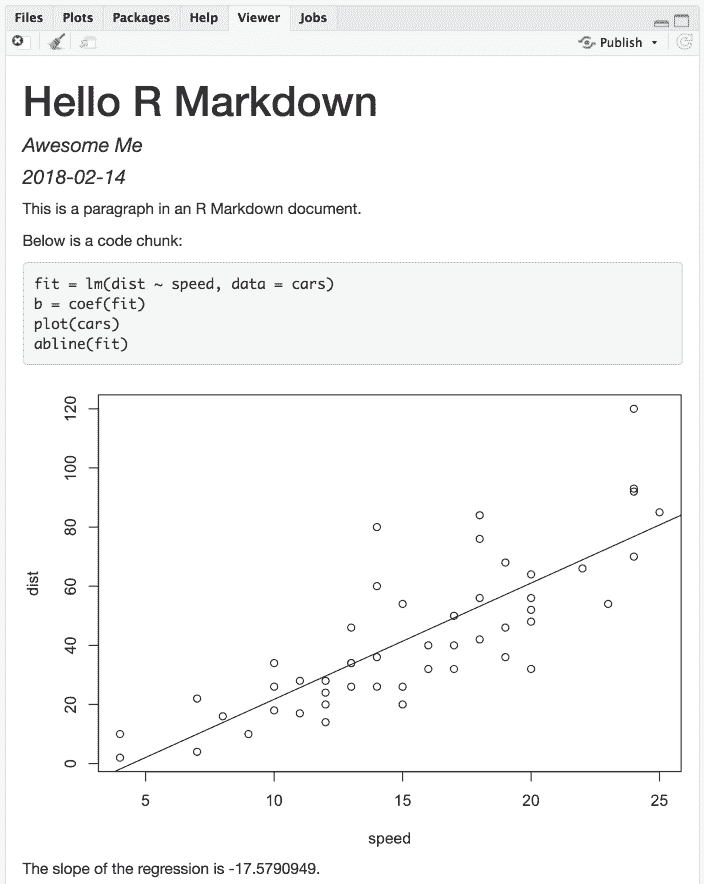


FIGURE 2.2: The output document of the minimal R Markdown example in RStudio.

Now please take a closer look at the example. Did you notice a problem? The object b is the vector of coefficients of length 2 from the linear regression; b[1] is actually the intercept, and b[2] is the slope! This minimal example shows you why R Markdown is great for reproducible research: it includes the source code right inside the document, which makes it easy to discover and fix problems, as well as update the output document. All you have to do is change b[1] to b[2], and click the Knit button again. Had you copied a number -17.579 computed elsewhere into this document, it would be very difficult to realize the problem. In fact, I had used this example a few times by myself in my presentations before I discovered this problem during one of my talks, but I discovered it anyway.

Although the above is a toy example, it could become a horror story if it happens in scientific research that was not done in a reproducible way (e.g., cut-and-paste). Here are two of my personal favorite videos on this topic:

* “A reproducible workflow” by Ignasi Bartomeus and Francisco Rodríguez-Sánchez (<https://youtu.be/s3JldKoA0zw>). It is a 2-min video that looks artistic but also shows very common and practical problems in data analysis.
* “The Importance of Reproducible Research in High-Throughput Biology” by Keith Baggerly (<https://youtu.be/7gYIs7uYbMo>). You will be impressed by both the content and the style of this lecture. Keith Baggerly and Kevin Coombes were the two notable heroes in revealing [the Duke/Potti scandal](https://en.wikipedia.org/wiki/Anil_Potti), which was described as “one of the biggest medical research frauds ever” by the television program “60 Minutes”.

It is fine for humans to err (in computing), as long as the source code is readily available.

References

Xie, Yihui. 2016. *Bookdown: Authoring Books and Technical Documents with R Markdown*. Boca Raton, Florida: Chapman; Hall/CRC. <https://github.com/rstudio/bookdown>.

1. It is not limited to the R language; see Section [2.7](https://bookdown.org/yihui/rmarkdown/language-engines.html#language-engines) for how to use other languages.[↩](https://bookdown.org/yihui/rmarkdown/basics.html#fnref1)

## **2.1 Example applications**

Now you have learned the very basic concepts of R Markdown. The idea should be simple enough: interweave narratives with code in a document, knit the document to dynamically generate results from the code, and you will get a report. This idea was not invented by R Markdown, but came from an early programming paradigm called “Literate Programming” (Knuth [1984](https://bookdown.org/yihui/rmarkdown/basics-examples.html#ref-knuth1984)).

Due to the simplicity of Markdown and the powerful R language for data analysis, R Markdown has been widely used in many areas. Before we dive into the technical details, we want to show some examples to give you an idea of its possible applications.

### 2.1.1 Airbnb’s knowledge repository

Airbnb uses R Markdown to document all their analyses in R, so they can combine code and data visualizations in a single report (Bion, Chang, and Goodman [2018](https://bookdown.org/yihui/rmarkdown/basics-examples.html#ref-bion2018)). Eventually all reports are carefully peer-reviewed and published to a company knowledge repository, so that anyone in the company can easily find analyses relevant to their team. Data scientists are also able to learn as much as they want from previous work or reuse the code written by previous authors, because the full R Markdown source is available in the repository.

### 2.1.2 Homework assignments on RPubs

A huge number of homework assignments have been published to the website [https://RPubs.com](https://rpubs.com/) (a free publishing platform provided by RStudio), which shows that R Markdown is easy and convenient enough for students to do their homework assignments (see Figure [2.3](https://bookdown.org/yihui/rmarkdown/basics-examples.html#fig:rpubs)). When I was still a student, I did most of my homework assignments using Sweave, which was a much earlier implementation of literate programming based on the S language (later R) and LaTeX. I was aware of the importance of reproducible research but did not enjoy LaTeX, and few of my classmates wanted to use Sweave. Right after I graduated, R Markdown was born, and it has been great to see so many students do their homework in the reproducible manner.

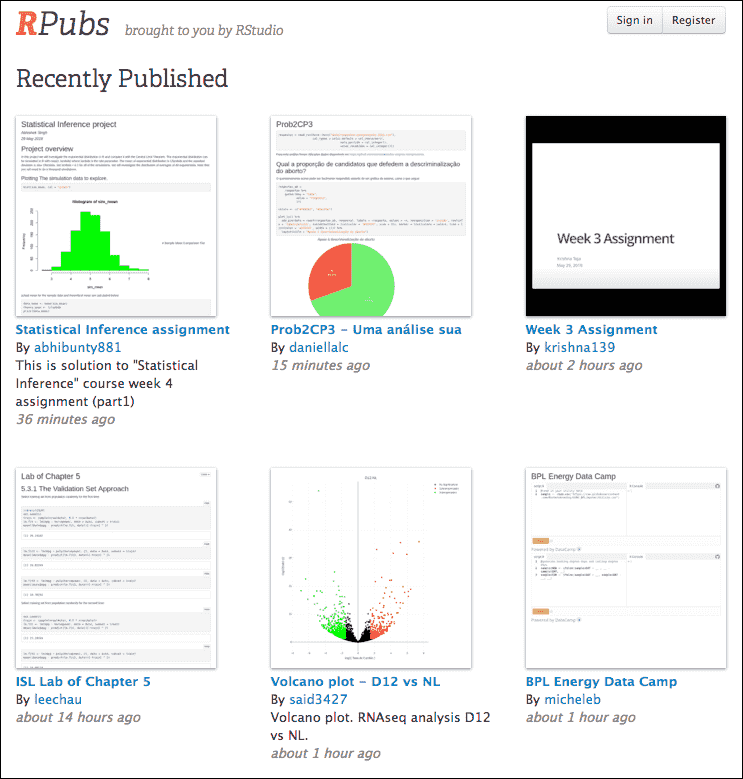


FIGURE 2.3: A screenshot of RPubs.com that contains some homework assginments submitted by students.

In a 2016 JSM (Joint Statistical Meetings) talk, I proposed that course instructors could sometimes intentionally insert some wrong values in the source data before providing it to the students for them to analyze the data in the homework, then correct these values the next time, and ask them to do the analysis again. This way, students should be able to realize the problems with the traditional cut-and-paste approach for data analysis (i.e., run the analysis separately and copy the results manually), and the advantage of using R Markdown to automatically generate the report.

### 2.1.3 Personalized mail

One thing you should remember about R Markdown is that you can programmatically generate reports, although most of the time you may be just clicking the Knit button in RStudio to generate a single report from a single source document. Being able to program reports is a super power of R Markdown.

Mine Çetinkaya-Rundel once wanted to create personalized handouts for her workshop participants. She used a template R Markdown file, and knitted it in a for-loop to generate 20 PDF files for the 20 participants. Each PDF contained both personalized information and common information. You may read the article <https://rmarkdown.rstudio.com/articles_mail_merge.html> for the technical details.

### 2.1.4 2017 Employer Health Benefits Survey

The [2017 Employer Health Benefits Survey](https://www.kff.org/health-costs/report/2017-employer-health-benefits-survey/) was designed and analyzed by the Kaiser Family Foundation, NORC at the University of Chicago, and Health Research & Educational Trust. The full PDF report was written in R Markdown (with the **bookdown** package). It has a unique appearance, which was made possible by heavy customizations in the LaTeX template. This example shows you that if you really care about typesetting, you are free to apply your knowledge about LaTeX to create highly sophisticated reports from R Markdown.

### 2.1.5 Journal articles

Chris Hartgerink explained how and why he used R Markdown to write dynamic research documents in the post at <https://elifesciences.org/labs/cad57bcf/composing-reproducible-manuscripts-using-r-markdown>. He published a paper titled “Too Good to be False: Nonsignificant Results Revisited” with two co-authors (Hartgerink, Wicherts, and Assen [2017](https://bookdown.org/yihui/rmarkdown/basics-examples.html#ref-hartgerink2017)). The manuscript was written in R Markdown, and results were dynamically generated from the code in R Markdown.

When checking the accuracy of P-values in the psychology literature, his colleagues and he found that P-values could be mistyped or miscalculated, which could lead to inaccurate or even wrong conclusions. If the P-values were dynamically generated and inserted instead of being manually copied from statistical programs, the chance for those problems to exist would be much lower.

Lowndes et al. ([2017](https://bookdown.org/yihui/rmarkdown/basics-examples.html#ref-lowndes2017)) also shows that using R Markdown (and version control) not only enhances reproducibility, but also produces better scientific research in less time.

### 2.1.6 Dashboards at eelloo

R Markdown is used at eelloo ([https://eelloo.nl](https://eelloo.nl/)) to design and generate research reports. Here is one of their examples (in Dutch): <https://eelloo.nl/groepsrapportages-met-infographics/>, where you can find gauges, bar charts, pie charts, wordclouds, and other types of graphs dynamically generated and embedded in dashboards.

### 2.1.7 Books

We will introduce the R Markdown extension **bookdown** in Chapter [12](https://bookdown.org/yihui/rmarkdown/books.html#books). It is an R package that allows you to write books and long-form reports with multiple Rmd files. After this package was published, a large number of books have emerged. You can find a subset of them at [https://bookdown.org](https://bookdown.org/). Some of these books have been printed, and some only have free online versions.

There have also been students who wrote their dissertations/theses with **bookdown**, such as Ed Berry: <https://eddjberry.netlify.com/post/writing-your-thesis-with-bookdown/>. Chester Ismay has even provided an R package **thesisdown** (<https://github.com/ismayc/thesisdown>) that can render a thesis in various formats. Several other people have customized this package for their own institutions, such as Zhian N. Kamvar’s **beaverdown** (<https://github.com/zkamvar/beaverdown>) and Ben Marwick’s **huskydown** (<https://github.com/benmarwick/huskydown>).

### 2.1.8 Websites

The **blogdown** package to be introduced in Chapter [10](https://bookdown.org/yihui/rmarkdown/websites.html#websites) can be used to build general-purpose websites (including blogs and personal websites) based on R Markdown. You may find tons of examples at <https://github.com/rbind> or by searching on Twitter: <https://twitter.com/search?q=blogdown>. Here are a few impressive websites that I can quickly think of off the top of my head:

* Rob J Hyndman’s personal website: [https://robjhyndman.com](https://robjhyndman.com/) (a very comprehensive academic website).
* Amber Thomas’s personal website: [https://amber.rbind.io](https://amber.rbind.io/) (a rich project portfolio).
* Emi Tanaka’s personal website: [https://emitanaka.github.io](https://emitanaka.github.io/) (in particular, check out the beautiful showcase page).
* “Live Free or Dichotomize” by Nick Strayer and Lucy D’Agostino McGowan: [http://livefreeordichotomize.com](http://livefreeordichotomize.com/) (the layout is elegant, and the posts are useful and practical).

### References

Bion, Ricardo, Robert Chang, and Jason Goodman. 2018. “How R Helps Airbnb Make the Most of Its Data.” The American Statistician 72 (1). Taylor & Francis: 46–52. <https://doi.org/10.1080/00031305.2017.1392362>.

Hartgerink, Chris HJ, Jelte M Wicherts, and Marcel ALM van Assen. 2017. “Too Good to Be False: Nonsignificant Results Revisited.” Collabra: Psychology 3 (1). The Regents of the University of California.

Knuth, Donald E. 1984. “Literate Programming.” The Computer Journal 27 (2). British Computer Society: 97–111.

Lowndes, Julia S Stewart, Benjamin D Best, Courtney Scarborough, Jamie C Afflerbach, Melanie R Frazier, Casey C O’Hara, Ning Jiang, and Benjamin S Halpern. 2017. “Our Path to Better Science in Less Time Using Open Data Science Tools.” Nature Ecology & Evolution 1 (6). Nature Publishing Group.

## **2.2 Compile an R Markdown document**

The usual way to compile an R Markdown document is to click the Knit button as shown in Figure [2.1](https://bookdown.org/yihui/rmarkdown/basics.html#fig:hello-rmd), and the corresponding keyboard shortcut is Ctrl + Shift + K (Cmd + Shift + K on macOS). Under the hood, RStudio calls the function rmarkdown::render() to render the document in a new R session. Please note the emphasis here, which often confuses R Markdown users. Rendering an Rmd document in a new R session means that none of the objects in your current R session (e.g., those you created in your R console) are available to that session.[2](https://bookdown.org/yihui/rmarkdown/compile.html#fn2) Reproducibility is the main reason that RStudio uses a new R session to render your Rmd documents: in most cases, you may want your documents to continue to work the next time you open R, or in other people’s computing environments. See [this StackOverflow answer](https://stackoverflow.com/a/48494678/559676) if you want to know more.

If you must render a document in the current R session, you can also call rmarkdown::render() by yourself, and pass the path of the Rmd file to this function. The second argument of this function is the output format, which defaults to the first output format you specify in the YAML metadata (if it is missing, the default is html\_document). When you have multiple output formats in the metadata, and do not want to use the first one, you can specify the one you want in the second argument, e.g., for an Rmd document foo.Rmd with the metadata:

output:

html\_document:

toc: true

pdf\_document:

keep\_tex: true

You can render it to PDF via:

rmarkdown::**render**('foo.Rmd', 'pdf\_document')

The function call gives you much more freedom (e.g., you can generate a series of reports in a loop), but you should bear reproducibility in mind when you render documents this way. Of course, you can start a new and clean R session by yourself, and call rmarkdown::render() in that session. As long as you do not manually interact with that session (e.g., manually creating variables in the R console), your reports should be reproducible.

Another main way to work with Rmd documents is the R Markdown Notebooks, which will be introduced in Section [3.2](https://bookdown.org/yihui/rmarkdown/notebook.html#notebook). With notebooks, you can run code chunks individually and see results right inside the RStudio editor. This is a convenient way to interact or experiment with code in an Rmd document, because you do not have to compile the whole document. Without using the notebooks, you can still partially execute code chunks, but the execution only occurs in the R console, and the notebook interface presents results of code chunks right beneath the chunks in the editor, which can be a great advantage. Again, for the sake of reproducibility, you will need to compile the whole document eventually in a clean environment.

Lastly, I want to mention an “unofficial” way to compile Rmd documents: the function xaringan::inf\_mr(), or equivalently, the RStudio addin “Infinite Moon Reader”. Obviously, this requires you to install the **xaringan** package (Xie [2019f](https://bookdown.org/yihui/rmarkdown/compile.html#ref-R-xaringan)), which is available on CRAN. The main advantage of this way is LiveReload: a technology that enables you to live preview the output as soon as you save the source document, and you do not need to hit the Knit button. The other advantage is that it compiles the Rmd document in the current R session, which may or may not be what you desire. Note that this method only works for Rmd documents that output to HTML, including HTML documents and presentations.

A few R Markdown extension packages, such as **bookdown** and **blogdown**, have their own way of compiling documents, and we will introduce them later.

Note that it is also possible to render a series of reports instead of single one from a single R Markdown source document. You can parameterize an R Markdown document, and generate different reports using different parameters. See Chapter [15](https://bookdown.org/yihui/rmarkdown/parameterized-reports.html#parameterized-reports) for details.

### References

Xie, Yihui. 2019f. Xaringan: Presentation Ninja. [https://CRAN.R-project.org/package=xaringan](https://cran.r-project.org/package=xaringan).

1. This is not strictly true, but mostly true. You may save objects in your current R session to a file, e.g., .RData, and load it in a new R session.[↩](https://bookdown.org/yihui/rmarkdown/compile.html#fnref2)

## **2.3 Cheat sheets**

RStudio has created a large number of cheat sheets, including the one-page R Markdown cheat sheet, which are freely available at <https://www.rstudio.com/resources/cheatsheets/>. There is also a more detailed R Markdown reference guide. Both documents can be used as quick references after you become more familiar with R Markdown.

## **2.4 Output formats**

There are two types of output formats in the **rmarkdown** package: documents, and presentations. All available formats are listed below:

* beamer\_presentation
* github\_document
* html\_document
* ioslides\_presentation
* latex\_document
* md\_document
* odt\_document
* pdf\_document
* powerpoint\_presentation
* rtf\_document
* slidy\_presentation
* word\_document

We will document these output formats in detail in Chapters [3](https://bookdown.org/yihui/rmarkdown/documents.html#documents) and [4](https://bookdown.org/yihui/rmarkdown/presentations.html#presentations). There are more output formats provided in other extension packages (starting from Chapter [5](https://bookdown.org/yihui/rmarkdown/dashboards.html#dashboards)). For the output format names in the YAML metadata of an Rmd file, you need to include the package name if a format is from an extension package, e.g.,

output: tufte::tufte\_html

If the format is from the **rmarkdown** package, you do not need the rmarkdown:: prefix (although it will not hurt).

When there are multiple output formats in a document, there will be a dropdown menu behind the RStudio Knit button that lists the output format names (Figure [2.4](https://bookdown.org/yihui/rmarkdown/output-formats.html#fig:format-dropdown)).

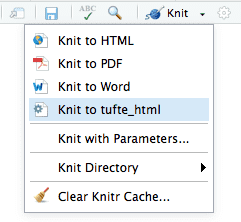


FIGURE 2.4: The output formats listed in the dropdown menu on the RStudio toolbar.

Each output format is often accompanied with several format options. All these options are documented on the R package help pages. For example, you can type ?rmarkdown::html\_document in R to open the help page of the html\_document format. When you want to use certain options, you have to translate the values from R to YAML, e.g.,

**html\_document**(toc = TRUE, toc\_depth = 2, dev = 'svg')

can be written in YAML as:

output:

html\_document:

toc: true

toc\_depth: 2

dev: 'svg'

The translation is often straightforward. Remember that R’s TRUE, FALSE, and NULL are true, false, and null, respectively, in YAML. Character strings in YAML often do not require the quotes (e.g., dev: 'svg' and dev: svg are the same), unless they contain special characters, such as the colon :. If you are not sure if a string should be quoted or not, test it with the **yaml** package, e.g.,

**cat**(yaml::**as.yaml**(**list**(

title = 'A Wonderful Day',

subtitle = 'hygge: a quality of coziness'

)))

title: A Wonderful Day

subtitle: 'hygge: a quality of coziness'

Note that the subtitle in the above example is quoted because of the colon.

If you have options that need to be the result of an evaluated R expression, you can use !expr, which tells the **yaml** package that it needs to parse and evaluate that option. Below is an example that uses a random theme for the HTML output:

output:

html\_document:

theme: !expr sample(c("yeti", "united", "lumen"), 1)

If a certain option has sub-options (which means the value of this option is a list in R), the sub-options need to be further indented, e.g.,

output:

html\_document:

toc: true

includes:

in\_header: header.html

before\_body: before.html

Some options are passed to **knitr**, such as dev, fig\_width, and fig\_height. Detailed documentation of these options can be found on the **knitr** documentation page: <https://yihui.name/knitr/options/>. Note that the actual **knitr** option names can be different. In particular, **knitr** uses . in names, but **rmarkdown** uses \_, e.g., fig\_width in **rmarkdown** corresponds to fig.width in **knitr**. We apologize for the inconsistencies—programmers often strive for consistencies in their own world, yet one standard plus one standard [often equals three standards.](https://xkcd.com/927/) If I were to design the **knitr** package again, I would definitely use \_.

Some options are passed to Pandoc, such as toc, toc\_depth, and number\_sections. You should consult the Pandoc documentation when in doubt. R Markdown output format functions often have a pandoc\_args argument, which should be a character vector of extra arguments to be passed to Pandoc. If you find any Pandoc features that are not represented by the output format arguments, you may use this ultimate argument, e.g.,

output:

pdf\_document:

toc: true

pandoc\_args: **[**"--wrap=none"**,** "--top-level-division=chapter"**]**

## **2.5 Markdown syntax**

The text in an R Markdown document is written with the Markdown syntax. Precisely speaking, it is Pandoc’s Markdown. There are many flavors of Markdown invented by different people, and Pandoc’s flavor is the most comprehensive one to our knowledge. You can find the full documentation of Pandoc’s Markdown at <https://pandoc.org/MANUAL.html>. We strongly recommend that you read this page at least once to know all the possibilities with Pandoc’s Markdown, even if you will not use all of them. This section is adapted from [Section 2.1](https://bookdown.org/yihui/bookdown/markdown-syntax.html) of Xie ([2016](https://bookdown.org/yihui/rmarkdown/markdown-syntax.html#ref-xie2016)), and only covers a small subset of Pandoc’s Markdown syntax.

### 2.5.1 Inline formatting

Inline text will be italic if surrounded by underscores or asterisks, e.g., \_text\_ or \*text\*. **Bold** text is produced using a pair of double asterisks (\*\*text\*\*). A pair of tildes (~) turn text to a subscript (e.g., H~3~PO~4~ renders H3PO4). A pair of carets (^) produce a superscript (e.g., Cu^2+^ renders Cu2+).

To mark text as inline code, use a pair of backticks, e.g., `code`. To include nn literal backticks, use at least n+1n+1 backticks outside, e.g., you can use four backticks to preserve three backtick inside: ```` ```code``` ````, which is rendered as ```code```.

Hyperlinks are created using the syntax [text](link), e.g., [RStudio](https://www.rstudio.com). The syntax for images is similar: just add an exclamation mark, e.g., ![alt text or image title](path/to/image). Footnotes are put inside the square brackets after a caret ^[], e.g., ^[This is a footnote.].

There are multiple ways to insert citations, and we recommend that you use BibTeX databases, because they work better when the output format is LaTeX/PDF. [Section 2.8](https://bookdown.org/yihui/bookdown/citations.html) of Xie ([2016](https://bookdown.org/yihui/rmarkdown/markdown-syntax.html#ref-xie2016)) has explained the details. The key idea is that when you have a BibTeX database (a plain-text file with the conventional filename extension .bib) that contains entries like:

@Manual{R-base,

title = {R: A Language and Environment for Statistical

Computing},

author = {{R Core Team}},

organization = {R Foundation for Statistical Computing},

address = {Vienna, Austria},

year = {2017},

url = {https://www.R-project.org/},

}

You may add a field named bibliography to the YAML metadata, and set its value to the path of the BibTeX file. Then in Markdown, you may use @R-base (which generates “R Core Team ([2019](https://bookdown.org/yihui/rmarkdown/markdown-syntax.html#ref-R-base))”) or [@R-base] (which generates “(R Core Team [2019](https://bookdown.org/yihui/rmarkdown/markdown-syntax.html#ref-R-base))”) to reference the BibTeX entry. Pandoc will automatically generated a list of references in the end of the document.

### 2.5.2 Block-level elements

Section headers can be written after a number of pound signs, e.g.,

# First-level header

## Second-level header

### Third-level header

If you do not want a certain heading to be numbered, you can add {-} or {.unnumbered} after the heading, e.g.,

# Preface {-}

Unordered list items start with \*, -, or +, and you can nest one list within another list by indenting the sub-list, e.g.,

- one item

- one item

- one item

- one more item

- one more item

- one more item

The output is:

* one item
* one item
* one item
  + one more item
  + one more item
  + one more item

Ordered list items start with numbers (you can also nest lists within lists), e.g.,

1. the first item

2. the second item

3. the third item

- one unordered item

- one unordered item

The output does not look too much different with the Markdown source:

1. the first item
2. the second item
3. the third item
   * one unordered item
   * one unordered item

Blockquotes are written after >, e.g.,

> "I thoroughly disapprove of duels. If a man should challenge me,

I would take him kindly and forgivingly by the hand and lead him

to a quiet place and kill him."

>

> --- Mark Twain

The actual output (we customized the style for blockquotes in this book):

“I thoroughly disapprove of duels. If a man should challenge me, I would take him kindly and forgivingly by the hand and lead him to a quiet place and kill him.”

— Mark Twain

Plain code blocks can be written after three or more backticks, and you can also indent the blocks by four spaces, e.g.,

```

This text is displayed verbatim / preformatted

```

Or indent by four spaces:

This text is displayed verbatim / preformatted

In general, you’d better leave at least one empty line between adjacent but different elements, e.g., a header and a paragraph. This is to avoid ambiguity to the Markdown renderer. For example, does “#” indicate a header below?

In R, the character

# indicates a comment.

And does “-” mean a bullet point below?

The result of 5

- 3 is 2.

Different flavors of Markdown may produce different results if there are no blank lines.

### 2.5.3 Math expressions

Inline LaTeX equations can be written in a pair of dollar signs using the LaTeX syntax, e.g., $f(k) = {n \choose k} p^{k} (1-p)^{n-k}$ (actual output: f(k)=(nk)pk(1−p)n−kf(k)=(nk)pk(1−p)n−k); math expressions of the display style can be written in a pair of double dollar signs, e.g., $$f(k) = {n \choose k} p^{k} (1-p)^{n-k}$$, and the output looks like this:

f(k)=(nk)pk(1−p)n−kf(k)=(nk)pk(1−p)n−k

You can also use math environments inside $ $ or $$ $$, e.g.,

$$\begin{array}{ccc}

x\_{11} & x\_{12} & x\_{13}\\

x\_{21} & x\_{22} & x\_{23}

\end{array}$$

x11x12x13x21x22x23x11x12x13x21x22x23

$$X = \begin{bmatrix}1 & x\_{1}\\

1 & x\_{2}\\

1 & x\_{3}

\end{bmatrix}$$

X=⎡⎢⎣1x11x21x3⎤⎥⎦X=[1x11x21x3]

$$\Theta = \begin{pmatrix}\alpha & \beta\\

\gamma & \delta

\end{pmatrix}$$

Θ=(αβγδ)Θ=(αβγδ)

$$\begin{vmatrix}a & b\\

c & d

\end{vmatrix}=ad-bc$$

∣∣∣abcd∣∣∣=ad−bc|abcd|=ad−bc

### References

R Core Team. 2019. R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing. [https://www.R-project.org/](https://www.r-project.org/).

Xie, Yihui. 2016. Bookdown: Authoring Books and Technical Documents with R Markdown. Boca Raton, Florida: Chapman; Hall/CRC. <https://github.com/rstudio/bookdown>.

## **2.6 R code chunks and inline R code**

You can insert an R code chunk either using the RStudio toolbar (the Insert button) or the keyboard shortcut Ctrl + Alt + I (Cmd + Option + I on macOS).

There are a lot of things you can do in a code chunk: you can produce text output, tables, or graphics. You have fine control over all these output via chunk options, which can be provided inside the curly braces (between ```{r and }). For example, you can choose hide text output via the chunk option results = 'hide', or set the figure height to 4 inches via fig.height = 4. Chunk options are separated by commas, e.g.,

```{r, chunk-label, results='hide', fig.height=4}

The value of a chunk option can be an arbitrary R expression, which makes chunk options extremely flexible. For example, the chunk option eval controls whether to evaluate (execute) a code chunk, and you may conditionally evaluate a chunk via a variable defined previously, e.g.,

```{r}

# execute code if the date is later than a specified day

do\_it = Sys.Date() > '2018-02-14'

```

```{r, eval=do\_it}

x = rnorm(100)

```

There are a large number of chunk options in **knitr** documented at <https://yihui.name/knitr/options>. We list a subset of them below:

* eval: Whether to evaluate a code chunk.
* echo: Whether to echo the source code in the output document (someone may not prefer reading your smart source code but only results).
* results: When set to 'hide', text output will be hidden; when set to 'asis', text output is written “as-is”, e.g., you can write out raw Markdown text from R code (like cat('\*\*Markdown\*\* is cool.\n')). By default, text output will be wrapped in verbatim elements (typically plain code blocks).
* collapse: Whether to merge text output and source code into a single code block in the output. This is mostly cosmetic: collapse = TRUE makes the output more compact, since the R source code and its text output are displayed in a single output block. The default collapse = FALSE means R expressions and their text output are separated into different blocks.
* warning, message, and error: Whether to show warnings, messages, and errors in the output document. Note that if you set error = FALSE, rmarkdown::render() will halt on error in a code chunk, and the error will be displayed in the R console. Similarly, when warning = FALSE or message = FALSE, these messages will be shown in the R console.
* include: Whether to include anything from a code chunk in the output document. When include = FALSE, this whole code chunk is excluded in the output, but note that it will still be evaluated if eval = TRUE. When you are trying to set echo = FALSE, results = 'hide', warning = FALSE, and message = FALSE, chances are you simply mean a single option include = FALSE instead of suppressing different types of text output individually.
* cache: Whether to enable caching. If caching is enabled, the same code chunk will not be evaluated the next time the document is compiled (if the code chunk was not modified), which can save you time. However, I want to honestly remind you of the two hard problems in computer science (via Phil Karlton): naming things, and cache invalidation. Caching can be handy but also tricky sometimes.
* fig.width and fig.height: The (graphical device) size of R plots in inches. R plots in code chunks are first recorded via a graphical device in **knitr**, and then written out to files. You can also specify the two options together in a single chunk option fig.dim, e.g., fig.dim = c(6, 4) means fig.width = 6 and fig.height = 4.
* out.width and out.height: The output size of R plots in the output document. These options may scale images. You can use percentages, e.g., out.width = '80%' means 80% of the page width.
* fig.align: The alignment of plots. It can be 'left', 'center', or 'right'.
* dev: The graphical device to record R plots. Typically it is 'pdf' for LaTeX output, and 'png' for HTML output, but you can certainly use other devices, such as 'svg' or 'jpeg'.
* fig.cap: The figure caption.
* child: You can include a child document in the main document. This option takes a path to an external file.

Chunk options in **knitr** can be surprisingly powerful. For example, you can create animations from a series of plots in a code chunk. I will not explain how here because [it requires an external software package](https://blogdown-demo.rbind.io/2018/01/31/gif-animations/), but encourage you to read the documentation carefully to discover the possibilities. You may also read Xie ([2015](https://bookdown.org/yihui/rmarkdown/r-code.html#ref-xie2015)), which is a comprehensive guide to the **knitr** package, but unfortunately biased towards LaTeX users for historical reasons (which was one of the reasons why I wanted to write this R Markdown book).

There is an optional chunk option that does not take any value, which is the chunk label. It should be the first option in the chunk header. Chunk labels are mainly used in filenames of plots and cache. If the label of a chunk is missing, a default one of the form unnamed-chunk-i will be generated, where i is incremental. I strongly recommend that you only use alphanumeric characters (a-z, A-Z and 0-9) and dashes (-) in labels, because they are not special characters and will surely work for all output formats. Other characters, spaces and underscores in particular, may cause trouble in certain packages, such as **bookdown**.

If a certain option needs to be frequently set to a value in multiple code chunks, you can consider setting it globally in the first code chunk of your document, e.g.,

```{r, setup, include=FALSE}

knitr::opts\_chunk$set(fig.width = 8, collapse = TRUE)

```

Besides code chunks, you can also insert values of R objects inline in text. For example:

```{r}

x = 5 # radius of a circle

```

For a circle with the radius `r x`,

its area is `r pi \* x^2`.

### 2.6.1 Figures

By default, figures produced by R code will be placed immediately after the code chunk they were generated from. For example:

```{r}

plot(cars, pch = 18)

```

You can provide a figure caption using fig.cap in the chunk options. If the document output format supports the option fig\_caption: true (e.g., the output format rmarkdown::html\_document), the R plots will be placed into figure environments. In the case of PDF output, such figures will be automatically numbered. If you also want to number figures in other formats (such as HTML), please see the **bookdown** package in Chapter [12](https://bookdown.org/yihui/rmarkdown/books.html#books) (in particular, see Section [12.4.4](https://bookdown.org/yihui/rmarkdown/bookdown-output.html#a-single-document)).

PDF documents are generated through the LaTeX files generated from R Markdown. A highly surprising fact to LaTeX beginners is that figures float by default: even if you generate a plot in a code chunk on the first page, the whole figure environment may float to the next page. This is just how LaTeX works by default. It has a tendency to float figures to the top or bottom of pages. Although it can be annoying and distracting, we recommend that you refrain from playing the “Whac-A-Mole” game in the beginning of your writing, i.e., desparately trying to position figures “correctly” while they seem to be always dodging you. You may wish to fine-tune the positions once the content is complete using the fig.pos chunk option (e.g., fig.pos = 'h'). See <https://www.sharelatex.com/learn/Positioning_images_and_tables> for possible values of fig.pos and more general tips about this behavior in LaTeX. In short, this can be a difficult problem for PDF output.

To place multiple figures side-by-side from the same code chunk, you can use the fig.show='hold' option along with the out.width option. Figure [2.5](https://bookdown.org/yihui/rmarkdown/r-code.html#fig:hold-position) shows an example with two plots, each with a width of 50%.

**par**(mar = **c**(4, 4, 0.2, 0.1))

**plot**(cars, pch = 19)

**plot**(pressure, pch = 17)

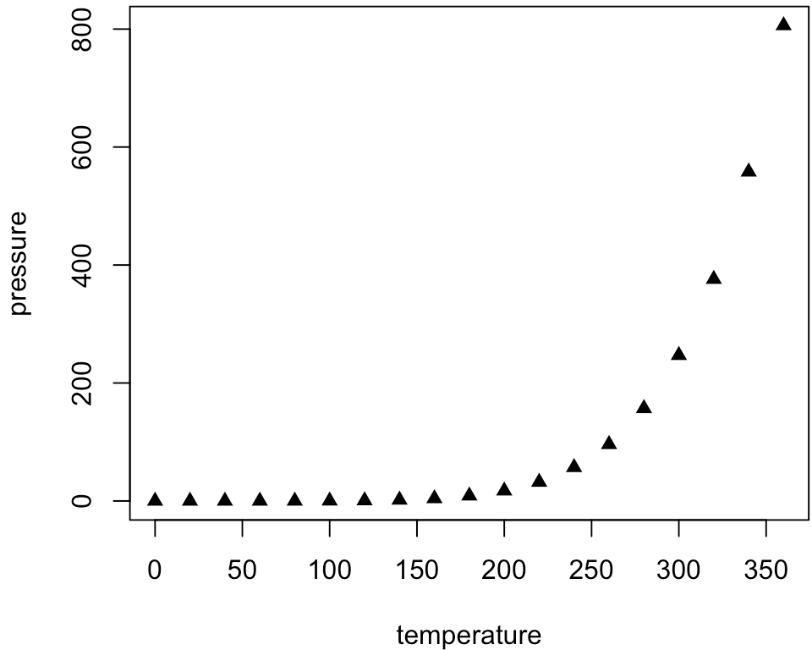
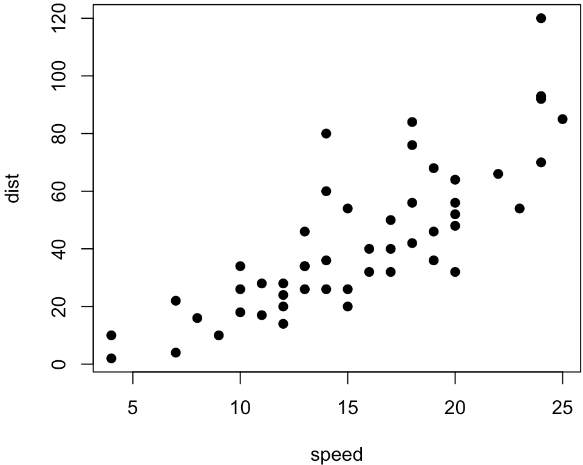


FIGURE 2.5: Two plots side-by-side.

If you want to include a graphic that is not generated from R code, you may use the knitr::include\_graphics() function, which gives you more control over the attributes of the image than the Markdown syntax of ![alt text or image title](path/to/image) (e.g., you can specify the image width via out.width). Figure [2.6](https://bookdown.org/yihui/rmarkdown/r-code.html#fig:include-graphics) provides an example of this.

```{r, out.width='25%', fig.align='center', fig.cap='...'}

knitr::include\_graphics('images/hex-rmarkdown.png')

```

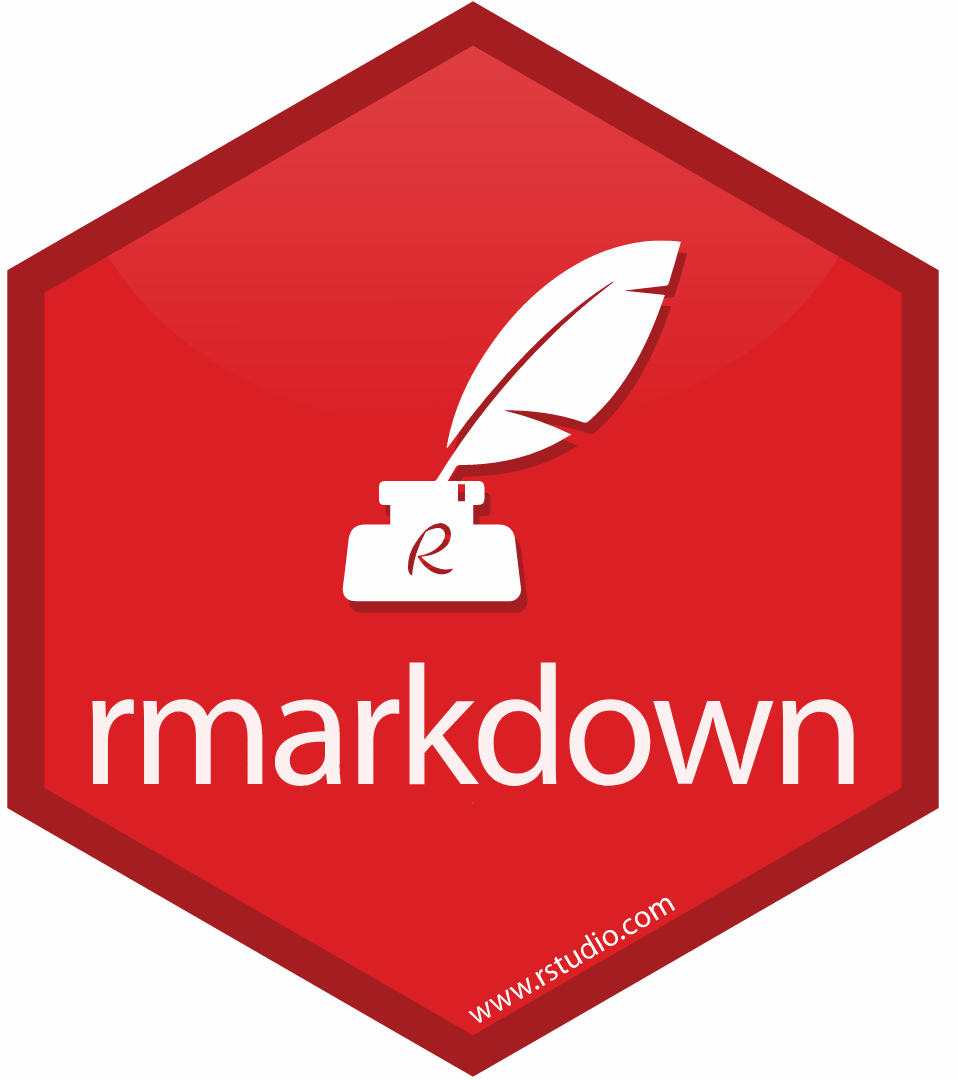


FIGURE 2.6: The R Markdown hex logo.

### 2.6.2 Tables

The easiest way to include tables is by using knitr::kable(), which can create tables for HTML, PDF and Word outputs.[3](https://bookdown.org/yihui/rmarkdown/r-code.html#fn3) Table captions can be included by passing caption to the function, e.g.,

```{r tables-mtcars}

knitr::kable(iris[1:5, ], caption = 'A caption')

```

Tables in non-LaTeX output formats will always be placed after the code block. For LaTeX/PDF output formats, tables have the same issue as figures: they may float. If you want to avoid this behavior, you will need to use the LaTeX package [longtable](https://www.ctan.org/pkg/longtable), which can break tables across multiple pages. This can be achieved by adding \usepackage{longtable} to your LaTeX preamble, and passing longtable = TRUE to kable().

If you are looking for more advanced control of the styling of tables, you are recommended to use the [**kableExtra**](https://cran.r-project.org/package=kableExtra) package, which provides functions to customize the appearance of PDF and HTML tables. Formatting tables can be a very complicated task, especially when certain cells span more than one column or row. It is even more complicated when you have to consider different output formats. For example, it is difficult to make a complex table work for both PDF and HTML output. We know it is disappointing, but sometimes you may have to consider alternative ways of presenting data, such as using graphics.

We explain in Section [12.3](https://bookdown.org/yihui/rmarkdown/bookdown-markdown.html#bookdown-markdown) how the **bookdown** package extends the functionality of **rmarkdown** to allow for figures and tables to be easily cross-referenced within your text.

### References

Xie, Yihui. 2015. Dynamic Documents with R and Knitr. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. <https://yihui.name/knitr/>.

1. You may also consider the **pander** package. There are several other packages for producing tables, including **xtable**, **Hmisc**, and **stargazer**, but these are generally less compatible with multiple output formats.[↩](https://bookdown.org/yihui/rmarkdown/r-code.html#fnref3)

## **2.8 Interactive documents**

R Markdown documents can also generate interactive content. There are two types of interactive R Markdown documents: you can use the HTML Widgets framework, or the Shiny framework (or both). They will be described in more detail in Chapter [16](https://bookdown.org/yihui/rmarkdown/html-widgets.html#html-widgets) and Chapter [19](https://bookdown.org/yihui/rmarkdown/shiny-documents.html#shiny-documents), respectively.

### 2.8.1 HTML widgets

The HTML Widgets framework is implemented in the R package  **htmlwidgets** (Vaidyanathan et al. [2019](https://bookdown.org/yihui/rmarkdown/interactive-documents.html#ref-R-htmlwidgets)), interfacing JavaScript libraries that create interactive applications, such as interactive graphics and tables. Several widget packages have been developed based on this framework, such as **DT** (Xie, Cheng, and Tan [2019](https://bookdown.org/yihui/rmarkdown/interactive-documents.html#ref-R-DT)), **leaflet** (Cheng, Karambelkar, and Xie [2019](https://bookdown.org/yihui/rmarkdown/interactive-documents.html#ref-R-leaflet)), and **dygraphs** (Vanderkam et al. [2018](https://bookdown.org/yihui/rmarkdown/interactive-documents.html#ref-R-dygraphs)). Visit [https://www.htmlwidgets.org](https://www.htmlwidgets.org/) to know more about widget packages as well as how to develop a widget package by yourself.

Figure [2.7](https://bookdown.org/yihui/rmarkdown/interactive-documents.html#fig:leaflet) shows an interactive map created via the **leaflet** package, and the source document is below:

---

title: "An Interactive Map"

---

Below is a map that shows the location of the

Department of Statistics, Iowa State University.

```{r out.width='100%', echo=FALSE}

library(leaflet)

leaflet() %>% addTiles() %>%

setView(-93.65, 42.0285, zoom = 17) %>%

addPopups(

-93.65, 42.0285,

'Here is the <b>Department of Statistics</b>, ISU'

)

```

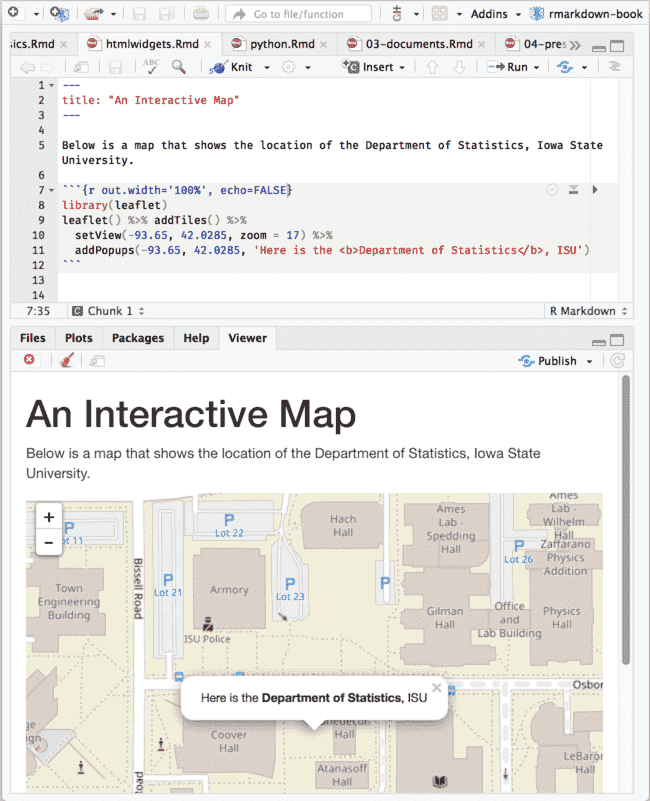


FIGURE 2.7: An R Markdown document with a leaflet map widget.

Although HTML widgets are based on JavaScript, the syntax to create them in R is often pure R syntax.

If you include an HTML widget in a non-HTML output format, such as a PDF, **knitr** will try to embed a screenshot of the widget if you have installed the R package **webshot** (Chang [2018](https://bookdown.org/yihui/rmarkdown/interactive-documents.html#ref-R-webshot)) and the PhantomJS package (via webshot::install\_phantomjs()).

### 2.8.2 Shiny documents

The **shiny** package (Chang et al. [2019](https://bookdown.org/yihui/rmarkdown/interactive-documents.html#ref-R-shiny)) builds interactive web apps powered by R. To call Shiny code from an R Markdown document, add runtime: shiny to the YAML metadata, like in this document:

---

title: "A Shiny Document"

output: html\_document

runtime: shiny

---

A standard R plot can be made interactive by wrapping

it in the Shiny `renderPlot()` function. The `selectInput()`

function creates the input widget to drive the plot.

```{r eruptions, echo=FALSE}

selectInput(

'breaks', label = 'Number of bins:',

choices = c(10, 20, 35, 50), selected = 20

)

renderPlot({

par(mar = c(4, 4, .1, .5))

hist(

faithful$eruptions, as.numeric(input$breaks),

col = 'gray', border = 'white',

xlab = 'Duration (minutes)', main = ''

)

})

```

Figure [2.8](https://bookdown.org/yihui/rmarkdown/interactive-documents.html#fig:shiny) shows the output, where you can see a dropdown menu that allows you to choose the number of bins in the histogram.

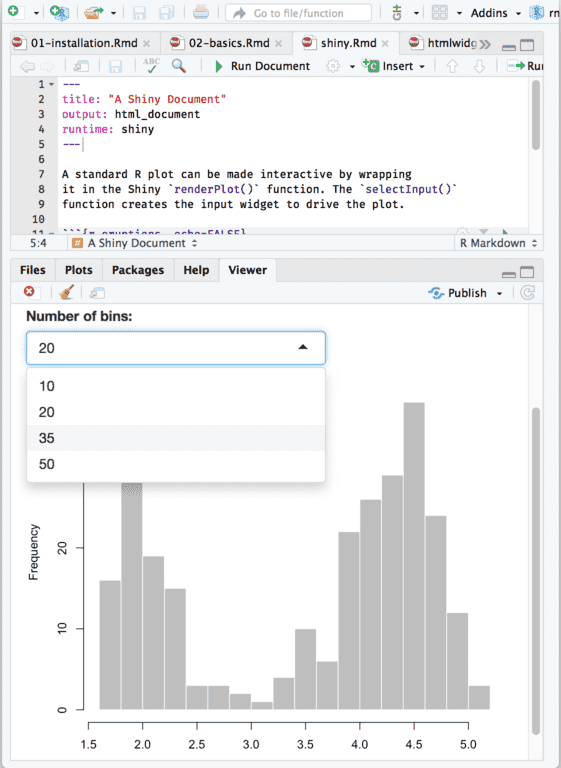


FIGURE 2.8: An R Markdown document with a Shiny widget.

You may use Shiny to run any R code that you like in response to user actions. Since web browsers cannot execute R code, Shiny interactions occur on the server side and rely on a live R session. By comparison, HTML widgets do not require a live R session to support them, because the interactivity comes from the client side (via JavaScript in the web browser).

You can learn more about Shiny at [https://shiny.rstudio.com](https://shiny.rstudio.com/).

HTML widgets and Shiny elements rely on HTML and JavaScript. They will work in any R Markdown format that is viewed in a web browser, such as HTML documents, dashboards, and HTML5 presentations.

### References

Chang, Winston. 2018. Webshot: Take Screenshots of Web Pages. [https://CRAN.R-project.org/package=webshot](https://cran.r-project.org/package=webshot).

Chang, Winston, Joe Cheng, JJ Allaire, Yihui Xie, and Jonathan McPherson. 2019. Shiny: Web Application Framework for R. [https://CRAN.R-project.org/package=shiny](https://cran.r-project.org/package=shiny).

Cheng, Joe, Bhaskar Karambelkar, and Yihui Xie. 2019. Leaflet: Create Interactive Web Maps with the Javascript ’Leaflet’ Library. [https://CRAN.R-project.org/package=leaflet](https://cran.r-project.org/package=leaflet).

Vaidyanathan, Ramnath, Yihui Xie, JJ Allaire, Joe Cheng, and Kenton Russell. 2019. Htmlwidgets: HTML Widgets for R. [https://CRAN.R-project.org/package=htmlwidgets](https://cran.r-project.org/package=htmlwidgets).

Vanderkam, Dan, JJ Allaire, Jonathan Owen, Daniel Gromer, and Benoit Thieurmel. 2018. Dygraphs: Interface to ’Dygraphs’ Interactive Time Series Charting Library. [https://CRAN.R-project.org/package=dygraphs](https://cran.r-project.org/package=dygraphs).

Xie, Yihui, Joe Cheng, and Xianying Tan. 2019. DT: A Wrapper of the Javascript Library ’Datatables’. <https://github.com/rstudio/DT>.

# Chapter 3 Documents

The very original version of Markdown was invented mainly to write HTML content more easily. For example, you can write a bullet with - text instead of the verbose HTML code <ul><li>text</li></ul>, or a quote with > text instead of <blockquote>text</blockquote>.

The syntax of Markdown has been greatly extended by Pandoc. What is more, Pandoc makes it possible to convert a Markdown document to a large variety of output formats. In this chapter, we will introduce the features of various document output formats. In the next two chapters, we will document the presentation formats and other R Markdown extensions, respectively.

## **3.1 HTML document**

As we just mentioned before, Markdown was originally designed for HTML output, so it may not be surprising that the HTML format has the richest features among all output formats. We recommend that you read this full section before you learn other output formats, because other formats have several features in common with the HTML document format, and we will not repeat these features in the corresponding sections.

To create an HTML document from R Markdown, you specify the html\_document output format in the YAML metadata of your document:

---

title: Habits

author: John Doe

date: March 22, 2005

output: html\_document

---

### 3.1.1 Table of contents

You can add a table of contents (TOC) using the toc option and specify the depth of headers that it applies to using the toc\_depth option. For example:

---

title: "Habits"

output:

html\_document:

toc: true

toc\_depth: 2

---

If the table of contents depth is not explicitly specified, it defaults to 3 (meaning that all level 1, 2, and 3 headers will be included in the table of contents).

#### **3.1.1.1 Floating TOC**

You can specify the toc\_float option to float the table of contents to the left of the main document content. The floating table of contents will always be visible even when the document is scrolled. For example:

---

title: "Habits"

output:

html\_document:

toc: true

toc\_float: true

---

You may optionally specify a list of options for the toc\_float parameter which control its behavior. These options include:

* collapsed (defaults to TRUE) controls whether the TOC appears with only the top-level (e.g., H2) headers. If collapsed initially, the TOC is automatically expanded inline when necessary.
* smooth\_scroll (defaults to TRUE) controls whether page scrolls are animated when TOC items are navigated to via mouse clicks.

For example:

---

title: "Habits"

output:

html\_document:

toc: true

toc\_float:

collapsed: false

smooth\_scroll: false

---

### 3.1.2 Section numbering

You can add section numbering to headers using the number\_sections option:

---

title: "Habits"

output:

html\_document:

toc: true

number\_sections: true

---

Note that if you do choose to use the number\_sections option, you will likely also want to use # (H1) headers in your document as ## (H2) headers will include a decimal point, because without H1 headers, you H2 headers will be numbered with 0.1, 0.2, and so on.

### 3.1.3 Tabbed sections

You can organize content using tabs by applying the .tabset class attribute to headers within a document. This will cause all sub-headers of the header with the .tabset attribute to appear within tabs rather than as standalone sections. For example:

## Quarterly Results {.tabset}

### By Product

(tab content)

### By Region

(tab content)

You can also specify two additional attributes to control the appearance and behavior of the tabs. The .tabset-fade attribute causes the tabs to fade in and out when switching between tabs. The .tabset-pills attribute causes the visual appearance of the tabs to be “pill” (see Figure [3.1](https://bookdown.org/yihui/rmarkdown/html-document.html#fig:tabset)) rather than traditional tabs. For example:

## Quarterly Results {.tabset .tabset-fade .tabset-pills}

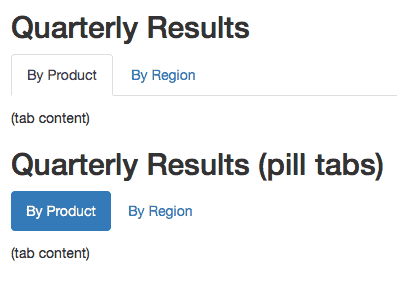


FIGURE 3.1: Traditional tabs and pill tabs on an HTML page.

### 3.1.4 Appearance and style

There are several options that control the appearance of HTML documents:

* theme specifies the Bootstrap theme to use for the page (themes are drawn from the [Bootswatch](https://bootswatch.com/3/) theme library). Valid themes include default, cerulean, journal, flatly, darkly, readable, spacelab, united, cosmo, lumen, paper, sandstone, simplex, and yeti. Pass null for no theme (in this case you can use the css parameter to add your own styles).
* highlight specifies the syntax highlighting style. Supported styles include default, tango, pygments, kate, monochrome, espresso, zenburn, haddock, breezedark, and textmate. Pass null to prevent syntax highlighting.
* smart indicates whether to produce typographically correct output, converting straight quotes to curly quotes, --- to em-dashes, -- to en-dashes, and ... to ellipses. Note that smart is enabled by default.

For example:

---

title: "Habits"

output:

html\_document:

theme: united

highlight: tango

---

#### **3.1.4.1 Custom CSS**

You can add your own CSS to an HTML document using the css option:

---

title: "Habits"

output:

html\_document:

css: styles.css

---

If you want to provide all of the styles for the document from your own CSS you set the theme (and potentially highlight) to null:

---

title: "Habits"

output:

html\_document:

theme: null

highlight: null

css: styles.css

---

You can also target specific sections of documents with custom CSS by adding ids or classes to section headers within your document. For example the following section header:

## Next Steps {#nextsteps .emphasized}

Would enable you to apply CSS to all of its content using either of the following CSS selectors:

#nextsteps {

**color**: blue;

}

.emphasized {

**font-size**: 1.2em;

}

### 3.1.5 Figure options

There are a number of options that affect the output of figures within HTML documents:

* fig\_width and fig\_height can be used to control the default figure width and height (7x5 is used by default).
* fig\_retina specifies the scaling to perform for retina displays (defaults to 2, which currently works for all widely used retina displays). Set to null to prevent retina scaling.
* fig\_caption controls whether figures are rendered with captions.
* dev controls the graphics device used to render figures (defaults to png).

For example:

---

title: "Habits"

output:

html\_document:

fig\_width: 7

fig\_height: 6

fig\_caption: true

---

### 3.1.6 Data frame printing

You can enhance the default display of data frames via the df\_print option. Valid values are shown in Table [3.1](https://bookdown.org/yihui/rmarkdown/html-document.html#tab:df-print).

| TABLE 3.1: The possible values of the df\_print option for the html\_document format. | |
| --- | --- |
| **Option** | **Description** |
| default | Call the print.data.frame generic method |
| kable | Use the knitr::kable function |
| tibble | Use the tibble::print.tbl\_df function |
| paged | Use rmarkdown::paged\_table to create a pageable table |

#### **3.1.6.1 Paged printing**

When the df\_print option is set to paged, tables are printed as HTML tables with support for pagination over rows and columns. For instance (see Figure [3.2](https://bookdown.org/yihui/rmarkdown/html-document.html#fig:paged)):

---

title: "Motor Trend Car Road Tests"

output:

html\_document:

df\_print: paged

---

```{r}

mtcars

```



FIGURE 3.2: A paged table in the HTML output document.

Table [3.2](https://bookdown.org/yihui/rmarkdown/html-document.html#tab:paged) shows the available options for paged tables.

| TABLE 3.2: The options for paged HTML tables. | |
| --- | --- |
| **Option** | **Description** |
| max.print | The number of rows to print. |
| rows.print | The number of rows to display. |
| cols.print | The number of columns to display. |
| cols.min.print | The minimum number of columns to display. |
| pages.print | The number of pages to display under page navigation. |
| paged.print | When set to FALSE turns off paged tables. |
| rownames.print | When set to FALSE turns off row names. |

These options are specified in each chunk like below:

```{r cols.print=3, rows.print=3}

mtcars

```

### 3.1.7 Code folding

When the **knitr** chunk option echo = TRUE is specified (the default behavior), the R source code within chunks is included within the rendered document. In some cases, it may be appropriate to exclude code entirely (echo = FALSE) but in other cases you might want the code to be available but not visible by default.

The code\_folding: hide option enables you to include R code but have it hidden by default. Users can then choose to show hidden R code chunks either individually or document wide. For example:

---

title: "Habits"

output:

html\_document:

code\_folding: hide

---

You can specify code\_folding: show to still show all R code by default but then allow users to hide the code if they wish.

### 3.1.8 MathJax equations

By default, [MathJax](https://www.mathjax.org/) scripts are included in HTML documents for rendering LaTeX and MathML equations. You can use the mathjax option to control how MathJax is included:

* Specify "default" to use an HTTPS URL from a CDN host (currently provided by RStudio).
* Specify "local" to use a local version of MathJax (which is copied into the output directory). Note that when using "local" you also need to set the self\_contained option to false.
* Specify an alternate URL to load MathJax from another location.
* Specify null to exclude MathJax entirely.

For example, to use a local copy of MathJax:

---

title: "Habits"

output:

html\_document:

mathjax: local

self\_contained: false

---

To use a self-hosted copy of MathJax:

---

title: "Habits"

output:

html\_document:

mathjax: "http://example.com/MathJax.js"

---

To exclude MathJax entirely:

---

title: "Habits"

output:

html\_document:

mathjax: null

---

### 3.1.9 Document dependencies

By default, R Markdown produces standalone HTML files with no external dependencies, using data: URIs to incorporate the contents of linked scripts, stylesheets, images, and videos. This means you can share or publish the file just like you share Office documents or PDFs. If you would rather keep dependencies in external files, you can specify self\_contained: false. For example:

---

title: "Habits"

output:

html\_document:

self\_contained: false

---

Note that even for self-contained documents, MathJax is still loaded externally (this is necessary because of its big size). If you want to serve MathJax locally, you should specify mathjax: local and self\_contained: false.

One common reason to keep dependencies external is for serving R Markdown documents from a website (external dependencies can be cached separately by browsers, leading to faster page load times). In the case of serving multiple R Markdown documents you may also want to consolidate dependent library files (e.g. Bootstrap, and MathJax, etc.) into a single directory shared by multiple documents. You can use the lib\_dir option to do this. For example:

---

title: "Habits"

output:

html\_document:

self\_contained: false

lib\_dir: libs

---

### 3.1.10 Advanced customization

#### **3.1.10.1 Keeping Markdown**

When **knitr** processes an R Markdown input file, it creates a Markdown (\*.md) file that is subsequently transformed into HTML by Pandoc. If you want to keep a copy of the Markdown file after rendering, you can do so using the keep\_md option:

---

title: "Habits"

output:

html\_document:

keep\_md: true

---

#### **3.1.10.2 Includes**

You can do more advanced customization of output by including additional HTML content or by replacing the core Pandoc template entirely. To include content in the document header or before/after the document body, you use the includes option as follows:

---

title: "Habits"

output:

html\_document:

includes:

in\_header: header.html

before\_body: doc\_prefix.html

after\_body: doc\_suffix.html

---

#### **3.1.10.3 Custom templates**

You can also replace the underlying Pandoc template using the template option:

---

title: "Habits"

output:

html\_document:

template: quarterly\_report.html

---

Consult the documentation on [Pandoc templates](http://pandoc.org/MANUAL.html#templates) for additional details on templates. You can also study the [default HTML template default.html5](https://github.com/jgm/pandoc-templates/) as an example.

#### **3.1.10.4 Markdown extensions**

By default, R Markdown is defined as all Pandoc Markdown extensions with the following tweaks for backward compatibility with the old **markdown** package (J. Allaire, Horner, et al. [2019](https://bookdown.org/yihui/rmarkdown/html-document.html#ref-R-markdown)):

+autolink\_bare\_uris

+tex\_math\_single\_backslash

You can enable or disable Markdown extensions using the md\_extensions option (you preface an option with - to disable and + to enable it). For example:

---

title: "Habits"

output:

html\_document:

md\_extensions: -autolink\_bare\_uris+hard\_line\_breaks

---

The above would disable the autolink\_bare\_uris extension, and enable the hard\_line\_breaks extension.

For more on available markdown extensions see the [Pandoc Markdown specification](http://pandoc.org/MANUAL.html#pandocs-markdown).

#### **3.1.10.5 Pandoc arguments**

If there are Pandoc features that you want to use but lack equivalents in the YAML options described above, you can still use them by passing custom pandoc\_args. For example:

---

title: "Habits"

output:

html\_document:

pandoc\_args: **[**

"--title-prefix"**,** "Foo"**,**

"--id-prefix"**,** "Bar"

**]**

---

Documentation on all available pandoc arguments can be found in the [Pandoc User Guide](http://pandoc.org/MANUAL.html#options).

### 3.1.11 Shared options

If you want to specify a set of default options to be shared by multiple documents within a directory, you can include a file named \_output.yml within the directory. Note that no YAML delimiters (---) or the enclosing output field are used in this file. For example:

html\_document:

self\_contained: false

theme: united

highlight: textmate

It should not be written as:

---

output:

html\_document:

self\_contained: false

theme: united

highlight: textmate

---

All documents located in the same directory as \_output.yml will inherit its options. Options defined explicitly within documents will override those specified in the shared options file.

### 3.1.12 HTML fragments

If want to create an HTML fragment rather than a full HTML document you can use the html\_fragment format. For example:

---

output: html\_fragment

---

Note that HTML fragments are not complete HTML documents. They do not contain the standard header content that HTML documents do (they only contain content in the <body> tags of normal HTML documents). They are intended for inclusion within other web pages or content management systems (like blogs). As such, they do not support features like themes or code highlighting (it is expected that the environment they are ultimately published within handles these things).

### References

Allaire, JJ, Jeffrey Horner, Yihui Xie, Vicent Marti, and Natacha Porte. 2019. Markdown: Render Markdown with the c Library ’Sundown’. [https://CRAN.R-project.org/package=markdown](https://cran.r-project.org/package=markdown).

## **3.2 Notebook**

An R Notebook is an R Markdown document with chunks that can be executed independently and interactively, with output visible immediately beneath the input. See Figure [3.3](https://bookdown.org/yihui/rmarkdown/notebook.html#fig:notebook-demo) for an example.

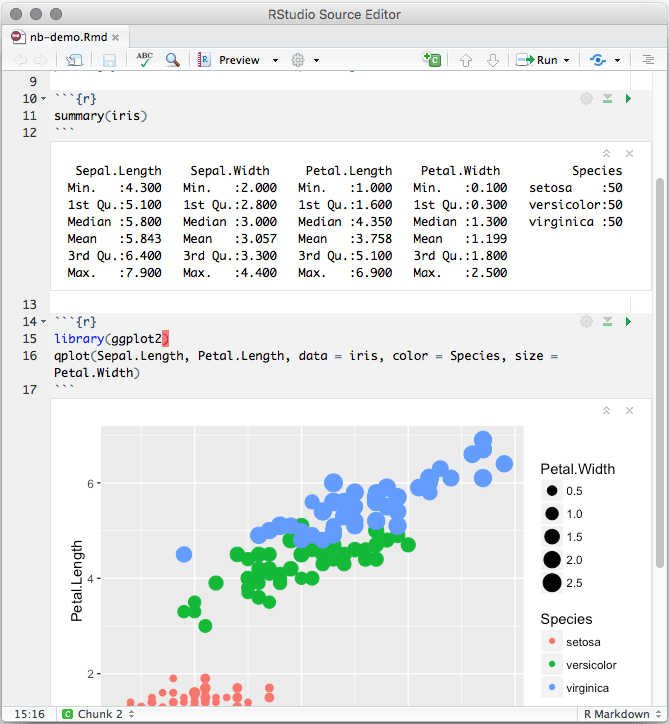


FIGURE 3.3: An R Notebook example.

R Notebooks are an implementation of [Literate Programming](https://en.wikipedia.org/wiki/Literate_programming) that allows for direct interaction with R while producing a reproducible document with publication-quality output.

Any R Markdown document can be used as a notebook, and all R Notebooks can be rendered to other R Markdown document types. A notebook can therefore be thought of as a special execution mode for R Markdown documents. The immediacy of notebook mode makes it a good choice while authoring the R Markdown document and iterating on code. When you are ready to publish the document, you can share the notebook directly, or render it to a publication format with the Knit button.

### 3.2.1 Using Notebooks

#### **3.2.1.1 Creating a Notebook**

You can create a new notebook in RStudio with the menu command File -> New File -> R Notebook, or by using the html\_notebook output type in your document’s YAML metadata.

---

title: "My Notebook"

output: html\_notebook

---

By default, RStudio enables inline output (Notebook mode) on all R Markdown documents, so you can interact with any R Markdown document as though it were a notebook. If you have a document with which you prefer to use the traditional console method of interaction, you can disable notebook mode by clicking the gear button in the editor toolbar, and choosing Chunk Output in Console (Figure [3.4](https://bookdown.org/yihui/rmarkdown/notebook.html#fig:notebook-console)).

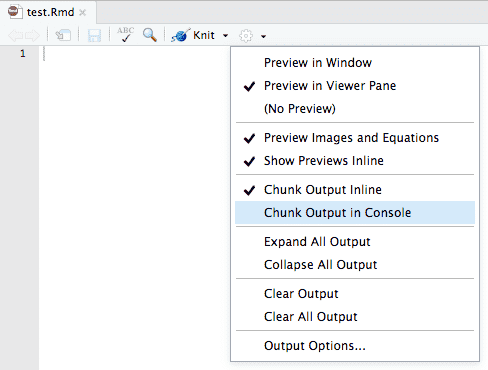


FIGURE 3.4: Send the R code chunk output to the console.

If you prefer to use the console by default for all your R Markdown documents (restoring the behavior in previous versions of RStudio), you can make Chunk Output in Console the default: Tools -> Options -> R Markdown -> Show output inline for all R Markdown documents.

#### **3.2.1.2 Inserting chunks**

Notebook chunks can be inserted quickly using the keyboard shortcut Ctrl + Alt + I (macOS: Cmd + Option + I), or via the Insert menu in the editor toolbar.

Because all of a chunk’s output appears beneath the chunk (not alongside the statement which emitted the output, as it does in the rendered R Markdown output), it is often helpful to split chunks that produce multiple outputs into two or more chunks which each produce only one output. To do this, select the code to split into a new chunk (Figure [3.5](https://bookdown.org/yihui/rmarkdown/notebook.html#fig:notebook-split-before)), and use the same keyboard shortcut for inserting a new code chunk (Figure [3.6](https://bookdown.org/yihui/rmarkdown/notebook.html#fig:notebook-split-after)).

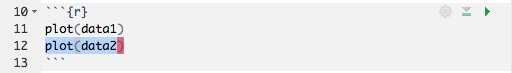


FIGURE 3.5: Select the code to split into a new chunk.

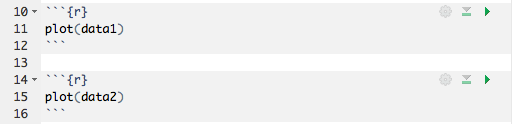


FIGURE 3.6: Insert a new chunk from the code selected before.

#### **3.2.1.3 Executing code**

Code in the notebook is executed with the same gestures you would use to execute code in an R Markdown document:

1. Use the green triangle button on the toolbar of a code chunk that has the tooltip “Run Current Chunk”, or Ctrl + Shift + Enter (macOS: Cmd + Shift + Enter) to run the current chunk.
2. Press Ctrl + Enter (macOS: Cmd + Enter) to run just the current statement. Running a single statement is much like running an entire chunk consisting only of that statement.
3. There are other ways to run a batch of chunks if you click the menu Run on the editor toolbar, such as Run All, Run All Chunks Above, and Run All Chunks Below.

The primary difference is that when executing chunks in an R Markdown document, all the code is sent to the console at once, but in a notebook, only one line at a time is sent. This allows execution to stop if a line raises an error.

There is also a Restart R and Run All Chunks item in the Run menu on the editor toolbar, which gives you a fresh R session prior to running all the chunks. This is similar to the Knit button, which launches a separate R session to compile the document.

When you execute code in a notebook, an indicator will appear in the gutter to show you execution progress (Figure [3.7](https://bookdown.org/yihui/rmarkdown/notebook.html#fig:notebook-longrunning)). Lines of code that have been sent to R are marked with dark green; lines that have not yet been sent to R are marked with light green. If at least one chunk is waiting to be executed, you will see a progress meter appear in the editor’s status bar, indicating the number of chunks remaining to be executed. You can click on this meter at any time to jump to the currently executing chunk. When a chunk is waiting to execute, the Run button in its toolbar will change to a “queued” icon. If you do not want the chunk to run, you can click on the icon to remove it from the execution queue.

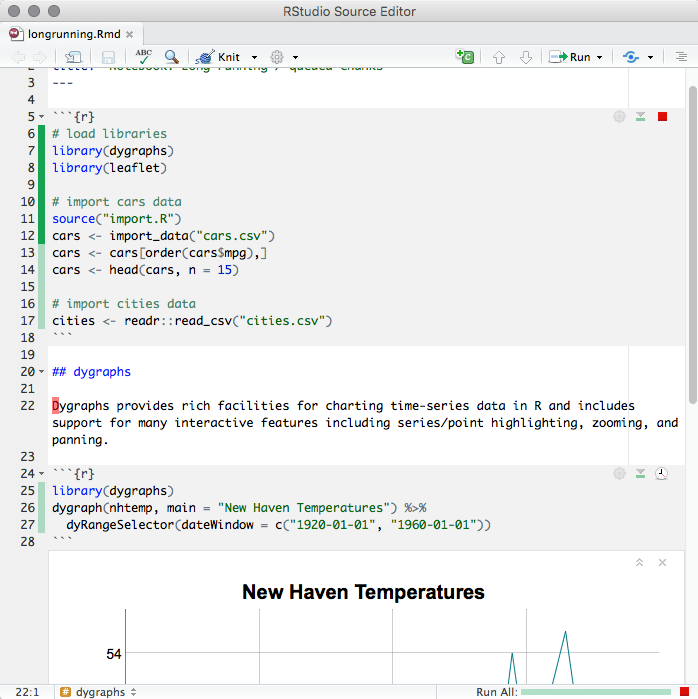


FIGURE 3.7: The indicator in the gutter to show the execution progress of a code chunk in the notebook.

In general, when you execute code in a notebook chunk, it will do exactly the same thing as it would if that same code were typed into the console. There are however a few differences:

* **Output**: The most obvious difference is that most forms of output produced from a notebook chunk are shown in the chunk output rather than, for example, the RStudio Viewer or the Plots pane. Console output (including warnings and messages) appears both at the console and in the chunk output.
* **Working directory**: The current working directory inside a notebook chunk is always the directory containing the notebook .Rmd file. This makes it easier to use relative paths inside notebook chunks, and also matches the behavior when knitting, making it easier to write code that works identically both interactively and in a standalone render.

You’ll get a warning if you try to change the working directory inside a notebook chunk, and the directory will revert back to the notebook’s directory once the chunk is finished executing. You can suppress this warning by using the warnings = FALSE chunk option.

If it is necessary to execute notebook chunks in a different directory, you can change the working directory for **all** your chunks by using the **knitr** root.dir option. For instance, to execute all notebook chunks in the grandparent folder of the notebook:

knitr::opts\_knit$**set**(root.dir = **normalizePath**(".."))

This option is only effective when used inside the setup chunk. Also note that, as in **knitr**, the root.dir chunk option applies only to chunks; relative paths in Markdown are still relative to the notebook’s parent folder.

* **Warnings**: Inside a notebook chunk, warnings are always displayed immediately rather than being held until the end, as in options(warn = 1).
* **Plots**: Plots emitted from a chunk are rendered to match the width of the editor at the time the chunk was executed. The height of the plot is determined by the [golden ratio](https://en.wikipedia.org/wiki/Golden_ratio). The plot’s display list is saved, too, and the plot is re-rendered to match the editor’s width when the editor is resized.

You can use the fig.width, fig.height, and fig.asp chunk options to manually specify the size of rendered plots in the notebook; you can also use knitr::opts\_chunk$set(fig.width = ..., fig.height = ...) in the setup chunk to to set a default rendered size. Note, however, specifying a chunk size manually suppresses the generation of the display list, so plots with manually specified sizes will be resized using simple image scaling when the notebook editor is resized.

To execute an inline R expression in the notebook, put your cursor inside the chunk and press Ctrl + Enter (macOS: Cmd + Enter). As in the execution of ordinary chunks, the content of the expression will be sent to the R console for evaluation. The results will appear in a small pop-up window next to the code (Figure [3.8](https://bookdown.org/yihui/rmarkdown/notebook.html#fig:notebook-inline-output)).

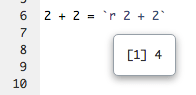


FIGURE 3.8: Output from an inline R expression in the notebook.

In notebooks, inline R expressions can only produce text (not figures or other kinds of output). It is also important that inline R expressions executes quickly and do not have side-effects, as they are executed whenever you save the notebook.

Notebooks are typically self-contained. However, in some situations, it is preferable to re-use code from an R script as a notebook chunk, as in **knitr**’s [code externalization](https://yihui.name/knitr/demo/externalization/). This can be done by using knitr::read\_chunk() in your notebook’s setup chunk, along with a special ## ---- chunkname annotation in the R file from which you intend to read code. Here is a minimal example with two files:

**example.Rmd**

```{r setup}

knitr::read\_chunk("example.R")

```

**example.R**

*## ---- chunk*

1 + 1

When you execute the empty chunk in the notebook example.Rmd, code from the external file example.R will be inserted, and the results displayed inline, as though the chunk contained that code (Figure [3.9](https://bookdown.org/yihui/rmarkdown/notebook.html#fig:notebook-external-code)).

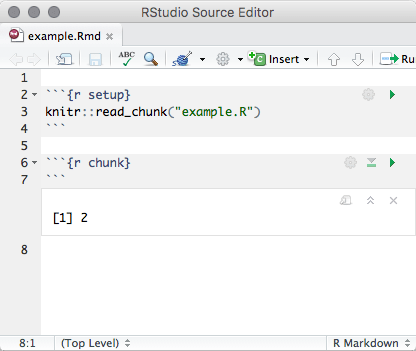


FIGURE 3.9: Execute a code chunk read from an external R script.

#### **3.2.1.4 Chunk output**

When code is executed in the notebook, its output appears beneath the code chunk that produced it. You can clear an individual chunk’s output by clicking the X button in the upper right corner of the output, or collapse it by clicking the chevron.

It is also possible to clear or collapse all of the output in the document at once using the Collapse All Output and Clear All Output menu items available on the gear menu in the editor toolbar (Figure [3.4](https://bookdown.org/yihui/rmarkdown/notebook.html#fig:notebook-console)).

If you want to fully reset the state of the notebook, the item Restart R and Clear Output on the Run menu on the editor toolbar will do the job.

Ordinary R Markdown documents are “knitted”, but notebooks are “previewed”. While the notebook preview looks similar to a rendered R Markdown document, the notebook preview does not execute any of your R code chunks. It simply shows you a rendered copy of the Markdown output of your document along with the most recent chunk output. This preview is generated automatically whenever you save the notebook (whether you are viewing it in RStudio or not); see the section beneath on the \*.nb.html file for details.

When html\_notebook is the topmost (default) format in your YAML metadata, you will see a Preview button in the editor toolbar. Clicking it will show you the notebook preview (Figure [3.10](https://bookdown.org/yihui/rmarkdown/notebook.html#fig:notebook-preview)).

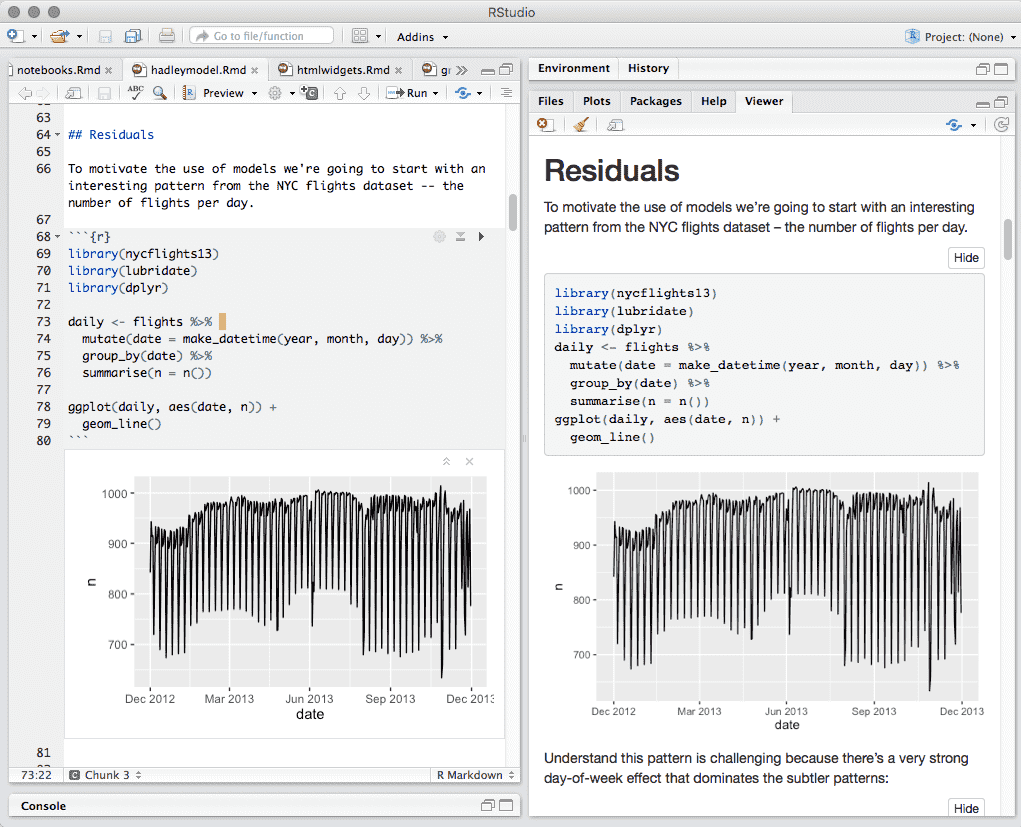


FIGURE 3.10: Preview a notebook.

If you have configured R Markdown previewing to use the Viewer pane (as illustrated in Figure [3.10](https://bookdown.org/yihui/rmarkdown/notebook.html#fig:notebook-preview)), the preview will be automatically updated whenever you save your notebook.

When an error occurs while a notebook chunk is executing (Figure [3.11](https://bookdown.org/yihui/rmarkdown/notebook.html#fig:notebook-error)):

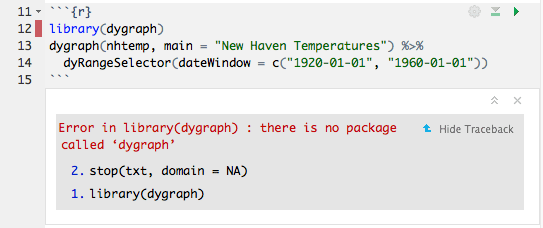


FIGURE 3.11: Errors in a notebook.

1. Execution will stop; the remaining lines of that chunk (and any chunks that have not yet been run) will not be executed.
2. The editor will scroll to the error.
3. The line of code that caused the error will have a red indicator in the editor’s gutter.

If you want your notebook to keep running after an error, you can suppress the first two behaviors by specifying error = TRUE in the chunk options.

In most cases, it should not be necessary to have the console open while using the notebook, as you can see all of the console output in the notebook itself. To preserve vertical space, the console will be automatically collapsed when you open a notebook or run a chunk in the notebook.

If you prefer not to have the console hidden when chunks are executed, uncheck the option from the menu Tools -> Global Options -> R Markdown -> Hide console automatically when executing notebook chunks.

### 3.2.2 Saving and sharing

#### **3.2.2.1 Notebook file**

When a notebook \*.Rmd file is saved, a \*.nb.html file is created alongside it. This file is a self-contained HTML file which contains both a rendered copy of the notebook with all current chunk outputs (suitable for display on a website) and a copy of the \*.Rmd file itself.

You can view the \*.nb.html file in any ordinary web browser. It can also be opened in RStudio; when you open there (e.g., using File -> Open File), RStudio will do the following:

1. Extract the bundled \*.Rmd file, and place it alongside the \*.nb.html file.
2. Open the \*.Rmd file in a new RStudio editor tab.
3. Extract the chunk outputs from the \*.nb.html file, and place them appropriately in the editor.

Note that the \*.nb.html file is only created for R Markdown documents that are notebooks (i.e., at least one of their output formats is html\_notebook). It is possible to have an R Markdown document that includes inline chunk output beneath code chunks, but does not produce an \*.nb.html file, when html\_notebook is not specified as an output format for the R Markdown document.

#### **3.2.2.2 Output storage**

The document’s chunk outputs are also stored in an internal RStudio folder beneath the project’s .Rproj.user folder. If you work with a notebook but do not have a project open, the outputs are stored in the RStudio state folder in your home directory (the location of this folder varies between the [desktop](https://support.rstudio.com/hc/en-us/articles/200534577) and the [server](https://support.rstudio.com/hc/en-us/articles/218730228)).

#### **3.2.2.3 Version control**

One of the major advantages of R Notebooks compared to other notebook systems is that they are plain-text files and therefore work well with version control. We recommend checking in both the \*.Rmd and \*.nb.html files into version control, so that both your source code and output are available to collaborators. However, you can choose to include only the \*.Rmd file (with a .gitignore that excludes \*.nb.html) if you want each collaborator to work with their own private copies of the output.

### 3.2.3 Notebook format

While RStudio provides a set of integrated tools for authoring R Notebooks, the notebook file format itself is decoupled from RStudio. The **rmarkdown** package provides several functions that can be used to read and write R Notebooks outside of RStudio.

In this section, we describe the internals of the notebook format. It is primarily intended for front-end applications using or embedding R, or other users who are interested in reading and writing documents using the R Notebook format. We recommend that beginners skip this section when reading this book or using notebooks for the first time.

R Notebooks are HTML documents with data written and encoded in such a way that:

1. The source Rmd document can be recovered, and
2. Chunk outputs can be recovered.

To generate an R Notebook, you can use rmarkdown::render() and specify the html\_notebook output format in your document’s YAML metadata. Documents rendered in this form will be generated with the .nb.html file extension, to indicate that they are HTML notebooks.

To ensure chunk outputs can be recovered, the elements of the R Markdown document are enclosed with HTML comments, providing more information on the output. For example, chunk output might be serialized in the form:

*<!-- rnb-chunk-begin -->*

*<!-- rnb-output-begin -->*

**<pre><code>**Hello, World!**</code></pre>**

*<!-- rnb-output-end -->*

*<!-- rnb-chunk-end -->*

Because R Notebooks are just HTML documents, they can be opened and viewed in any web browser; in addition, hosting environments can be configured to recover and open the source Rmd document, and also recover and display chunk outputs as appropriate.

#### **3.2.3.1 Generating R Notebooks with custom output**

It is possible to render an HTML notebook with custom chunk outputs inserted in lieu of the result that would be generated by evaluating the associated R code. This can be useful for front-end editors that show the output of chunk execution inline, or for conversion programs from other notebook formats where output is already available from the source format. To facilitate this, one can provide a custom “output source” to rmarkdown::render(). Let’s investigate with a simple example:

rmd\_stub = "examples/r-notebook-stub.Rmd"

**cat**(**readLines**(rmd\_stub), sep = "\n")

---

title: "R Notebook Stub"

output: html\_notebook

---

```{r chunk-one}

print("Hello, World!")

```

Let’s try to render this document with a custom output source, so that we can inject custom output for the single chunk within the document. The output source function will accept:

* code: The code within the current chunk.
* context: An environment containing active chunk options and other chunk information.
* ...: Optional arguments reserved for future expansion.

In particular, the context elements label and chunk.index can be used to help identify which chunk is currently being rendered.

output\_source = **function**(code, context, ...) {

logo = **file.path**(**R.home**("doc"), "html", "logo.jpg")

**if** (context$label == "chunk-one") **list**(

rmarkdown::**html\_notebook\_output\_code**("# R Code"),

**paste**("Custom output for chunk:", context$chunk.index),

rmarkdown::**html\_notebook\_output\_code**("# R Logo"),

rmarkdown::**html\_notebook\_output\_img**(logo)

)

}

We can pass our output\_source along as part of the output\_options list to rmarkdown::render().

output\_file = rmarkdown::**render**(

rmd\_stub,

output\_options = **list**(output\_source = output\_source),

quiet = TRUE

)

We have now generated an R Notebook. Open [this document](https://rmarkdown.rstudio.com/notebook/r-notebook-stub.nb.html) in a web browser, and it will show that the output\_source function has effectively side-stepped evaluation of code within that chunk, and instead returned the injected result.

#### **3.2.3.2 Implementing output sources**

In general, you can provide regular R output in your output source function, but **rmarkdown** also provides a number of endpoints for insertion of custom HTML content. These are documented within ?html\_notebook\_output.

Using these functions ensures that you produce an R Notebook that can be opened in R frontends (e.g., RStudio).

#### **3.2.3.3 Parsing R Notebooks**

The rmarkdown::parse\_html\_notebook() function provides an interface for recovering and parsing an HTML notebook.

parsed = rmarkdown::**parse\_html\_notebook**(output\_file)

**str**(parsed, width = 60, strict.width = "wrap")

List of 4

$ source : chr [1:1832] "<!DOCTYPE html>" "" "<html

xmlns=\"http://www.w3.org/1999/xhtml\">" "" ...

$ rmd : chr [1:8] "---" "title: \"R Notebook Stub\""

"output: html\_notebook" "---" ...

$ header : chr [1:1718] "<head>" "" "<meta

charset=\"utf-8\" />" "<meta http-equiv=\"Content-Type\"

content=\"text/html; charset=utf-8\" />" ...

$ annotations:List of 12

..$ :List of 4

.. ..$ row : int 1751

.. ..$ label: chr "text"

.. ..$ state: chr "begin"

.. ..$ meta : NULL

..$ :List of 4

.. ..$ row : int 1752

.. ..$ label: chr "text"

.. ..$ state: chr "end"

.. ..$ meta : NULL

..$ :List of 4

.. ..$ row : int 1753

.. ..$ label: chr "chunk"

.. ..$ state: chr "begin"

.. ..$ meta : NULL

..$ :List of 4

.. ..$ row : int 1754

.. ..$ label: chr "source"

.. ..$ state: chr "begin"

.. ..$ meta :List of 1

.. .. ..$ data: chr "```r\n# R Code\n```"

..$ :List of 4

.. ..$ row : int 1756

.. ..$ label: chr "source"

.. ..$ state: chr "end"

.. ..$ meta : NULL

..$ :List of 4

.. ..$ row : int 1757

.. ..$ label: chr "output"

.. ..$ state: chr "begin"

.. ..$ meta :List of 1

.. .. ..$ data: chr "Custom output for chunk: 1\n"

..$ :List of 4

.. ..$ row : int 1759

.. ..$ label: chr "output"

.. ..$ state: chr "end"

.. ..$ meta : NULL

..$ :List of 4

.. ..$ row : int 1760

.. ..$ label: chr "source"

.. ..$ state: chr "begin"

.. ..$ meta :List of 1

.. .. ..$ data: chr "```r\n# R Logo\n```"

..$ :List of 4

.. ..$ row : int 1762

.. ..$ label: chr "source"

.. ..$ state: chr "end"

.. ..$ meta : NULL

..$ :List of 4

.. ..$ row : int 1763

.. ..$ label: chr "plot"

.. ..$ state: chr "begin"

.. ..$ meta : NULL

..$ :List of 4

.. ..$ row : int 1765

.. ..$ label: chr "plot"

.. ..$ state: chr "end"

.. ..$ meta : NULL

..$ :List of 4

.. ..$ row : int 1766

.. ..$ label: chr "chunk"

.. ..$ state: chr "end"

.. ..$ meta : NULL

This interface can be used to recover the original Rmd source, and also (with some more effort from the front-end) the ability to recover chunk outputs from the document itself.

## **3.3 PDF document**

To create a PDF document from R Markdown, you specify the pdf\_document output format in the YAML metadata:

---

title: "Habits"

author: John Doe

date: March 22, 2005

output: pdf\_document

---

Within R Markdown documents that generate PDF output, you can use raw LaTeX, and even define LaTeX macros. See Pandoc’s documentation on the [raw\_tex extension](http://pandoc.org/MANUAL.html#extension-raw_tex) for details.

Note that PDF output (including Beamer slides) requires an installation of LaTeX (see Chapter [1](https://bookdown.org/yihui/rmarkdown/installation.html#installation)).

### 3.3.1 Table of contents

You can add a table of contents using the toc option and specify the depth of headers that it applies to using the toc\_depth option. For example:

---

title: "Habits"

output:

pdf\_document:

toc: true

toc\_depth: 2

---

If the TOC depth is not explicitly specified, it defaults to 3 (meaning that all level 1, 2, and 3 headers will be included in the TOC).

You can add section numbering to headers using the number\_sections option:

---

title: "Habits"

output:

pdf\_document:

toc: true

number\_sections: true

---

If you are familiar with LaTeX, number\_sections: true means \section{}, and number\_sections: false means \section\*{} for sections in LaTeX (it also applies to other levels of “sections” such as \chapter{}, and \subsection{}).

### 3.3.2 Figure options

There are a number of options that affect the output of figures within PDF documents:

* fig\_width and fig\_height can be used to control the default figure width and height (6x4.5 is used by default).
* fig\_crop controls whether the pdfcrop utility, if available in your system, is automatically applied to PDF figures (this is true by default).
  + If you are using [TinyTeX](https://yihui.name/tinytex/) as your LaTeX distribution, we recommend that you run tinytex::tlmgr\_install("pdfcrop") to install the LaTeX package pdfcrop. You also have to make sure the system package [ghostscript](https://www.ghostscript.com/) is available in your system for pdfcrop to work. For macOS users who have installed Homebrew, ghostscript can be installed via brew install ghostscript.
  + If your graphics device is postscript, we recommend that you disable this feature (see more info in the **knitr** issue [#1365](https://github.com/yihui/knitr/issues/1365)).
* fig\_caption controls whether figures are rendered with captions (this is false by default).
* dev controls the graphics device used to render figures (defaults to pdf).

For example:

---

title: "Habits"

output:

pdf\_document:

fig\_width: 7

fig\_height: 6

fig\_caption: true

---

### 3.3.3 Data frame printing

You can enhance the default display of data frames via the df\_print option. Valid values are presented in Table [3.3](https://bookdown.org/yihui/rmarkdown/pdf-document.html#tab:df-print-pdf).

| TABLE 3.3: The possible values of the df\_print option for the pdf\_document format. | |
| --- | --- |
| **Option** | **Description** |
| default | Call the print.data.frame generic method |
| kable | Use the knitr::kable() function |
| tibble | Use the tibble::print.tbl\_df() function |

For example:

---

title: "Habits"

output:

pdf\_document:

df\_print: kable

---

### 3.3.4 Syntax highlighting

The highlight option specifies the syntax highlighting style. Its usage in pdf\_document is the same as html\_document (Section [3.1.4](https://bookdown.org/yihui/rmarkdown/html-document.html#appearance-and-style)). For example:

---

title: "Habits"

output:

pdf\_document:

highlight: tango

---

### 3.3.5 LaTeX options

Many aspects of the LaTeX template used to create PDF documents can be customized using top-level YAML metadata (note that these options do not appear underneath the output section, but rather appear at the top level along with title, author, and so on). For example:

---

title: "Crop Analysis Q3 2013"

output: pdf\_document

fontsize: 11pt

geometry: margin=1in

---

A few available metadata variables are displayed in Table [3.4](https://bookdown.org/yihui/rmarkdown/pdf-document.html#tab:latex-vars) (consult the Pandoc manual for the full list):

| TABLE 3.4: Available top-level YAML metadata variables for LaTeX output. | |
| --- | --- |
| **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 |

### 3.3.6 LaTeX packages for citations

By default, citations are processed through pandoc-citeproc, which works for all output formats. For PDF output, sometimes it is better to use LaTeX packages to process citations, such as natbib or biblatex. To use one of these packages, just set the option citation\_package to be natbib or biblatex, e.g.

---

output:

pdf\_document:

citation\_package: natbib

---

### 3.3.7 Advanced customization

#### **3.3.7.1 LaTeX engine**

By default, PDF documents are rendered using pdflatex. You can specify an alternate engine using the latex\_engine option. Available engines are pdflatex, xelatex, and lualatex. For example:

---

title: "Habits"

output:

pdf\_document:

latex\_engine: xelatex

---

The main reasons you may want to use xelatex or lualatex are: (1) They support Unicode better; (2) It is easier to make use of system fonts. See some posts on Stack Overflow for more detailed explanations, e.g., <https://tex.stackexchange.com/q/3393/9128> and <https://tex.stackexchange.com/q/36/9128>.

#### **3.3.7.2 Keeping intermediate TeX**

R Markdown documents are converted to PDF by first converting to a TeX file and then calling the LaTeX engine to convert to PDF. By default, this TeX file is removed, however if you want to keep it (e.g., for an article submission), you can specify the keep\_tex option. For example:

---

title: "Habits"

output:

pdf\_document:

keep\_tex: true

---

#### **3.3.7.3 Includes**

You can do more advanced customization of PDF output by including additional LaTeX directives and/or content or by replacing the core Pandoc template entirely. To include content in the document header or before/after the document body, you use the includes option as follows:

---

title: "Habits"

output:

pdf\_document:

includes:

in\_header: preamble.tex

before\_body: doc-prefix.tex

after\_body: doc-suffix.tex

---

#### **3.3.7.4 Custom templates**

You can also replace the underlying Pandoc template using the template option:

---

title: "Habits"

output:

pdf\_document:

template: quarterly-report.tex

---

Consult the documentation on [Pandoc templates](https://pandoc.org/README.html#templates) for additional details on templates. You can also study the [default LaTeX template](https://github.com/jgm/pandoc-templates/blob/master/default.latex) as an example.

### 3.3.8 Other features

Similar to HTML documents, you can enable or disable certain Markdown extensions for generating PDF documents. See Section [3.1.10.4](https://bookdown.org/yihui/rmarkdown/html-document.html#markdown-extensions) for details. You can also pass more custom Pandoc arguments through the pandoc\_args option (Section [3.1.10.5](https://bookdown.org/yihui/rmarkdown/html-document.html#pandoc-arguments)), and define shared options in \_output.yml (Section [3.1.11](https://bookdown.org/yihui/rmarkdown/html-document.html#shared-options)).

## **3.4 Word document**

To create a Word document from R Markdown, you specify the word\_document output format in the YAML metadata of your document:

---

title: "Habits"

author: John Doe

date: March 22, 2005

output: word\_document

---

The most notable feature of Word documents is the Word template, which is also known as the “style reference document”. You can specify a document to be used as a style reference in producing a \*.docx file (a Word document). This will allow you to customize things such as margins and other formatting characteristics. For best results, the reference document should be a modified version of a .docx file produced using **rmarkdown** or Pandoc. The path of such a document can be passed to the reference\_docx argument of the word\_document format. Pass "default" to use the default styles. For example:

---

title: "Habits"

output:

word\_document:

reference\_docx: my-styles.docx

---

For more on how to create and use a reference document, you may watch this short video: <https://vimeo.com/110804387>, or read this detailed article: <https://rmarkdown.rstudio.com/articles_docx.html>.

### 3.4.1 Other features

Refer to Section [3.1](https://bookdown.org/yihui/rmarkdown/html-document.html#html-document) for the documentation of most features of Word documents, including figure options (Section [3.1.5](https://bookdown.org/yihui/rmarkdown/html-document.html#figure-options)), data frame printing (Section [3.1.6](https://bookdown.org/yihui/rmarkdown/html-document.html#data-frame-printing)), syntax highlighting (Section [3.1.4](https://bookdown.org/yihui/rmarkdown/html-document.html#appearance-and-style)), keeping Markdown (Section [3.1.10.1](https://bookdown.org/yihui/rmarkdown/html-document.html#keeping-markdown)), Markdown extensions (Section [3.1.10.4](https://bookdown.org/yihui/rmarkdown/html-document.html#markdown-extensions)), Pandoc arguments (Section [3.1.10.5](https://bookdown.org/yihui/rmarkdown/html-document.html#pandoc-arguments)), and shared options (Section [3.1.11](https://bookdown.org/yihui/rmarkdown/html-document.html#shared-options)).

## **3.5 OpenDocument Text document**

To create an OpenDocument Text (ODT) document from R Markdown, you specify the odt\_document output format in the YAML metadata of your document:

---

title: "Habits"

author: John Doe

date: March 22, 2005

output: odt\_document

---

Similar to word\_document, you can also provide a style reference document to odt\_document throught the reference\_odt option. For best results, the reference ODT document should be a modified version of an ODT file produced using **rmarkdown** or Pandoc. For example:

---

title: "Habits"

output:

odt\_document:

reference\_odt: my-styles.odt

---

### 3.5.1 Other features

Refer to Section [3.1](https://bookdown.org/yihui/rmarkdown/html-document.html#html-document) for the documentation of most features of ODT documents, including figure options (Section [3.1.5](https://bookdown.org/yihui/rmarkdown/html-document.html#figure-options)), keeping Markdown (Section [3.1.10.1](https://bookdown.org/yihui/rmarkdown/html-document.html#keeping-markdown)), header and before/after body inclusions (Section [3.1.10.2](https://bookdown.org/yihui/rmarkdown/html-document.html#includes)), custom templates (Section [3.1.10.3](https://bookdown.org/yihui/rmarkdown/html-document.html#custom-templates)), Markdown extensions (Section [3.1.10.4](https://bookdown.org/yihui/rmarkdown/html-document.html#markdown-extensions)), Pandoc arguments (Section [3.1.10.5](https://bookdown.org/yihui/rmarkdown/html-document.html#pandoc-arguments)), and shared options (Section [3.1.11](https://bookdown.org/yihui/rmarkdown/html-document.html#shared-options)).

## **3.6 Rich Text Format document**

To create a Rich Text Format (RTF) document from R Markdown, you specify the rtf\_document output format in the YAML metadata of your document:

---

title: "Habits"

author: John Doe

date: March 22, 2005

output: rtf\_document

---

If you know the RTF format really well, you can actually embed raw RTF content in R Markdown. For example, you may create a table in RTF using other software packages, and insert it to your final RTF output document. An RTF document is essentially a plain-text document, so you can read it into R using functions like readLines(). Now suppose you have an RTF table in the file table.rtf. To embed it in R Markdown, you read it and pass to knitr::raw\_output(), e.g.,

```{r, echo=FALSE}

knitr::raw\_output(readLines('table.rtf'))

```

### 3.6.1 Other features

Refer to Section [3.1](https://bookdown.org/yihui/rmarkdown/html-document.html#html-document) for the documentation of most features of RTF documents, including table of contents (Section [3.1.1](https://bookdown.org/yihui/rmarkdown/html-document.html#table-of-contents)), figure options (Section [3.1.5](https://bookdown.org/yihui/rmarkdown/html-document.html#figure-options)), keeping Markdown (Section [3.1.10.1](https://bookdown.org/yihui/rmarkdown/html-document.html#keeping-markdown)), Markdown extensions (Section [3.1.10.4](https://bookdown.org/yihui/rmarkdown/html-document.html#markdown-extensions)), Pandoc arguments (Section [3.1.10.5](https://bookdown.org/yihui/rmarkdown/html-document.html#pandoc-arguments)), and shared options (Section [3.1.11](https://bookdown.org/yihui/rmarkdown/html-document.html#shared-options)).

## **3.7 Markdown document**

In some cases, you might want to produce plain Markdown output from R Markdown (e.g., to create a document for a system that accepts Markdown input like [Stack Overflow](https://stackoverflow.com/editing-help)).

To create a Markdown document from R Markdown, you specify the md\_document output format in the front-matter of your document:

---

title: "Habits"

author: John Doe

date: March 22, 2005

output: md\_document

---

### 3.7.1 Markdown variants

By default, the md\_document format produces “strict” Markdown (i.e., conforming to the original Markdown specification with no extensions). You can generate a different flavor of Markdown using the variant option. For example:

---

output:

md\_document:

variant: markdown\_github

---

Valid values are:

* markdown (Full Pandoc Markdown)
* markdown\_strict (Original Markdown specification; the default)
* markdown\_github (GitHub Flavored Markdown)
* markdown\_mmd (MultiMarkdown)
* markdown\_phpextra (PHP Markdown extra)

You can also compose custom Markdown variants. For example:

---

output:

md\_document:

variant: markdown\_strict+backtick\_code\_blocks+autolink\_bare\_uris

---

See Pandoc’s Manual for all of the Markdown extensions and their names to be used in composing custom variants.

#### **3.7.1.1 Publishing formats**

Many popular publishing systems now accept Markdown as input. Table [3.5](https://bookdown.org/yihui/rmarkdown/markdown-document.html#tab:markdown-variant) shows the correct Markdown variants to use for some popular systems.

| TABLE 3.5: Markdown variants for some popular publishing systems. | |
| --- | --- |
| **System** | **Markdown Variant** |
| GitHub Wikis | markdown\_github |
| Drupal | markdown\_phpextra |
| WordPress.com | markdown\_phpextra+backtick\_code\_blocks |
| StackOverflow | markdown\_strict+autolink\_bare\_uris |

In many cases, you can simply copy and paste the Markdown generated by rmarkdown::render() into the editing interface of the system you are targeting. Note, however, that if you have embedded plots or other images, you will need to upload them separately and fix up their URLs to point to the uploaded location. If you intend to build websites based on R Markdown, we recommend that you use the more straightforward solutions such as **blogdown** (Xie, Hill, and Thomas [2017](https://bookdown.org/yihui/rmarkdown/markdown-document.html#ref-xie2017); Xie [2019a](https://bookdown.org/yihui/rmarkdown/markdown-document.html#ref-R-blogdown)) as introduced in Section [10](https://bookdown.org/yihui/rmarkdown/websites.html#websites) instead of manually copying the Markdown content.

### 3.7.2 Other features

Refer to Section [3.1](https://bookdown.org/yihui/rmarkdown/html-document.html#html-document) for the documentation of other features of Markdown documents, including table of contents (Section [3.1.1](https://bookdown.org/yihui/rmarkdown/html-document.html#table-of-contents)), figure options (Section [3.1.5](https://bookdown.org/yihui/rmarkdown/html-document.html#figure-options)), header and before/after body inclusions (Section [3.1.10.2](https://bookdown.org/yihui/rmarkdown/html-document.html#includes)), Pandoc arguments (Section [3.1.10.5](https://bookdown.org/yihui/rmarkdown/html-document.html#pandoc-arguments)), and shared options (Section [3.1.11](https://bookdown.org/yihui/rmarkdown/html-document.html#shared-options)).

### References

Xie, Yihui. 2019a. Blogdown: Create Blogs and Websites with R Markdown. [https://CRAN.R-project.org/package=blogdown](https://cran.r-project.org/package=blogdown).

Xie, Yihui, Alison Presmanes Hill, and Amber Thomas. 2017. Blogdown: Creating Websites with R Markdown. Boca Raton, Florida: Chapman; Hall/CRC. <https://github.com/rstudio/blogdown>.

## **3.8 R package vignette**

The html\_vignette format provides a lightweight alternative to html\_document suitable for inclusion in packages to be released to CRAN. It reduces the size of a basic vignette from 600Kb to around 10Kb. The format differs from a conventional HTML document as follows:

* Never uses retina figures
* Has a smaller default figure size
* Uses a custom lightweight CSS stylesheet

To use html\_vignette, you specify it as the output format and add some additional vignette related settings via the \Vignette\*{} macros:

---

title: "Your Vignette Title"

output: rmarkdown::html\_vignette

vignette: >

%\VignetteEngine{knitr::rmarkdown}

%\VignetteIndexEntry**{**Your Vignette Title**}**

%\VignetteEncoding**{**UTF-8**}**

---

Note that you should change the title field and the \VignetteIndexEntry{} to match the title of your vignette.

Most options for html\_document (Section [3.1](https://bookdown.org/yihui/rmarkdown/html-document.html#html-document)) also work for html\_vignette, except fig\_retina and theme, which have been set to null internally in this format.

The html\_vignette template includes a basic CSS theme. To override this theme, you can specify your own CSS in the document metadata as follows:

output:

rmarkdown::html\_vignette:

css: mystyles.css

The default figure size is 3x3. Because the figure width is small, usually you will be able to put two images side-by-side if you set the chunk option fig.show='hold', e.g.,

```{r, fig.show='hold'}

plot(1:10)

plot(10:1)

```

If you want larger figure sizes you can change the fig\_width and fig\_height in the document output options or alternatively override the default options on a per-chunk basis.

# Chapter 4 Presentations

For documents, the basic units are often sections. For presentations, the basic units are slides. A section in the Markdown source document often indicates a new slide in the presentation formats. In this chapter, we introduce the built-in presentation formats in the **rmarkdown** package.

## **4.1 ioslides presentation**

To create an ioslides presentation from R Markdown, you specify the ioslides\_presentation output format in the YAML metadata of your document. You can create a slide show broken up into sections by using the # and ## heading tags (you can also create a new slide without a header using a horizontal rule (---). For example here is a simple slide show (see Figure [4.1](https://bookdown.org/yihui/rmarkdown/ioslides-presentation.html#fig:ioslides) for two sample slides):

---

title: "Habits"

author: John Doe

date: March 22, 2005

output: ioslides\_presentation

---

# In the morning

## Getting up

- Turn off alarm

- Get out of bed

## Breakfast

- Eat eggs

- Drink coffee

# In the evening

## Dinner

- Eat spaghetti

- Drink wine

---

```{r, cars, fig.cap="A scatterplot.", echo=FALSE}

plot(cars)

```

## Going to sleep

- Get in bed

- Count sheep

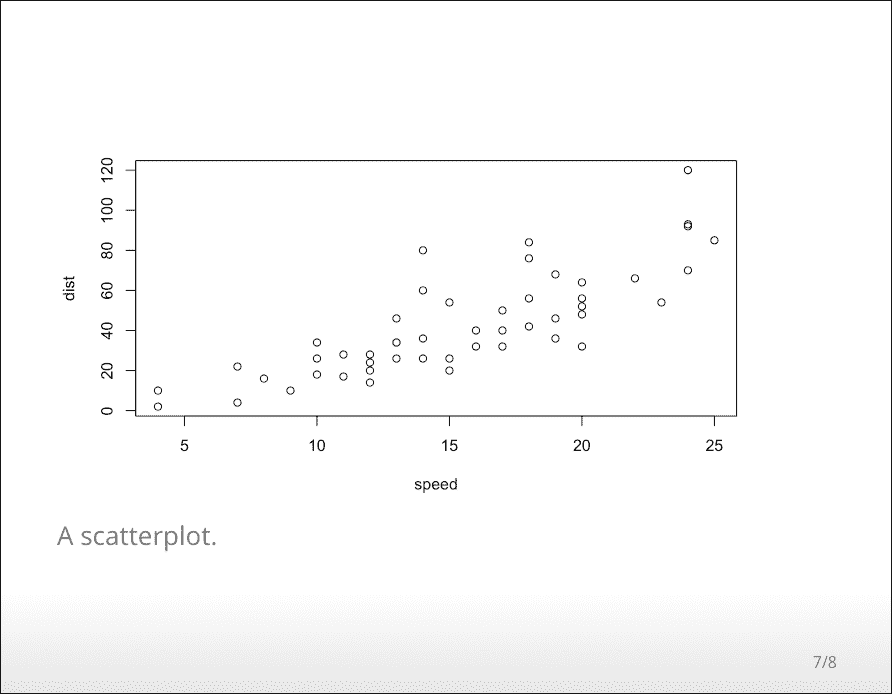
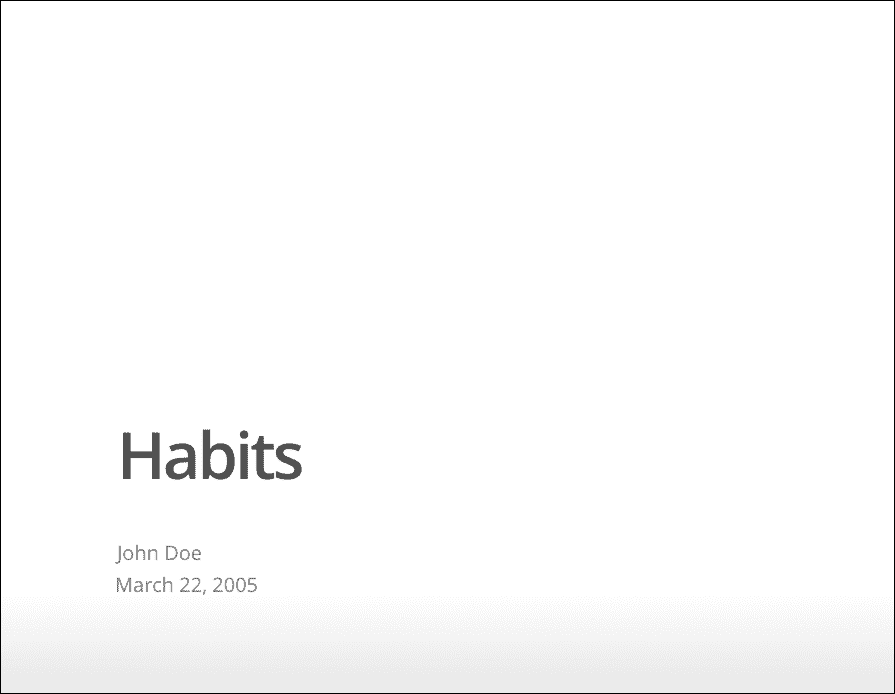


FIGURE 4.1: Two sample slides in an ioslides presentation.

You can add a subtitle to a slide or section by including text after the pipe (|) character. For example:

## Getting up | What I like to do first thing

### 4.1.1 Display modes

The following single character keyboard shortcuts enable alternate display modes:

* 'f': enable fullscreen mode
* 'w': toggle widescreen mode
* 'o': enable overview mode
* 'h': enable code highlight mode
* 'p': show presenter notes

Pressing Esc exits all of these modes. See the sections below on Code Highlighting and Presenter Mode for additional detail on those modes.

### 4.1.2 Incremental bullets

You can render bullets incrementally by adding the incremental option:

---

output:

ioslides\_presentation:

incremental: true

---

If you want to render bullets incrementally for some slides but not others you can (ab)use this syntax for blockquotes:

> - Eat eggs

> - Drink coffee

### 4.1.3 Visual appearance

#### **4.1.3.1 Presentation size**

You can display the presentation using a wider form factor using the widescreen option. You can specify that smaller text be used with the smaller option. For example:

---

output:

ioslides\_presentation:

widescreen: true

smaller: true

---

You can also enable the smaller option on a slide-by-slide basis by adding the .smaller attribute to the slide header:

## Getting up {.smaller}

#### **4.1.3.2 Transition speed**

You can customize the speed of slide transitions using transition option. This can be "default", "slower", "faster", or a numeric value with a number of seconds (e.g., 0.5) . For example:

---

output:

ioslides\_presentation:

transition: slower

---

#### **4.1.3.3 Build slides**

Slides can also have a .build attribute that indicate that their content should be displayed incrementally. For example:

## Getting up {.build}

Slide attributes can be combined if you need to specify more than one. For example:

## Getting up {.smaller .build}

#### **4.1.3.4 Background images**

You can specify a background image for a slide using the attribute data-background, and use other attributes including data-background-size, data-background-position, and data-background-repeat to tweak the style of the image. You need to be familiar with CSS to fully understand these four attributes, and they correspond to [the CSS properties](https://www.w3schools.com/cssref/css3_pr_background.asp) background, background-size, background-position, and background-repeat, respectively. For example:

## Getting up {data-background=foo.png data-background-size=cover}

#### **4.1.3.5 Custom CSS**

You can add your own CSS to an ioslides presentation using the css option:

---

output:

ioslides\_presentation:

css: styles.css

---

You can also target specific slides or classes of slides with custom CSS by adding IDs or classes to the slides headers within your document. For example the following slide header:

## Future Steps {#future-steps .emphasized}

Would enable you to apply CSS to all of its content using either of the following CSS selectors:

#future-steps {

**color**: blue;

}

.emphasized {

**font-size**: 1.2em;

}

### 4.1.4 Code highlighting

It is possible to select subsets of code for additional emphasis by adding a special “highlight” comment around the code. For example:

*### <b>*

x <- 10

y <- x \* 2

*### </b>*

The highlighted region will be displayed with a bold font. When you want to help the audience focus exclusively on the highlighted region press the h key and the rest of the code will fade away.

### 4.1.5 Adding a logo

You can add a logo to the presentation using the logo option (by default, the logo will be displayed in a 85x85 pixel square). For example:

---

output:

ioslides\_presentation:

logo: logo.png

---

The logo graphic will be rescaled to 85x85 (if necessary) and added to the title slide. A smaller icon version of the logo will be included in the bottom-left footer of each slide.

The logo on the title page and the rectangular element containing it can be customised with CSS. For example:

.gdbar img {

**width**: 300px !important;

**height**: 150px !important;

**margin**: 8px 8px;

}

.gdbar {

**width**: 400px !important;

**height**: 170px !important;

}

These selectors are to be placed in the CSS text file.

Similarly, the logo in the footer of each slide can be resized to any desired size. For example:

slides > slide***:not***(.nobackground)***:before*** {

**width**: 150px;

**height**: 75px;

**background-size**: 150px 75px;

}

This will make the footer logo 150 by 75 pixels in size.

### 4.1.6 Tables

The ioslides template has an attractive default style for tables so you should not hesitate to add tables for presenting more complex sets of information. Pandoc Markdown supports several syntaxes for defining tables, which are described in the Pandoc Manual.

### 4.1.7 Advanced layout

You can center content on a slide by adding the .flexbox and .vcenter attributes to the slide title. For example:

## Dinner {.flexbox .vcenter}

You can horizontally center content by enclosing it in a div tag with class centered. For example:

**<div** class="centered"**>**

This text is centered.

**</div>**

You can do a two-column layout using the columns-2 class. For example:

**<div** class="columns-2"**>**

![](image.png)

- Bullet 1

- Bullet 2

- Bullet 3

**</div>**

Note that content will flow across the columns so if you want to have an image on one side and text on the other you should make sure that the image has sufficient height to force the text to the other side of the slide.

### 4.1.8 Text color

You can color content using base color classes red, blue, green, yellow, and gray (or variations of them, e.g., red2, red3, blue2, blue3, etc.). For example:

**<div** class="red2"**>**

This text is red

**</div>**

### 4.1.9 Presenter mode

A separate presenter window can also be opened (ideal for when you are presenting on one screen but have another screen that is private to you). The window stays in sync with the main presentation window and also shows presenter notes and a thumbnail of the next slide. To enable presenter mode add ?presentme=true to the URL of the presentation. For example:

my-presentation.html?presentme=true

The presenter mode window will open and will always re-open with the presentation until it is disabled with:

my-presentation.html?presentme=false

To add presenter notes to a slide, you include it within a “notes” div. For example:

**<div** class="notes"**>**

This is my \*note\*.

- It can contain markdown

- like this list

**</div>**

### 4.1.10 Printing and PDF output

You can print an ioslides presentation from within browsers that have good support for print CSS (as of this writing, Google Chrome has the best support). Printing maintains most of the visual styles of the HTML version of the presentation.

To create a PDF version of a presentation, you can use the menu Print to PDF from Google Chrome.

### 4.1.11 Custom templates

You can replace the underlying Pandoc template using the template option:

---

title: "Habits"

output:

ioslides\_presentation:

template: quarterly-report.html

---

However, please note that the level of customization that can be achieved is limited compared to the templates of other output formats, because the slides are generated by custom formatting written in Lua, and as such the template used must include the string RENDERED\_SLIDES as can be found in the default template file with the path [rmarkdown:::rmarkdown\_system\_file("rmd/ioslides/default.html")](https://github.com/rstudio/rmarkdown/blob/master/inst/rmd/ioslides/default.html).

### 4.1.12 Other features

Refer to Section [3.1](https://bookdown.org/yihui/rmarkdown/html-document.html#html-document) for the documentation of other features of ioslides presentations, including figure options (Section [3.1.5](https://bookdown.org/yihui/rmarkdown/html-document.html#figure-options)), MathJax equations (Section [3.1.8](https://bookdown.org/yihui/rmarkdown/html-document.html#mathjax-equations)), data frame printing (Section [3.1.6](https://bookdown.org/yihui/rmarkdown/html-document.html#data-frame-printing)), Markdown extensions (Section [3.1.10.4](https://bookdown.org/yihui/rmarkdown/html-document.html#markdown-extensions)), keeping Markdown (Section [3.1.10.1](https://bookdown.org/yihui/rmarkdown/html-document.html#keeping-markdown)), document dependencies (Section [3.1.9](https://bookdown.org/yihui/rmarkdown/html-document.html#document-dependencies)), header and before/after body inclusions (Section [3.1.10.2](https://bookdown.org/yihui/rmarkdown/html-document.html#includes)), Pandoc arguments (Section [3.1.10.5](https://bookdown.org/yihui/rmarkdown/html-document.html#pandoc-arguments)), and shared options (Section [3.1.11](https://bookdown.org/yihui/rmarkdown/html-document.html#shared-options)).

## **4.2 Slidy presentation**

To create a [Slidy](https://www.w3.org/Talks/Tools/Slidy2/) presentation from R Markdown, you specify the slidy\_presentation output format in the YAML metadata of your document. You can create a slide show broken up into sections by using the ## heading tag (you can also create a new slide without a header using a horizontal rule (---). For example, here is a simple slide show (see Figure [4.2](https://bookdown.org/yihui/rmarkdown/slidy-presentation.html#fig:slidy) for two sample slides):

---

title: "Habits"

author: John Doe

date: March 22, 2005

output: slidy\_presentation

---

# In the morning

## Getting up

- Turn off alarm

- Get out of bed

## Breakfast

- Eat eggs

- Drink coffee

# In the evening

## Dinner

- Eat spaghetti

- Drink wine

---

```{r, cars, fig.cap="A scatterplot.", echo=FALSE}

plot(cars)

```

## Going to sleep

- Get in bed

- Count sheep

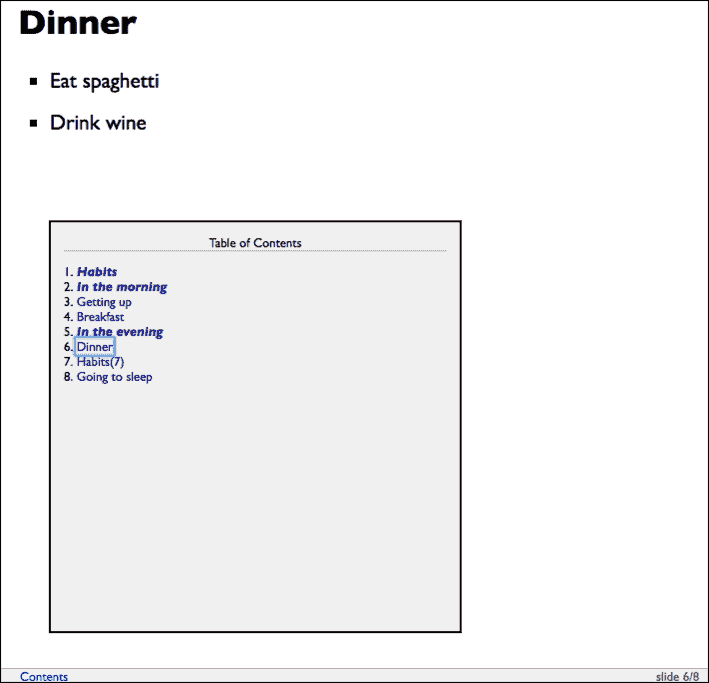
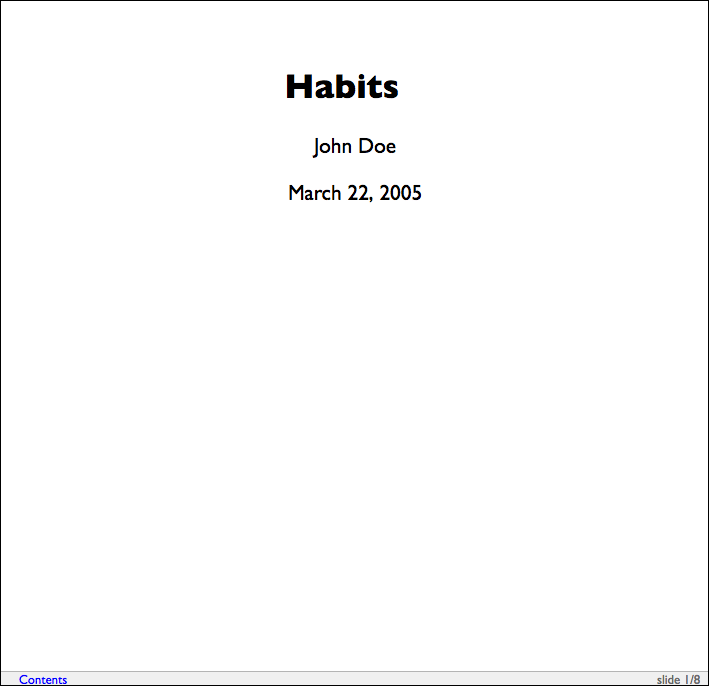


FIGURE 4.2: Two sample slides in a Slidy presentation.

### 4.2.1 Display modes

The following single character keyboard shortcuts enable alternate display modes:

* 'C': Show table of contents (the right sub-figure in Figure [4.2](https://bookdown.org/yihui/rmarkdown/slidy-presentation.html#fig:slidy) has shown the table of contents).
* 'F': Toggles the display of the footer.
* 'A': Toggles display of current vs all slides (useful for printing handouts).
* 'S': Make fonts smaller.
* 'B': Make fonts larger .

### 4.2.2 Text size

You can use the font\_adjustment option to increase or decrease the default font size (e.g., -1 or +1) for the entire presentation. For example:

---

output:

slidy\_presentation:

font\_adjustment: -1

---

If you want to decrease the text size on an individual slide you can use the .smaller slide attribute. For example:

## Getting up {.smaller}

If you want to increase the text size on an individual slide you can use the .bigger slide attribute. For example:

## Getting up {.bigger}

You can also manually adjust the font size during the presentation using the 'S' (smaller) and 'B' (bigger) keys.

### 4.2.3 Footer elements

You can add a countdown timer to the footer of your slides using the duration option (duration is specified in minutes). For example:

---

output:

slidy\_presentation:

duration: 45

---

You can also add custom footer text (e.g., organization name and/or copyright) using the footer option. For example:

---

output:

slidy\_presentation:

footer: "Copyright (c) 2014, RStudio"

---

### 4.2.4 Other features

Refer to Section [3.1](https://bookdown.org/yihui/rmarkdown/html-document.html#html-document) for the documentation of other features of Slidy presentations, including figure options (Section [3.1.5](https://bookdown.org/yihui/rmarkdown/html-document.html#figure-options)), appearance and style (Section [3.1.4](https://bookdown.org/yihui/rmarkdown/html-document.html#appearance-and-style)), MathJax equations (Section [3.1.8](https://bookdown.org/yihui/rmarkdown/html-document.html#mathjax-equations)), data frame printing (Section [3.1.6](https://bookdown.org/yihui/rmarkdown/html-document.html#data-frame-printing)), Markdown extensions (Section [3.1.10.4](https://bookdown.org/yihui/rmarkdown/html-document.html#markdown-extensions)), keeping Markdown (Section [3.1.10.1](https://bookdown.org/yihui/rmarkdown/html-document.html#keeping-markdown)), document dependencies (Section [3.1.9](https://bookdown.org/yihui/rmarkdown/html-document.html#document-dependencies)), header and before/after body inclusions (Section [3.1.10.2](https://bookdown.org/yihui/rmarkdown/html-document.html#includes)), custom templates (Section [3.1.10.3](https://bookdown.org/yihui/rmarkdown/html-document.html#custom-templates)), Pandoc arguments (Section [3.1.10.5](https://bookdown.org/yihui/rmarkdown/html-document.html#pandoc-arguments)), and shared options (Section [3.1.11](https://bookdown.org/yihui/rmarkdown/html-document.html#shared-options)).

Slidy presentations have several features in common with ioslides presentations in Section [4.1](https://bookdown.org/yihui/rmarkdown/ioslides-presentation.html#ioslides-presentation). For incremental bullets, see Section [4.1.2](https://bookdown.org/yihui/rmarkdown/ioslides-presentation.html#incremental-bullets). For custom CSS, see Section [4.1.3.5](https://bookdown.org/yihui/rmarkdown/ioslides-presentation.html#custom-css-ioslides). For printing Slidy slides to PDF, see Section [4.1.10](https://bookdown.org/yihui/rmarkdown/ioslides-presentation.html#printing-and-pdf-output).

## **4.3 Beamer presentation**

To create a Beamer presentation from R Markdown, you specify the beamer\_presentation output format in the YAML metadata of your document. You can create a slide show broken up into sections by using the # and ## heading tags (you can also create a new slide without a header using a horizontal rule (---). For example, here is a simple slide show (see Figure [4.3](https://bookdown.org/yihui/rmarkdown/beamer-presentation.html#fig:beamer) for two sample slides):

---

title: "Habits"

author: John Doe

date: March 22, 2005

output: beamer\_presentation

---

# In the morning

## Getting up

- Turn off alarm

- Get out of bed

## Breakfast

- Eat eggs

- Drink coffee

# In the evening

## Dinner

- Eat spaghetti

- Drink wine

---

```{r, cars, fig.cap="A scatterplot.", echo=FALSE}

plot(cars)

```

## Going to sleep

- Get in bed

- Count sheep

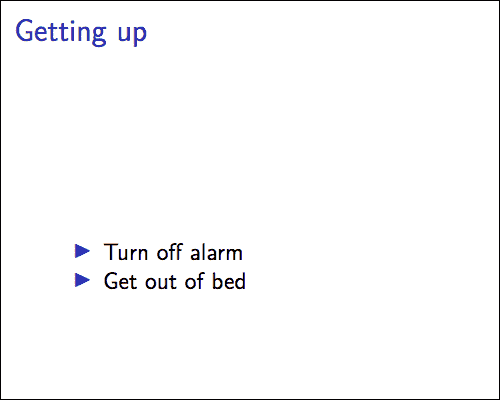
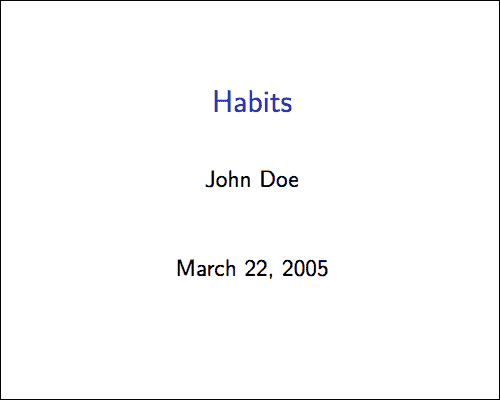


FIGURE 4.3: Two sample slides in a Beamer presentation.

Within R Markdown documents that generate PDF output, you can use raw LaTeX and even define LaTeX macros. See Pandoc’s manual for details.

### 4.3.1 Themes

You can specify Beamer themes using the theme, colortheme, and fonttheme options. For example:

---

output:

beamer\_presentation:

theme: "AnnArbor"

colortheme: "dolphin"

fonttheme: "structurebold"

---

Figure [4.4](https://bookdown.org/yihui/rmarkdown/beamer-presentation.html#fig:beamer-theme) shows two sample slides of the AnnArbor theme in the above example. You can find a list of possible themes and color themes at <https://hartwork.org/beamer-theme-matrix/>.

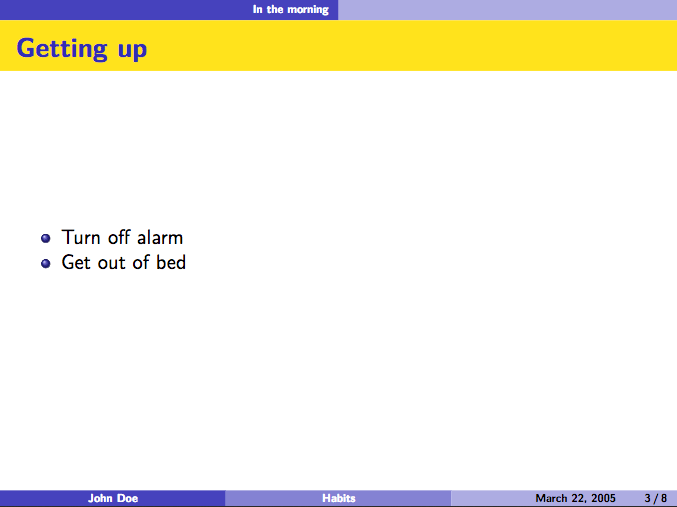
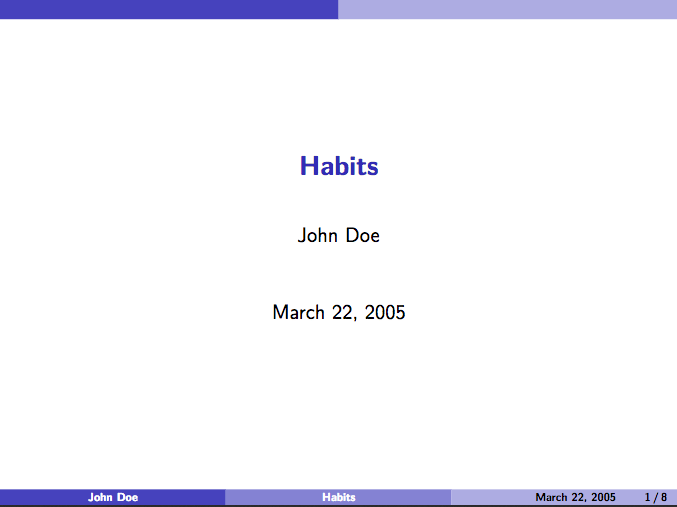


FIGURE 4.4: Two sample slides with the AnnArbor theme in Beamer.

### 4.3.2 Slide level

The slide\_level option defines the heading level that defines individual slides. By default, this is the highest header level in the hierarchy that is followed immediately by content, and not another header, somewhere in the document. This default can be overridden by specifying an explicit slide\_level:

---

output:

beamer\_presentation:

slide\_level: 2

---

### 4.3.3 Other features

Refer to Section [3.1](https://bookdown.org/yihui/rmarkdown/html-document.html#html-document) for the documentation of other features of Beamer presentations, including table of contents (Section [3.1.1](https://bookdown.org/yihui/rmarkdown/html-document.html#table-of-contents)), figure options (Section [3.1.5](https://bookdown.org/yihui/rmarkdown/html-document.html#figure-options)), appearance and style (Section [3.1.4](https://bookdown.org/yihui/rmarkdown/html-document.html#appearance-and-style)), data frame printing (Section [3.1.6](https://bookdown.org/yihui/rmarkdown/html-document.html#data-frame-printing)), Markdown extensions (Section [3.1.10.4](https://bookdown.org/yihui/rmarkdown/html-document.html#markdown-extensions)), header and before/after body inclusions (Section [3.1.10.2](https://bookdown.org/yihui/rmarkdown/html-document.html#includes)), custom templates (Section [3.1.10.3](https://bookdown.org/yihui/rmarkdown/html-document.html#custom-templates)), Pandoc arguments (Section [3.1.10.5](https://bookdown.org/yihui/rmarkdown/html-document.html#pandoc-arguments)), and shared options (Section [3.1.11](https://bookdown.org/yihui/rmarkdown/html-document.html#shared-options)).

Beamer presentations have a few features in common with ioslides presentations in Section [4.1](https://bookdown.org/yihui/rmarkdown/ioslides-presentation.html#ioslides-presentation) and PDF documents in Section [3.3](https://bookdown.org/yihui/rmarkdown/pdf-document.html#pdf-document). For incremental bullets, see Section [4.1.2](https://bookdown.org/yihui/rmarkdown/ioslides-presentation.html#incremental-bullets). For how to keep the intermediate LaTeX output file, see Section [3.3.7.2](https://bookdown.org/yihui/rmarkdown/pdf-document.html#keeping-intermediate-tex).

## **4.4 PowerPoint presentation**

To create a PowerPoint presentation from R Markdown, you specify the powerpoint\_presentation output format in the YAML metadata of your document. Please note that this output format is only available in **rmarkdown** >= v1.9, and requires at least Pandoc v2.0.5. You can check the versions of your **rmarkdown** package and Pandoc with packageVersion('rmarkdown') and rmarkdown::pandoc\_version() in R, respectively. The RStudio version 1.1.x ships Pandoc 1.19.2.1, which is not sufficient to generate PowerPoint presentations. You need to either install Pandoc 2.x by yourself if you use RStudio 1.1.x, or install [a preview version of RStudio](https://www.rstudio.com/products/rstudio/download/preview/) (>= 1.2.633), which has bundled Pandoc 2.x.

Below is a quick example (see Figure [4.5](https://bookdown.org/yihui/rmarkdown/powerpoint-presentation.html#fig:powerpoint) for a sample slide):

---

title: "Habits"

author: John Doe

date: March 22, 2005

output: powerpoint\_presentation

---

# In the morning

## Getting up

- Turn off alarm

- Get out of bed

## Breakfast

- Eat eggs

- Drink coffee

# In the evening

## Dinner

- Eat spaghetti

- Drink wine

---

```{r, cars, fig.cap="A scatterplot.", echo=FALSE}

plot(cars)

```

## Going to sleep

- Get in bed

- Count sheep

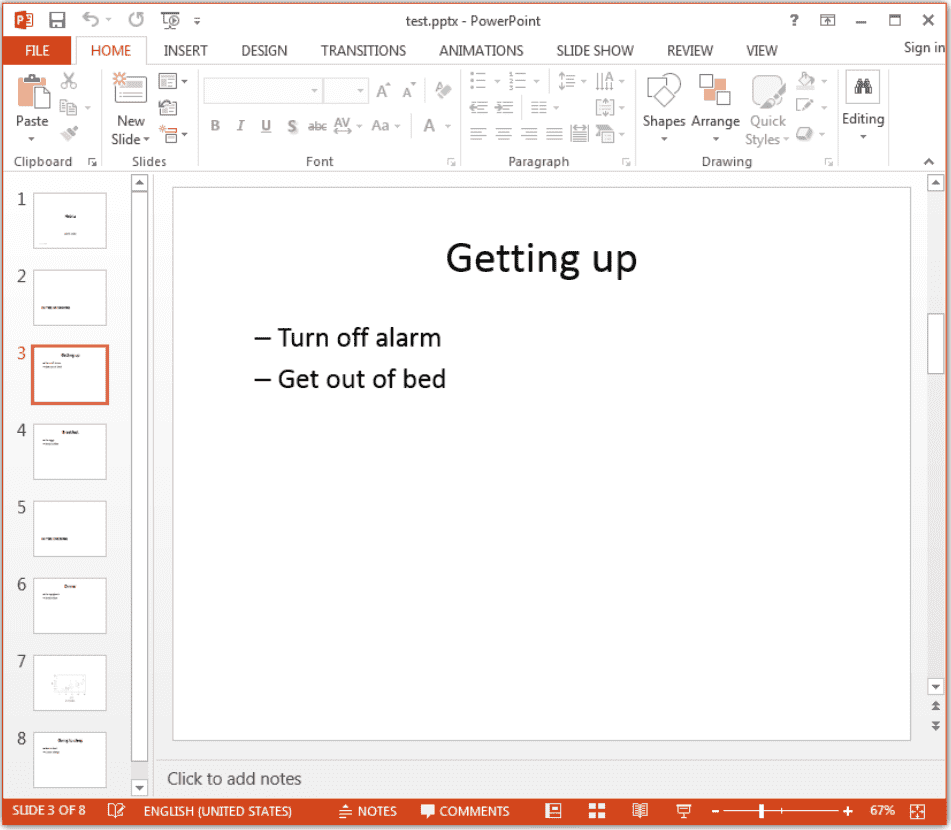


FIGURE 4.5: A sample slide in a PowerPoint presentation.

The default slide level (i.e., the heading level that defines individual slides) is determined in the same way as in Beamer slides (Section [4.3.2](https://bookdown.org/yihui/rmarkdown/beamer-presentation.html#slide-level)), and you can specify an explicit level via the slide\_level option under powerpoint\_presentation. You can also start a new slide without a header using a horizontal rule ---.

You can generate most elements supported by Pandoc’s Markdown (Section [2.5](https://bookdown.org/yihui/rmarkdown/markdown-syntax.html#markdown-syntax)) in PowerPoint output, such as bold/italic text, footnotes, bullets, LaTeX math expressions, images, and tables, etc.

Please note that images and tables will always be placed on new slides. The only elements that can coexist with an image or table on a slide are the slide header and image/table caption. When you have a text paragraph and an image on the same slide, the image will be moved to a new slide automatically. Images will be scaled automatically to fit the slide, and if the automatic size does not work well, you may manually control the image sizes: for static images included via the Markdown syntax ![](), you may use the width and/or height attributes in a pair of curly braces after the image, e.g., ![caption](foo.png){width=40%}; for images generated dynamically from R code chunks, you can use the chunk options fig.width and fig.height to control the sizes.

Please read the section “Producing slide shows with Pandoc” in Pandoc’s manual for more information on slide shows, such as the multi-column layout:

:::::: {.columns}

::: {.column width="40%"}

Content of the left column.

:::

::: {.column width="60%"}

Content of the right column.

:::

::::::

### 4.4.1 Custom templates

Like Word documents (Section [3.4](https://bookdown.org/yihui/rmarkdown/word-document.html#word-document)), you can customize the appearance of PowerPoint presentations by passing a custom reference document via the reference\_doc option, e.g.,

---

title: "Habits"

output:

powerpoint\_presentation:

reference\_doc: my-styles.pptx

---

Note that the reference\_doc option requires a version of **rmarkdown** higher than 1.9:

**if** (**packageVersion**("rmarkdown") <= "1.9") {

**install.packages**("rmarkdown") *# update rmarkdown from CRAN*

}

Basically any template included in a recent version of Microsoft PowerPoint should work. You can create a new \*.pptx file from the PowerPoint menu File -> New with your desired template, save the new file, and use it as the reference document (template) through the reference\_doc option. Pandoc will read the styles in the template and apply them to the PowerPoint presentation to be created from R Markdown.

### 4.4.2 Other features

Refer to Section [3.1](https://bookdown.org/yihui/rmarkdown/html-document.html#html-document) for the documentation of other features of PowerPoint presentations, including table of contents (Section [3.1.1](https://bookdown.org/yihui/rmarkdown/html-document.html#table-of-contents)), figure options (Section [3.1.5](https://bookdown.org/yihui/rmarkdown/html-document.html#figure-options)), data frame printing (Section [3.1.6](https://bookdown.org/yihui/rmarkdown/html-document.html#data-frame-printing)), keeping Markdown (Section [3.1.10.1](https://bookdown.org/yihui/rmarkdown/html-document.html#keeping-markdown)), Markdown extensions (Section [3.1.10.4](https://bookdown.org/yihui/rmarkdown/html-document.html#markdown-extensions)), Pandoc arguments (Section [3.1.10.5](https://bookdown.org/yihui/rmarkdown/html-document.html#pandoc-arguments)), and shared options (Section [3.1.11](https://bookdown.org/yihui/rmarkdown/html-document.html#shared-options)). As of Pandoc 2.2.1, incremental slides in PowerPoint are not supported yet.

Chapter 5 Dashboards

R Markdown is customizable and extensible. In Chapters [3](https://bookdown.org/yihui/rmarkdown/documents.html#documents) and [4](https://bookdown.org/yihui/rmarkdown/presentations.html#presentations), we have introduced basic document and presentation formats in the **rmarkdown** package, and explained how to customize them. From this chapter on, we will show several more existing extension packages that bring different styles, layouts, and applications to the R Markdown ecosystem. In this chapter, we introduce dashboards based on the **flexdashboard** package (Iannone, Allaire, and Borges [2018](https://bookdown.org/yihui/rmarkdown/dashboards.html#ref-R-flexdashboard)).

Dashboards are particularly common in business-style reports. They can be used to highlight brief and key summaries of a report. The layout of a dashboard is often grid-based, with components arranged in boxes of various sizes.

With the **flexdashboard** package, you can

* Use R Markdown to publish a group of related data visualizations as a dashboard.
* Embed a wide variety of components including HTML widgets, R graphics, tabular data, gauges, value boxes, and text annotations.
* Specify row or column-based layouts (components are intelligently re-sized to fill the browser and adapted for display on mobile devices).
* Create story boards for presenting sequences of visualizations and related commentary.
* Optionally use Shiny to drive visualizations dynamically.

To author a dashboard, you create an R Markdown document with the flexdashboard::flex\_dashboard output format. You can also create a document from within RStudio using the File -> New File -> R Markdown dialog, and choosing a “Flex Dashboard” template.

If you are not using RStudio, you can create a new flexdashboard R Markdown file from the R console:

rmarkdown::**draft**(

"dashboard.Rmd", template = "flex\_dashboard",

package = "flexdashboard"

)

The full documentation of **flexdashboard** is at <https://rmarkdown.rstudio.com/flexdashboard/>. We will only cover some basic features and usage in this chapter. Dashboards have many features in common with HTML documents (Section [3.1](https://bookdown.org/yihui/rmarkdown/html-document.html#html-document)), such as figure options (Section [3.1.5](https://bookdown.org/yihui/rmarkdown/html-document.html#figure-options)), appearance and style (Section [3.1.4](https://bookdown.org/yihui/rmarkdown/html-document.html#appearance-and-style)), MathJax equations (Section [3.1.8](https://bookdown.org/yihui/rmarkdown/html-document.html#mathjax-equations)), header and before/after body inclusions (Section [3.1.10.2](https://bookdown.org/yihui/rmarkdown/html-document.html#includes)), and Pandoc arguments (Section [3.1.10.5](https://bookdown.org/yihui/rmarkdown/html-document.html#pandoc-arguments)), and so on. We also recommend that you take a look at the R help page ?flexdashboard::flex\_dashboard to learn about more features and options.

References

Iannone, Richard, JJ Allaire, and Barbara Borges. 2018. *Flexdashboard: R Markdown Format for Flexible Dashboards*. [https://CRAN.R-project.org/package=flexdashboard](https://cran.r-project.org/package=flexdashboard).

## **5.1 Layout**

The overall rule about the dashboard layout is that a first-level section generates a page, a second-level section generates a column (or a row), and a third-level section generates a box (that contains one or more dashboard components). Below is a quick example:

---

title: "Get Started"

output: flexdashboard::flex\_dashboard

---

```{r setup, include=FALSE}

library(flexdashboard)

```

Column 1

--------------------------------------------------

### Chart A

```{r}

```

Column 2

--------------------------------------------------

### Chart B

```{r}

```

### Chart C

```{r}

```

Note that a series of dashes under a line of text is the alternative Markdown syntax for the second-level section header, i.e.,

Column 1

--------------------------------------------------

is equivalent to

## Column 1

We used a series of dashes just to make the second-level sections stand out in the source document. By default, the second-level sections generate columns on a dashboard, and the third level sections will be stacked vertically inside columns. You do not have to have columns on a dashboard: when all you have are the third-level sections in the source document, they will be stacked vertically as one column in the output.

The text of the second-level headers will not be displayed in the output. The second-level headers are for the sole purpose of layout, so the actual content of the headers does not matter at all. By contrast, the first-level and third-level headers will be displayed as titles.

Figure [5.1](https://bookdown.org/yihui/rmarkdown/layout.html#fig:dashboard-start) shows the output of the above example, in which you can see two columns, with the first column containing “Chart A”, and the second column containing “Chart B” and “Chart C”. We did not really include any R code in the code chunks, so all boxes are empty. In these code chunks, you may write arbitrary R code that generates R plots, HTML widgets, and various other components to be introduced in Section [5.2](https://bookdown.org/yihui/rmarkdown/dashboard-components.html#dashboard-components).

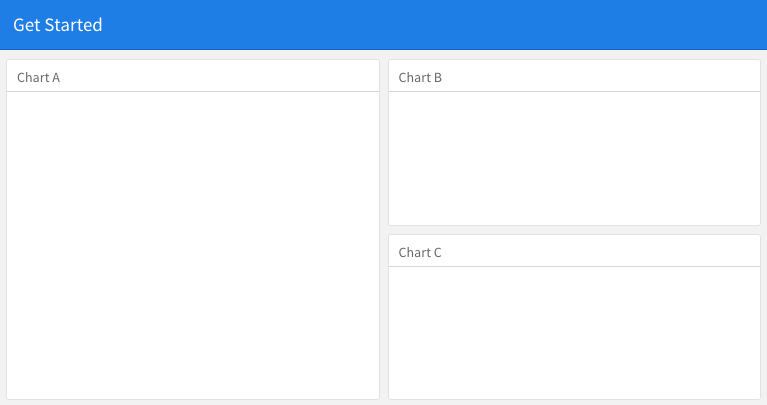


FIGURE 5.1: A quick example of the dashboard layout.

### 5.1.1 Row-based layouts

You may change the column-oriented layout to the row-oriented layout through the orientation option, e.g.,

output:

flexdashboard::flex\_dashboard:

orientation: rows

That means the second-level sections will be rows, and the third-level sections will be arranged as columns within rows.

### 5.1.2 Attributes on sections

The second-level section headers may have attributes on them, e.g., you can set the width of a column to 350:

A narrow column {data-width=350}

--------------------------------

For the row-oriented layout, you can set the data-height attribute for rows. The {.tabset} attribute can be applied on a column so that the third-level sections will be arranged in tabs, e.g.,

Two tabs {.tabset}

------------------

### Tab A

### Tab B

### 5.1.3 Multiple pages

When you have multiple first-level sections in the document, they will be displayed as separate pages on the dashboard. Below is an example, and Figure [5.2](https://bookdown.org/yihui/rmarkdown/layout.html#fig:dashboard-pages) shows the output. Note that a series of equal signs is the alternative Markdown syntax for the first-level section headers (you can use a single pound sign #, too).

---

title: "Multiple Pages"

output: flexdashboard::flex\_dashboard

---

Visualizations {data-icon="fa-signal"}

=====================================

### Chart 1

```{r}

```

### Chart 2

```{r}

```

Tables {data-icon="fa-table"}

=====================================

### Table 1

```{r}

```

### Table 2

```{r}

```

Page titles are displayed as a navigation menu at the top of the dashboard. In this example, we applied icons to page titles through the data-icon attribute. You can find other available icons from [https://fontawesome.com](https://fontawesome.com/).

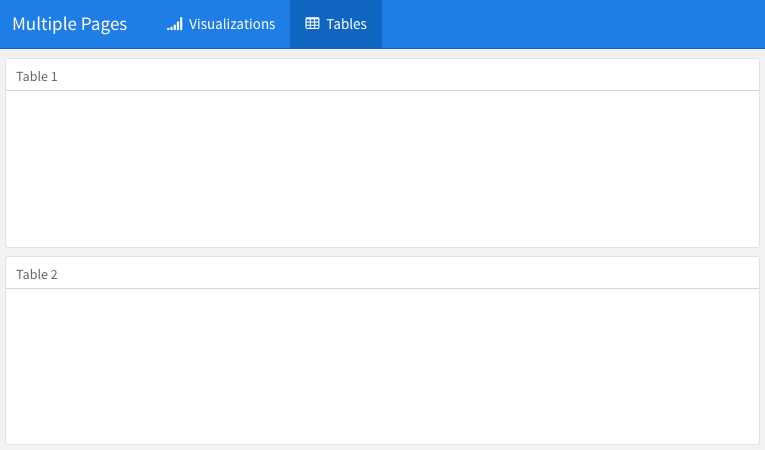


FIGURE 5.2: Multiple pages on a dashboard.

### 5.1.4 Story boards

Besides the column and row-based layouts, you may present a series of visualizations and related commentary through the “storyboard” layout. To enable this layout, you use the option storyboard. Below is an example, and Figure [5.3](https://bookdown.org/yihui/rmarkdown/layout.html#fig:dashboard-story) shows the output, in which you can see left/right navigation buttons at the top to help you go through all visualizations and associated commentaries one by one.

---

title: "Storyboard Commentary"

output:

flexdashboard::flex\_dashboard:

storyboard: true

---

### A nice scatterplot here

```{r}

plot(cars, pch = 20)

grid()

```

---

Some commentary about Frame 1.

### A beautiful histogram on this board

```{r}

hist(faithful$eruptions, col = 'gray', border = 'white', main = '')

```

---

Some commentary about Frame 2.

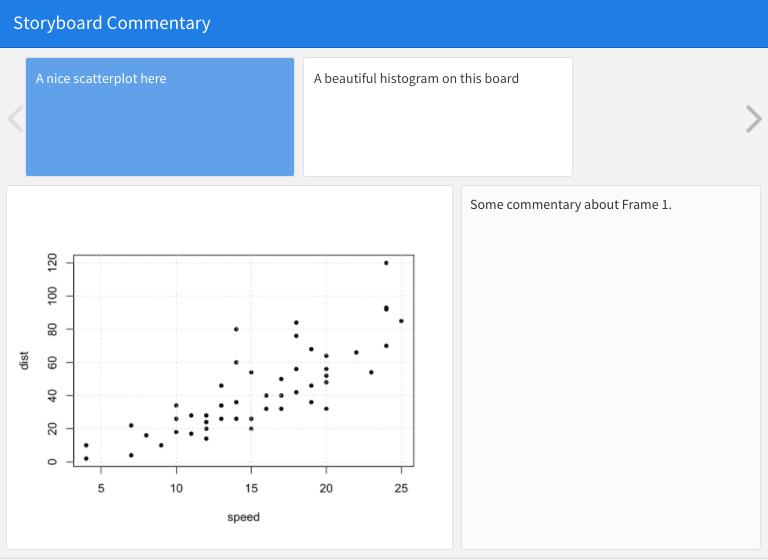


FIGURE 5.3: An example story board.

## **5.2 Components**

A wide variety of components can be included in a dashboard layout, including:

1. Interactive JavaScript data visualizations based on HTML widgets.
2. R graphical output including base, lattice, and grid graphics.
3. Tabular data (with optional sorting, filtering, and paging).
4. Value boxes for highlighting important summary data.
5. Gauges for displaying values on a meter within a specified range.
6. Text annotations of various kinds.
7. A navigation bar to provide more links related to the dashboard.

The first three components work in most R Markdown documents regardless of output formats. Only the latter four are specific to dashboards, and we briefly introduce them in this section.

### 5.2.1 Value boxes

Sometimes you want to include one or more simple values within a dashboard. You can use the valueBox() function in the **flexdashboard** package to display single values along with a title and an optional icon. For example, here are three side-by-side sections, each displaying a single value (see Figure [5.4](https://bookdown.org/yihui/rmarkdown/dashboard-components.html#fig:dashboard-valueboxes) for the output):

---

title: "Dashboard Value Boxes"

output:

flexdashboard::flex\_dashboard:

orientation: rows

---

```{r setup, include=FALSE}

library(flexdashboard)

# these computing functions are only toy examples

computeArticles = function(...) return(45)

computeComments = function(...) return(126)

computeSpam = function(...) return(15)

```

### Articles per Day

```{r}

articles = computeArticles()

valueBox(articles, icon = "fa-pencil")

```

### Comments per Day

```{r}

comments = computeComments()

valueBox(comments, icon = "fa-comments")

```

### Spam per Day

```{r}

spam = computeSpam()

valueBox(

spam, icon = "fa-trash",

color = ifelse(spam > 10, "warning", "primary")

)

```

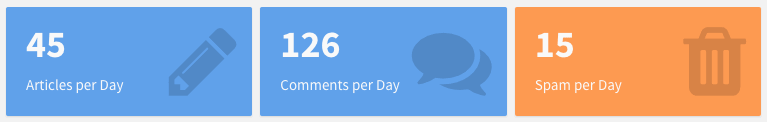


FIGURE 5.4: Three value boxes side by side on a dashboard.

The valueBox() function is called to emit a value and specify an icon.

The third code chunk (“Spam per Day”) makes the background color of the value box dynamic using the color parameter. Available colors include "primary", "info", "success", "warning", and "danger" (the default is "primary"). You can also specify any valid CSS color (e.g., "#ffffff", "rgb(100, 100, 100)", etc.).

### 5.2.2 Gauges

Gauges display values on a meter within a specified range. For example, here is a set of three gauges (see Figure [5.5](https://bookdown.org/yihui/rmarkdown/dashboard-components.html#fig:dashboard-gauges) for the output):

---

title: "Dashboard Gauges"

output:

flexdashboard::flex\_dashboard:

orientation: rows

---

```{r setup, include=FALSE}

library(flexdashboard)

```

### Contact Rate

```{r}

gauge(91, min = 0, max = 100, symbol = '%', gaugeSectors(

success = c(80, 100), warning = c(40, 79), danger = c(0, 39)

))

```

### Average Rating

```{r}

gauge(37.4, min = 0, max = 50, gaugeSectors(

success = c(41, 50), warning = c(21, 40), danger = c(0, 20)

))

```

### Cancellations

```{r}

gauge(7, min = 0, max = 10, gaugeSectors(

success = c(0, 2), warning = c(3, 6), danger = c(7, 10)

))

```

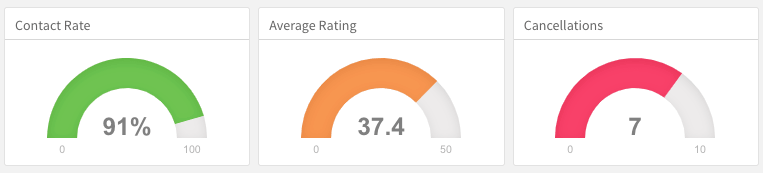


FIGURE 5.5: Three gauges side by side on a dashboard.

There are a few things to note about this example:

1. The gauge() function is used to output a gauge. It has three required arguments: value, min, and max (these can be any numeric values).
2. You can specify an optional symbol to be displayed alongside the value (in the example “%” is used to denote a percentage).
3. You can specify a set of custom color “sectors” using the gaugeSectors() function. By default, the current theme’s “success” color (typically green) is used for the gauge color. The sectors option enables you to specify a set of three value ranges (success, warning, and danger), which cause the gauge’s color to change based on its value.

### 5.2.3 Text annotations

If you need to include additional narrative or explanation within your dashboard, you can do so in the following ways:

1. You can include content at the top of the page before dashboard sections are introduced.
2. You can define dashboard sections that do not include a chart but rather include arbitrary content (text, images, and equations, etc.).

For example, the following dashboard includes some content at the top and a dashboard section that contains only text (see Figure [5.6](https://bookdown.org/yihui/rmarkdown/dashboard-components.html#fig:dashboard-text) for the output):

---

title: "Text Annotations"

output:

flexdashboard::flex\_dashboard:

orientation: rows

---

Monthly deaths from bronchitis, emphysema and asthma in the

UK, 1974–1979 (Source: P. J. Diggle, 1990, Time Series: A

Biostatistical Introduction. Oxford, table A.3)

```{r setup, include=FALSE}

library(dygraphs)

```

Row {data-height=600}

-------------------------------------

### All Lung Deaths

```{r}

dygraph(ldeaths)

```

Row {data-height=400}

-------------------------------------

### Male Deaths

```{r}

dygraph(mdeaths)

```

> Monthly deaths from lung disease in the UK, 1974–1979

### About dygraphs

This example makes use of the dygraphs R package. The dygraphs

package provides rich facilities for charting time-series data

in R. You can use dygraphs at the R console, within R Markdown

documents, and within Shiny applications.

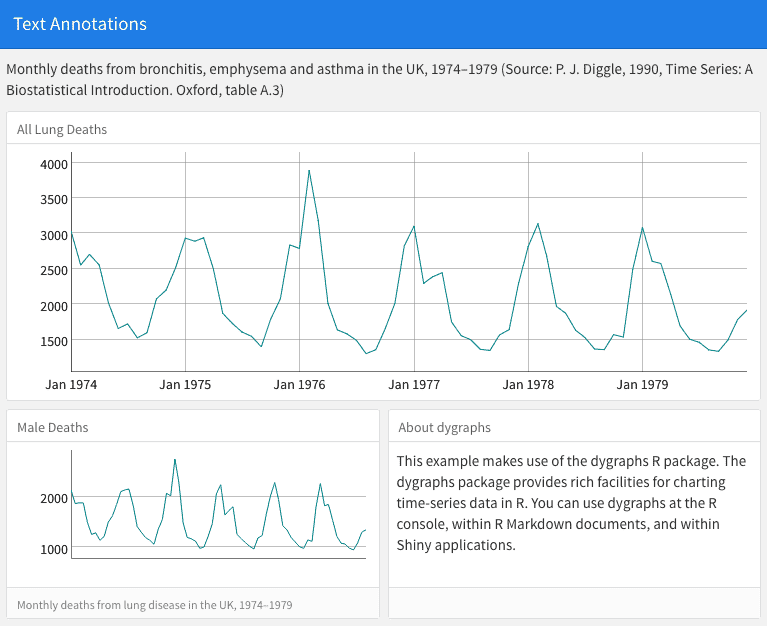


FIGURE 5.6: Text annotations on a dashboard.

Each component within a dashboard includes optional title and notes sections. The title is simply the text after the third-level (###) section heading. The notes are any text prefaced with > after the code chunk that yields the component’s output (see the second component of the above example).

You can exclude the title entirely by applying the .no-title attribute to a section heading.

### 5.2.4 Navigation bar

By default, the dashboard navigation bar includes the document’s title, author, and date. When a dashboard has multiple pages (Section [5.1.3](https://bookdown.org/yihui/rmarkdown/layout.html#multiple-pages)), links to the various pages are also included on the left side of the navigation bar. You can also add social links to the dashboard.

In addition, you can add custom links to the navigation bar using the navbar option. For example, the following options add an “About” link to the navigation bar:

---

title: "Navigation Bar"

output:

flexdashboard::flex\_dashboard:

navbar:

**-** **{** title: "About"**,** href: "https://example.com/about" **}**

---

Navigation bar items must include either a title or icon field (or both). You should also include a href as the navigation target. The align field is optional (it can be “left” or “right” and defaults to “right”).

You can include links to social sharing services via the social option. For example, the following dashboard includes Twitter and Facebook links as well as a drop-down menu with a more complete list of services:

---

title: "Social Links"

output:

flexdashboard::flex\_dashboard:

social: **[** "twitter"**,** "facebook"**,** "menu" **]**

---

The social option can include any number of the following services: "facebook", "twitter", "google-plus", "linkedin", and "pinterest". You can also specify "menu" to provide a generic sharing drop-down menu that includes all of the services.

## **5.3 Shiny**

By adding Shiny to a dashboard, you can let viewers change underlying parameters and see the results immediately, or let dashboards update themselves incrementally as their underlying data changes (see functions reactiveFileReader() and reactivePoll() in the **shiny** package). This is done by adding runtime: shiny to a standard dashboard document, and then adding one or more input controls and/or reactive expressions that dynamically drive the appearance of the components within the dashboard.

Using Shiny with **flexdashboard** turns a static R Markdown report into an interactive document. It is important to note that interactive documents need to be deployed to a Shiny Server to be shared broadly (whereas static R Markdown documents are standalone web pages that can be attached to emails or served from any standard web server).

Note that the [**shinydashboard**](https://rstudio.github.io/shinydashboard/) package provides another way to create dashboards with Shiny.

### 5.3.1 Getting started

The steps required to add Shiny components to a dashboard are:

1. Add runtime: shiny to the options declared at the top of the document (YAML metadata).
2. Add the {.sidebar} attribute to the first column of the dashboard to make it a host for Shiny input controls (note that this step is not strictly required, but this will generate a typical layout for Shiny-based dashboards).
3. Add Shiny inputs and outputs as appropriate.
4. When including plots, be sure to wrap them in a call to renderPlot(). This is important not only for dynamically responding to changes, but also to ensure that they are automatically re-sized when their container changes.

### 5.3.2 A Shiny dashboard example

Here is a simple example of a dashboard that uses Shiny (see Figure [5.7](https://bookdown.org/yihui/rmarkdown/shiny.html#fig:dashboard-shiny) for the output):

---

title: "Old Faithful Eruptions"

output: flexdashboard::flex\_dashboard

runtime: shiny

---

```{r global, include=FALSE}

# load data in 'global' chunk so it can be shared

# by all users of the dashboard

library(datasets)

data(faithful)

```

Column {.sidebar}

--------------------------------------------------

Waiting time between eruptions and the duration of the eruption

for the Old Faithful geyser in Yellowstone National Park,

Wyoming, USA.

```{r}

selectInput(

"n\_breaks", label = "Number of bins:",

choices = c(10, 20, 35, 50), selected = 20

)

sliderInput(

"bw\_adjust", label = "Bandwidth adjustment:",

min = 0.2, max = 2, value = 1, step = 0.2

)

```

Column

--------------------------------------------------

### Geyser Eruption Duration

```{r}

renderPlot({

erpt = faithful$eruptions

hist(

erpt, probability = TRUE, breaks = as.integer(input$n\_breaks),

xlab = "Duration (minutes)", main = "Geyser Eruption Duration",

col = 'gray', border = 'white'

)

dens = density(erpt, adjust = input$bw\_adjust)

lines(dens, col = "blue", lwd = 2)

})

```

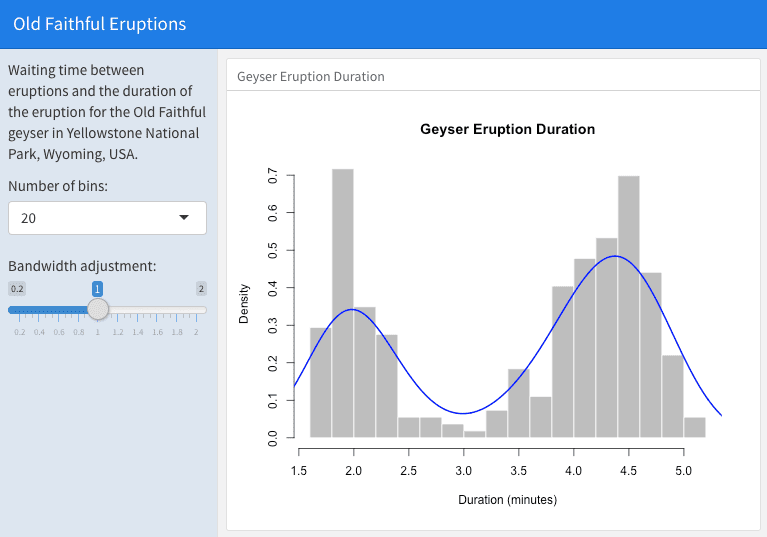


FIGURE 5.7: An interactive dashboard based on Shiny.

The first column includes the {.sidebar} attribute and two Shiny input controls; the second column includes the Shiny code required to render the chart based on the inputs.

One important thing to note about this example is the chunk labeled global at the top of the document. The global chunk has special behavior within **flexdashboard**: it is executed only once within the global environment, so that its results (e.g., data frames read from disk) can be accessed by all users of a multi-user dashboard. Loading your data within a global chunk will result in substantially better startup performance for your users, and hence is highly recommended.

### 5.3.3 Input sidebar

You add an input sidebar to a flexdashboard by adding the {.sidebar} attribute to a column, which indicates that it should be laid out flush to the left with a default width of 250 pixels and a special background color. Sidebars always appear on the left no matter where they are defined within the flow of the document.

If you are creating a dashboard with multiple pages, you may want to use a single sidebar that applies across all pages. In this case, you should define the sidebar using a first-level Markdown header.

### 5.3.4 Learning more

Below are some good resources for learning more about Shiny and creating interactive documents:

1. The official Shiny website ([http://shiny.rstudio.com](http://shiny.rstudio.com/)) includes extensive articles, tutorials, and examples to help you learn more about Shiny.
2. The article “[Introduction to Interactive Documents](http://shiny.rstudio.com/articles/interactive-docs.html)” on the Shiny website is a great guide for getting started with Shiny and R Markdown.
3. For deploying interactive documents, you may consider Shiny Server or RStudio Connect: <https://www.rstudio.com/products/shiny/shiny-server/>.

Chapter 6 Tufte Handouts

The Tufte handout style is a style that [Edward Tufte](https://en.wikipedia.org/wiki/Edward_Tufte) uses in his books and handouts. Tufte’s style is known for its extensive use of sidenotes, tight integration of graphics with text, and well-set typography. This style has been implemented in LaTeX and HTML/CSS,[5](https://bookdown.org/yihui/rmarkdown/tufte-handouts.html#fn5) respectively. Both implementations have been ported into the **tufte** package (Xie and Allaire [2019](https://bookdown.org/yihui/rmarkdown/tufte-handouts.html#ref-R-tufte)). If you want LaTeX/PDF output, you may use the tufte\_handout format for handouts, and tufte\_book for books. For HTML output, use tufte\_html, e.g.,

---

title: "An Example Using the Tufte Style"

author: "John Smith"

output:

tufte::tufte\_handout: default

tufte::tufte\_html: default

---

Figure [6.1](https://bookdown.org/yihui/rmarkdown/tufte-handouts.html#fig:tufte-overview) shows the basic layout of the Tufte style, in which you can see a main column on the left that contains the body of the document, and a side column on the right to display sidenotes.

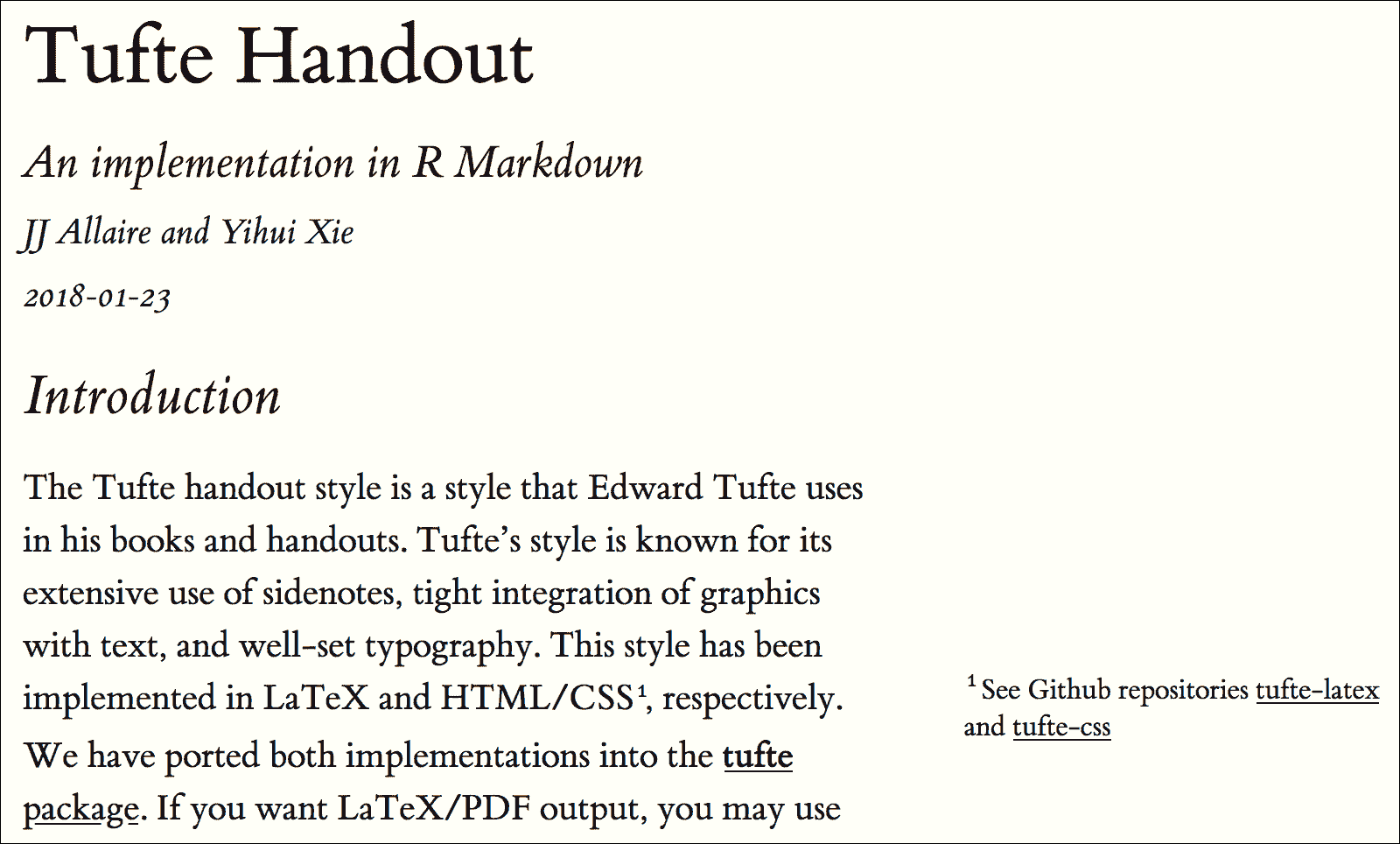


FIGURE 6.1: The basic layout of the Tufte style.

There are two goals for the **tufte** package:

1. To produce both PDF and HTML output with similar styles from the same R Markdown document.
2. To provide simple syntax to write elements of the Tufte style such as side notes and margin figures. For example, when you want a margin figure, all you need to do is the chunk option fig.margin = TRUE, and **tufte** will take care of the details for you, so you never need to think about LaTeX environments like \begin{marginfigure} \end{marginfigure} or HTML tags like <span class="marginfigure"> </span>; the LaTeX and HTML code under the hood may be complicated, but you never need to learn or write such code.

You can use the wizard in RStudio IDE from the menu File -> New File -> R Markdown -> From Template to create a new R Markdown document with a default example provided by the **tufte** package. Note that you need a LaTeX distribution if you want PDF output (see Chapter [1](https://bookdown.org/yihui/rmarkdown/installation.html#installation)).

References

Xie, Yihui, and JJ Allaire. 2019. *Tufte: Tufte’s Styles for R Markdown Documents*. [https://CRAN.R-project.org/package=tufte](https://cran.r-project.org/package=tufte).

1. See Github repositories <https://github.com/tufte-latex/tufte-latex> and <https://github.com/edwardtufte/tufte-css>.[↩](https://bookdown.org/yihui/rmarkdown/tufte-handouts.html#fnref5)

## **6.1 Headings**

The Tufte style provides the first and second-level headings (that is, # and ##), demonstrated in the next section. You may get unexpected output (and even errors) if you try to use ### and smaller headings.

In his later books,[6](https://bookdown.org/yihui/rmarkdown/tufte-headings.html#fn6), Tufte starts each section with a bit of vertical space, a non-indented paragraph, and sets the first few words of the sentence in small caps. To accomplish this using this style, call the newthought() function in **tufte** in an inline R expression `r `. Note that you should not assume **tufte** has been attached to your R session. You should either use library(tufte) in your R Markdown document before you call newthought(), or use tufte::newthought().

1. Such as “Beautiful Evidence”: <http://www.edwardtufte.com/tufte/books_be>.[↩](https://bookdown.org/yihui/rmarkdown/tufte-headings.html#fnref6)

## **6.2 Figures**

### 6.2.1 Margin figures

Images and graphics play an integral role in Tufte’s work. To place figures in the margin, you can use the **knitr** chunk option fig.margin = TRUE. For example:

```{r fig-margin, fig.margin=TRUE}

plot(cars)

```

As in other Rmd documents, you can use the fig.cap chunk option to provide a figure caption, and adjust figure sizes using the fig.width and fig.height chunk options, which are specified in inches, and will be automatically scaled down to fit within the handout margin.

Figure [6.2](https://bookdown.org/yihui/rmarkdown/tufte-figures.html#fig:tufte-margin) shows what a margin figure looks like.

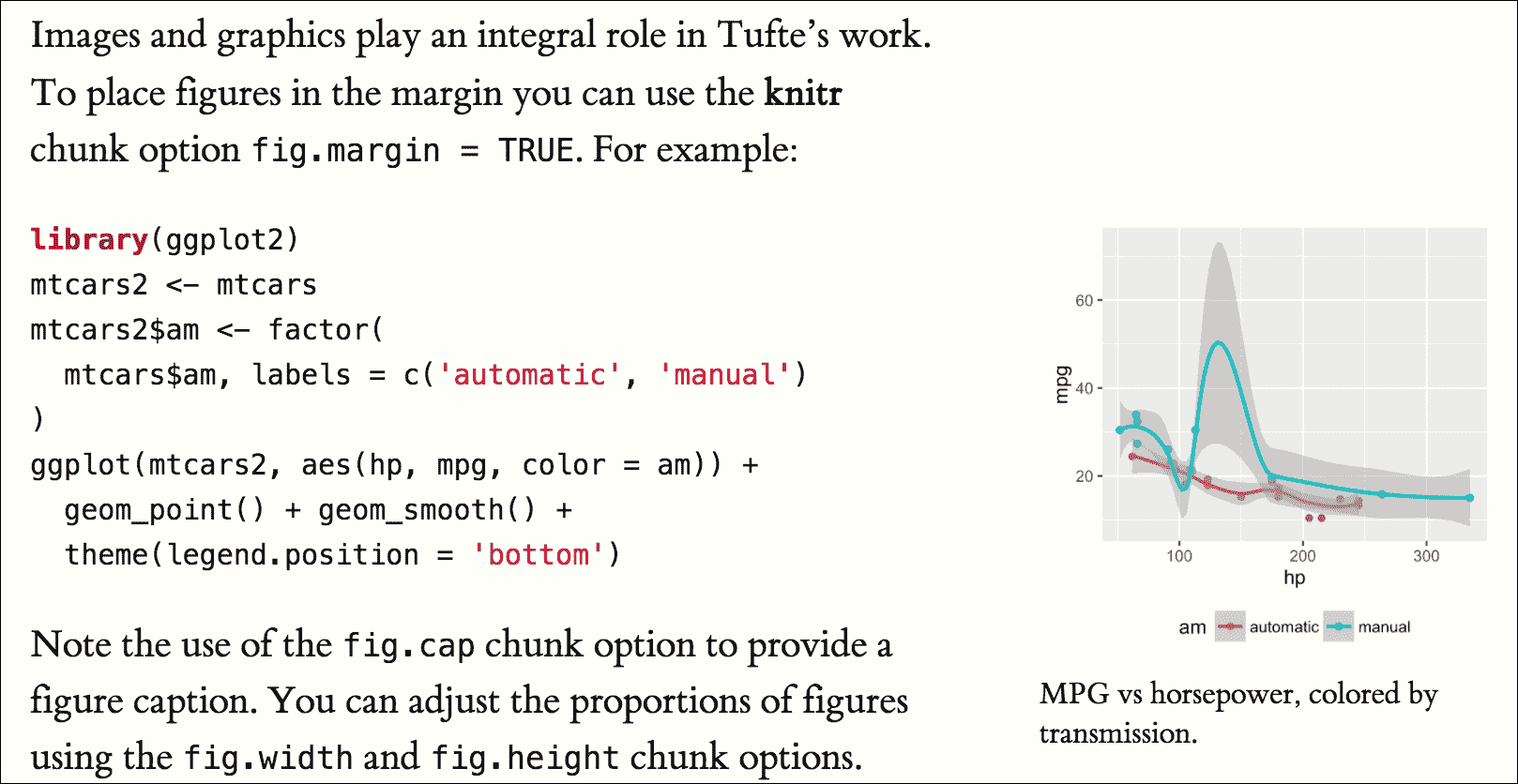


FIGURE 6.2: A margin figure in the Tufte style.

### 6.2.2 Arbitrary margin content

You can include anything in the margin using the **knitr** engine named marginfigure. Unlike R code chunks ```{r}, you write a chunk starting with ```{marginfigure} instead, then put the content in the chunk, e.g.,

```{marginfigure}

We know from \_the first fundamental theorem of calculus\_ that

for $x$ in $[a, b]$:

$$\frac{d}{dx}\left( \int\_{a}^{x} f(u)\,du\right)=f(x).$$

```

For the sake of portability between LaTeX and HTML, you should keep the margin content as simple as possible (syntax-wise) in the marginfigure blocks. You may use simple Markdown syntax like \*\*bold\*\* and \_italic\_ text, but please refrain from using footnotes, citations, or block-level elements (e.g., blockquotes and lists) there.

Note that if you set echo = FALSE in your global chunk options, you will have to add echo = TRUE to the chunk to display a margin figure, for example ```{marginfigure, echo = TRUE}.

### 6.2.3 Full-width figures

You can arrange for figures to span across the entire page by using the chunk option fig.fullwidth = TRUE, e.g.,

```{r, fig.width=10, fig.height=2, fig.fullwidth=TRUE}

par(mar = c(4, 4, .1, .2)); plot(sunspots)

```

Other chunk options related to figures can still be used, such as fig.width, fig.cap, and out.width, etc. For full-width figures, usually fig.width is large and fig.height is small. In the above example, the plot size is 10x2.

Figure [6.3](https://bookdown.org/yihui/rmarkdown/tufte-figures.html#fig:tufte-full) shows what a full-width figure looks like.

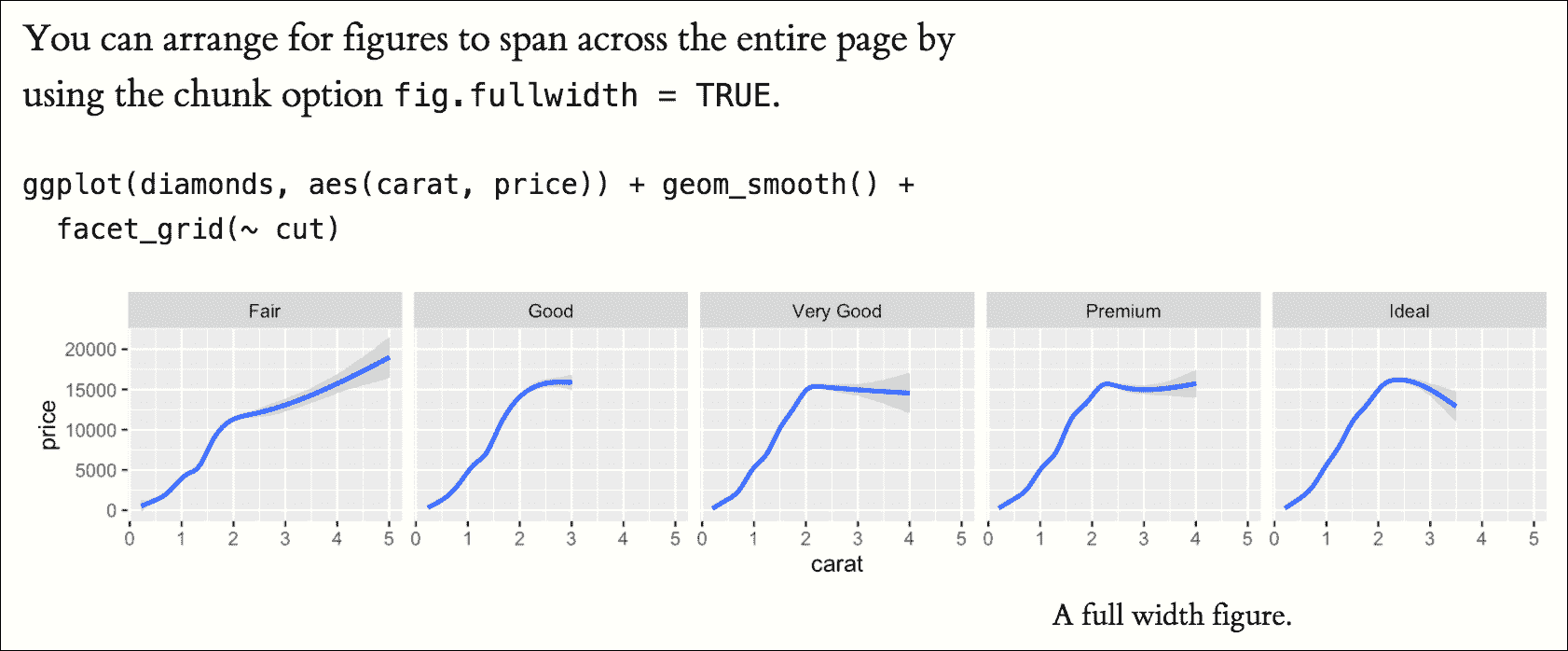


FIGURE 6.3: A full-width figure in the Tufte style.

### 6.2.4 Main column figures

Besides margin and full-width figures, you can certainly also include figures constrained to the main column. This is the default type of figures in the LaTeX/HTML output, and requires no special chunk options.

Figure [6.4](https://bookdown.org/yihui/rmarkdown/tufte-figures.html#fig:tufte-main) shows what a figure looks like in the main column.

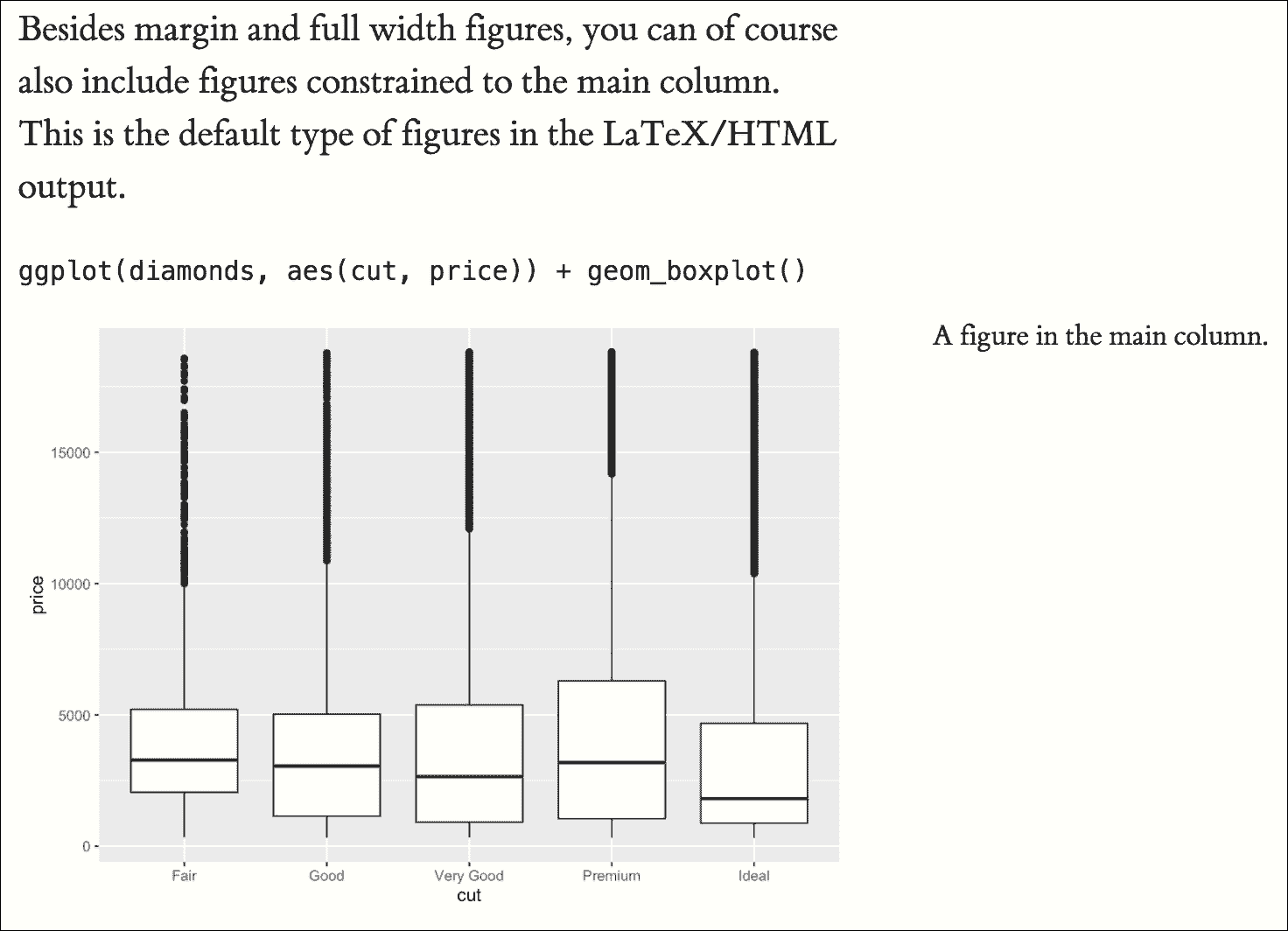


FIGURE 6.4: A figure in the main column in the Tufte style.

## **6.3 Sidenotes**

One of the most prominent and distinctive features of this style is the extensive use of sidenotes. There is a wide margin to provide ample room for sidenotes and small figures. Any use of a footnote, of which the Markdown syntax is ^[footnote content], will automatically be converted to a sidenote.

If you would like to place ancillary information in the margin without the sidenote mark (the superscript number), you can use the margin\_note() function from **tufte** in an inline R expression. This function does not process the text with Pandoc, so Markdown syntax will not work here. If you need to write anything in Markdown syntax, please use the marginfigure block described previously.

## **6.4 References**

References can be displayed as margin notes for HTML output. To enable this feature, you must set link-citations: yes in the YAML metadata, and the version of pandoc-citeproc should be at least 0.7.2. To check the version of pandoc-citeproc in your system, you may run this in R:

**system2**("pandoc-citeproc", "--version")

If your version of pandoc-citeproc is too low, or you did not set link-citations: yes in YAML, references in the HTML output will be placed at the end of the output document.

You can also explicitly disable this feature via the margin\_references option, e.g.,

---

output:

tufte::tufte\_html:

margin\_references: false

---

## **6.5 Tables**

You can use the kable() function from the **knitr** package to format tables that integrate well with the rest of the Tufte handout style. The table captions are placed in the margin like figures in the HTML output. A simple example (Figure [6.5](https://bookdown.org/yihui/rmarkdown/tufte-tables.html#fig:tufte-table) shows the output):

```{r}

knitr::kable(

mtcars[1:6, 1:6], caption = 'A subset of mtcars.'

)

```

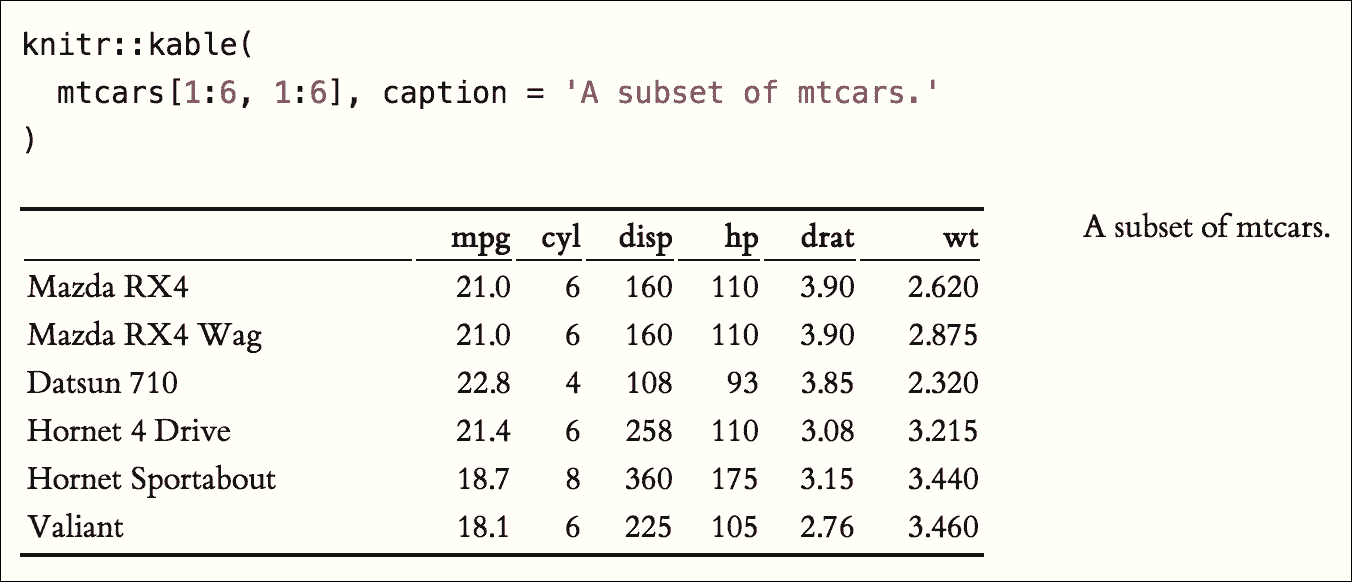


FIGURE 6.5: A table in the Tufte style.

## **6.6 Block quotes**

We know from the Markdown syntax that paragraphs that start with > are converted to block quotes. If you want to add a right-aligned footer for the quote, you may use the function quote\_footer() from **tufte** in an inline R expression. Here is an example:

> "If it weren't for my lawyer, I'd still be in prison.

> It went a lot faster with two people digging."

>

> `r tufte::quote\_footer('--- Joe Martin')`

## **6.7 Responsiveness**

The HTML page is responsive in the sense that when the page width is smaller than 760px, sidenotes and margin notes will be hidden by default. For sidenotes, you can click their numbers (the superscripts) to toggle their visibility. For margin notes, you may click the circled plus signs to toggle visibility (see Figure [6.6](https://bookdown.org/yihui/rmarkdown/tufte-responsiveness.html#fig:tufte-responsive)).

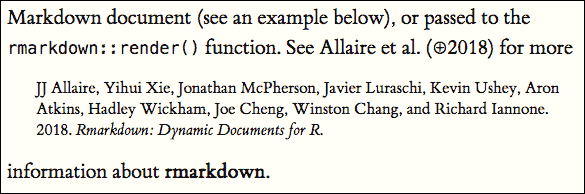


FIGURE 6.6: The Tufte HTML style on narrow screens.

## **6.8 Sans-serif fonts and epigraphs**

There are a few other things in Tufte CSS that we have not mentioned so far. If you prefer sans-serif fonts, use the function sans\_serif() in **tufte**. For epigraphs, you may use a pair of underscores to make the paragraph italic in a block quote, e.g.,

> \_I can win an argument on any topic, against any opponent.

> People know this, and steer clear of me at parties. Often,

> as a sign of their great respect, they don't even invite me.\_

>

> `r quote\_footer('--- Dave Barry')`

## **6.9 Customize CSS styles**

You can turn on/off some features of the Tufte style in HTML output. The default features enabled are:

---

output:

tufte::tufte\_html:

tufte\_features: **[**"fonts"**,** "background"**,** "italics"**]**

---

If you do not want the page background to be lightyellow, you can remove background from tufte\_features. You can also customize the style of the HTML page via a CSS file. For example, if you do not want the subtitle to be italic, you can define

h3.subtitle em {

**font-style**: normal;

}

in, say, a CSS file my-style.css (under the same directory of your Rmd document), and apply it to your HTML output via the css option, e.g.,

---

output:

tufte::tufte\_html:

tufte\_features: **[**"fonts"**,** "background"**]**

css: "my-style.css"

---

There is also a variant of the Tufte style in HTML/CSS named “[Envisioned CSS](http://nogginfuel.com/envisioned-css/)”. This style can be enabled by specifying the argument tufte\_variant = 'envisioned' in tufte\_html(),[7](https://bookdown.org/yihui/rmarkdown/tufte-css.html#fn7), e.g.,

---

output:

tufte::tufte\_html:

tufte\_variant: "envisioned"

---

You can see a live example at <https://rstudio.github.io/tufte/>. It is also available in Simplified Chinese: <https://rstudio.github.io/tufte/cn/>, and its envisioned style can be found at <https://rstudio.github.io/tufte/envisioned/>.

1. The actual Envisioned CSS was not used in the **tufte** package. Only the fonts, background color, and text color are changed based on the default Tufte style.[↩](https://bookdown.org/yihui/rmarkdown/tufte-css.html#fnref7)

Chapter 7 xaringan Presentations

We have introduced a few HTML5 presentation formats in Chapter [4](https://bookdown.org/yihui/rmarkdown/presentations.html#presentations). The **xaringan** package (Xie [2019f](https://bookdown.org/yihui/rmarkdown/xaringan.html#ref-R-xaringan)) is an R Markdown extension based on the JavaScript library remark.js ([https://remarkjs.com](https://remarkjs.com/)) to generate HTML5 presentations of a different style. See Figure [7.1](https://bookdown.org/yihui/rmarkdown/xaringan.html#fig:xaringan-sample) for two sample slides.

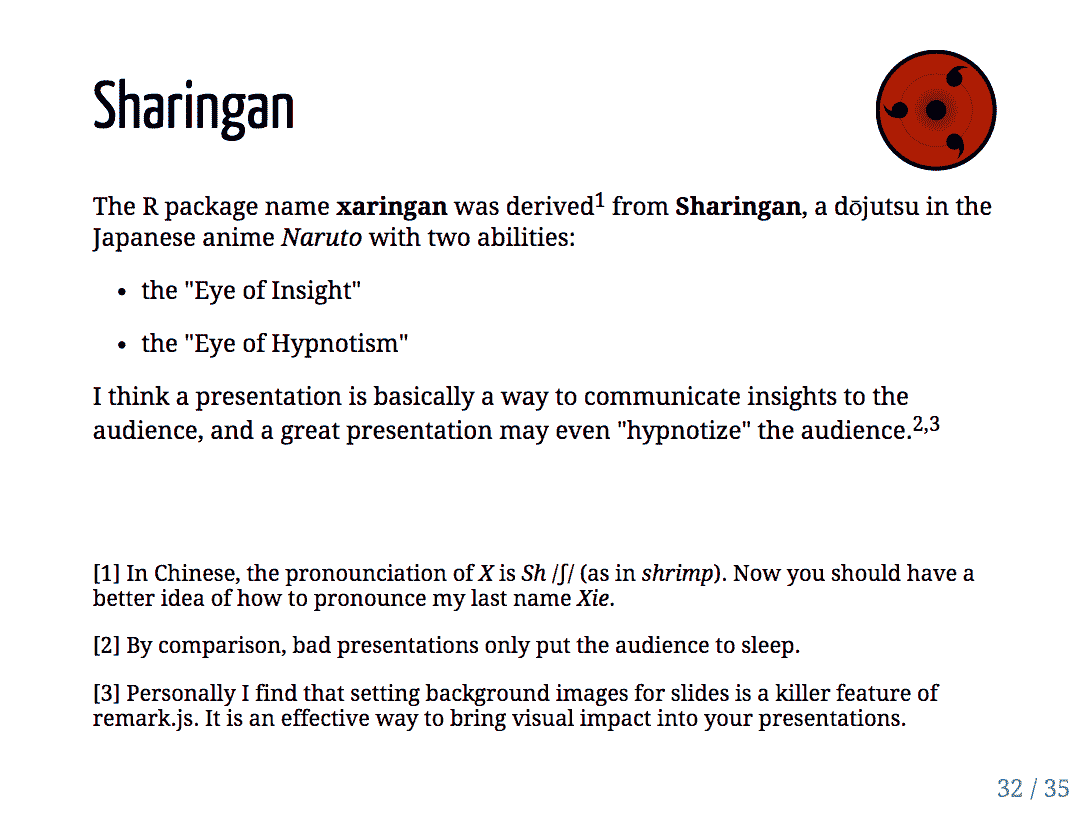


FIGURE 7.1: Two sample slides created from the xaringan package.

The name “xaringan” came from Sharingan (<http://naruto.wikia.com/wiki/Sharingan>) in the Japanese manga and anime “Naruto”. The word was deliberately chosen to be difficult to pronounce for most people (unless you have watched the anime), because its author (me) loved the style very much, and was concerned that it would become too popular.[8](https://bookdown.org/yihui/rmarkdown/xaringan.html#fn8) The concern was somewhat naive, because the style is actually very customizable, and users started to contribute more themes to the package later.

The **xaringan** package is based on the JavaScript library remark.js ([https://remarkjs.com](https://remarkjs.com/)); remark.js only supports Markdown, and **xaringan** added the support for R Markdown as well as other utilities to make it easier to build and preview slides.

You can learn more about the background stories and the usage of the **xaringan** package from the documentation at <http://slides.yihui.name/xaringan/>, which is actually a set of slides generated from **xaringan**. You may also read a potentially biased blog post of mine to know why I preferred **xaringan** / remark.js for HTML5 presentations: <https://yihui.name/en/2017/08/why-xaringan-remark-js/>.

References

Xie, Yihui. 2019f. *Xaringan: Presentation Ninja*. [https://CRAN.R-project.org/package=xaringan](https://cran.r-project.org/package=xaringan).

1. The main reason I stopped using LaTeX Beamer slides was because of its popularity: when you attend academic conferences, you see Beamer slides everywhere.[↩](https://bookdown.org/yihui/rmarkdown/xaringan.html#fnref8)

## **7.1 Get started**

You can install either the CRAN version or the development version on GitHub (<https://github.com/yihui/xaringan>):

*# install from CRAN*

**install.packages**('xaringan')

*# or GitHub*

devtools::**install\_github**('yihui/xaringan')

If you use RStudio, it is easy to get started from the menu File -> New File -> R Markdown -> From Template -> Ninja Presentation, and you will see an R Markdown example in the editor. Press the Knit button to compile it, or use the RStudio addin Infinite Moon Reader to live preview the slides: every time you update and save the Rmd document, the slides will be automatically reloaded.

The main R Markdown output format in this package is moon\_reader(). See the R help page ?xaringan::moon\_reader for all possible configurations. Below is a quick example:

---

title: "Presentation Ninja"

subtitle: "with xaringan"

author: "Yihui Xie"

date: "2016/12/12"

output:

xaringan::moon\_reader:

lib\_dir: libs

nature:

highlightStyle: github

countIncrementalSlides: false

---

One slide.

---

Another slide.

## **7.2 Keyboard shortcuts**

After opening slides generated from **xaringan** or remark.js, you may press the key h (Help) or ? on your keyboard to learn all possible keyboard shortcuts, which may help you better present your slides.

* To go the previous slide, you may press Up/Left arrows, PageUp, or k.
* To go the next slide, you may press Right/Down arrows, PageDown, Space, or j.
* You may press Home to go to the first slide, or End to go to the last slide, if you have these keys.
* Typing a number and pressing Return (or Enter), you can jump to a specific slide with that page number.
* Press b to black out a slide, and m to “mirror” a slide (reverse everything on the slide). These techniques can be useful when you do not want the audience to read the slide, e.g., when you have solutions on a slide but do not want to show them to your students immediately. I encourage you to try m; it can be a lot of fun. You can press these keys again to resume the normal slide.
* Press f to toggle the fullscreen mode.
* Press c to clone the slides to a new browser window; slides in the two windows will be in sync as you navigate through them. Press p to toggle the presenter mode. The presenter mode shows thumbnails of the current slide and the next slide on the left, presenter notes on the right (see Section [7.3.5](https://bookdown.org/yihui/rmarkdown/xaringan-format.html#xaringan-notes)), and also a timer on the top right. The keys c and p can be very useful when you present with your own computer connected to a second screen (such as a projector). On the second screen, you can show the normal slides, while cloning the slides to your own computer screen and using the presenter mode. Only you can see the presenter mode, which means only you can see presenter notes and the time, and preview the next slide. You may press t to restart the timer at any time.
* Press h or ? again to exit the help page.

## **7.3 Slide formatting**

The [remark.js Wiki](https://github.com/gnab/remark/wiki) contains detailed documentation about how to format slides and use the presentation (keyboard shortcuts). The **xaringan** package has simplified several things compared to the official remark.js guide, e.g., you do not need a boilerplate HTML file, you can set the autoplay mode via an option of moon\_reader(), and LaTeX math basically just works.

Please note that remark.js has its own Markdown interpreter that is not compatible with Pandoc’s Markdown converter, so you will not be able to use any advanced Pandoc Markdown features (e.g., the citation syntax [@key]). You may use raw HTML when there is something you desire that is not supported by remark.js. For example, you can generate an HTML table via knitr::kable(head(iris), 'html').

### 7.3.1 Slides and properties

Every new slide is created under a horizontal rule (---). The content of the slide can be arbitrary, e.g., it does not have to have a slide title, and if it does, the title can be of any level you prefer (#, ##, or ###).

A slide can have a few properties, including class and background-image, etc. Properties are written in the beginning of a slide, e.g.,

---

class: center, inverse

background-image: url("images/cool.png")

# A new slide

Content.

The class property assigns class names to the HTML tag of the slide, so that you can use CSS to style specific slides. For example, for a slide with the inverse class, you may define the CSS rules (to render text in white on a dark background):

.inverse {

**background-color**: #272822;

**color**: #d6d6d6;

**text-shadow**: 0 0 20px #333;

}

Then include the CSS file (say, my-style.css) via the css option of xaringan::moon\_reader:

---

output:

xaringan::moon\_reader:

css: "my-style.css"

---

Actually the style for the inverse class has been defined in the default theme of **xaringan**, so you do not really need to define it again unless you want to override it.

Other available class names are left, center, and right for the horizontal alignment of all elements on a slide, and top, middle, and bottom for the vertical alignment.

Background images can be set via the background-image property. The image can be either a local file or an online image. The path should be put inside url(), which is the CSS syntax. You can also set the background image size and position, e.g.,

background-image: url("`r xaringan:::karl`")

background-position: center

background-size: contain

All these properties require you to understand CSS.[9](https://bookdown.org/yihui/rmarkdown/xaringan-format.html#fn9) In the above example, we actually used an inline expression xaringan::karl to return a URL of an image of Karl Broman ([http://kbroman.org](http://kbroman.org/)), which is one of the highlights of the **xaringan** package.

### 7.3.2 The title slide

There is a special slide, the title slide, that is automatically generated from the YAML metadata of your Rmd document. It contains the title, subtitle, author, and date (all are optional). This slide has the classes inverse, center, middle, and title-slide by default, which looks like the left image in Figure [7.1](https://bookdown.org/yihui/rmarkdown/xaringan.html#fig:xaringan-sample). If you do not like the default style, you may either customize the .title-slide class, or provide a custom vector of classes via the titleSlideClass option under the nature option, e.g.,

---

output:

xaringan::moon\_reader:

nature:

titleSlideClass: **[**"right"**,** "top"**,** "my-title"**]**

---

You can also disable the automatic title slide via the seal option and create one manually by yourself:

---

output:

xaringan::moon\_reader:

seal: false

---

*# My Own Title*

*### Author*

Whatever you want to put on the title slide.

### 7.3.3 Content classes

You can assign classes to any elements on a slide, too. The syntax is .className[content]. This is a very powerful feature of remark.js, and one of very few features not available in Pandoc. Basically it makes it possible to style any elements on a slide via CSS. There are a few built-in content classes, .left[ ], .center[ ], and .right[ ], to align elements horizontally on a slide, e.g., you may center an image:

.center[![description of the image](images/foo.png)]

The content inside [ ] can be anything, such as several paragraphs, or lists. The default theme of **xaringan** has provided four more content classes:

* .left-column[ ] and .right-column[ ] provide a sidebar layout. The left sidebar is narrow (20% of the slide width), and the right column is the main column (75% of the slide width). If you have multiple level-2 (##) or level-3 (###) headings in the left column, the last heading will be highlighted, with previous headings being grayed out.
* .pull-left[ ] and .pull-right[ ] provide a two-column layout, and the two columns are of the same width. Below is an example:
* .pull-left[
* - One bullet.
* - Another bullet.
* ]
* .pull-right[
* **![an image](foo.png)**

]

You can design your own content classes if you know CSS, e.g., if you want to make text red via .red[ ], you may define this in CSS:

.red { **color**: red; }

### 7.3.4 Incremental slides

When you want to show content incrementally on a slide (e.g., holding a funny picture until the last moment), you can use two dashes to separate the content. The two dashes can appear anywhere except inside content classes, so you can basically split your content in any way you like, e.g.,

---

# Two dashes

The easiest way to build incremental slides is...

--

to use two dashes `--` to separate content on a slide.

--

You can divide a slide in \_any way you want\_.

--

- One bullet

- Another bullet

--

- And one more

--

.center[

**![Saw](https://slides.yihui.name/gif/saw-branch.gif)**

Don't saw your slides too hard.

]

There are a few other advanced ways to build incremental slides documented in the presentation at <https://slides.yihui.name/xaringan/incremental.html>.

### 7.3.5 Presenter notes

You can write notes for yourself to read in the presenter mode (press the keyboard shortcut p). These notes are written under three question marks ??? after a slide, and the syntax is also Markdown, which means you can write any elements supported by Markdown, such as paragraphs, lists, images, and so on. For example:

---

The holy passion of Friendship is of so sweet and steady

and loyal and enduring a nature that it will last through

a whole lifetime...

???

\_if not asked to lend money\_.

--- Mark Twain

A common mistake in presentations, especially for presenters without much experience, is to stuff a slide with too much content. The consequence is either a speaker, out of breath, reading the so many words out loud, or the audience starting to read the slides quietly by themselves without listening. Slides are not papers or books, so you should try to be brief in the visual content of slides but verbose in verbal narratives. If you have a lot to say about a slide, but cannot remember everything, you may consider using presenter notes.

I want to mention a technical note about the presenter mode: when connecting to a projector, you should make sure not to mirror the two screens. Instead, separate the two displays, so you can drag the window with the normal view of slides to the second screen. Figure [7.2](https://bookdown.org/yihui/rmarkdown/xaringan-format.html#fig:mirror-display) shows how to do it from the “System Preferences” on macOS (do not check the box “Mirror Displays”).

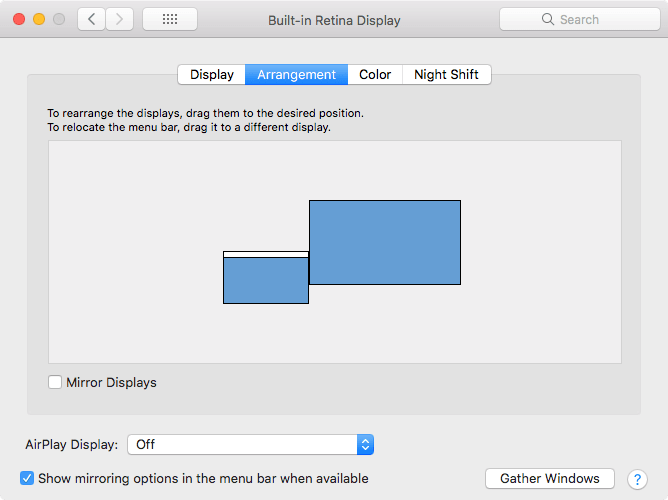


FIGURE 7.2: Separate the current display from the external display.

### 7.3.6 yolo: true

Inspired by a random feature request from [a tweet by Karthik Ram](https://twitter.com/_inundata/status/798970002992873472), the output format xaringan::moon\_reader provided an option named yolo (an acronym of “you only live once”). If you set it to true, a photo of Karl Broman (with a mustache) will be inserted into a random slide in your presentation.[10](https://bookdown.org/yihui/rmarkdown/xaringan-format.html#fn10)

---

output:

xaringan::moon\_reader:

yolo: true

---

The **xaringan** package is probably best known for this feature. I want to thank Karl for letting me use this photo. It always makes me happy for mysterious reasons.

A less well-known feature is that you can actually replace Karl’s picture with other pictures, and/or specify how many times you want a picture to randomly show up in your presentation. For example:

---

output:

xaringan::moon\_reader:

yolo:

img: kangaroo.jpg

times: 5

---

Developing software is fun, isn’t it?

1. There are many tutorials online if you search for “CSS background”, e.g., <https://www.w3schools.com/cssref/css3_pr_background.asp>.[↩](https://bookdown.org/yihui/rmarkdown/xaringan-format.html#fnref9)
2. For the full story behind the mustache, see Karl’s post at <http://kbroman.org/blog/2014/08/28/the-mustache-photo/>.[↩](https://bookdown.org/yihui/rmarkdown/xaringan-format.html#fnref10)

## **7.4 Build and preview slides**

You can knit the source document like other Rmd documents to view the output, but it may be tedious to have to knit it over and over again whenever you make changes. The other way to preview the slides is the RStudio addin “Infinite Moon Reader” or the function xaringan::inf\_mr(), as mentioned in Section [2.2](https://bookdown.org/yihui/rmarkdown/compile.html#compile). With this way, you can continuously preview your slides just by saving the source document. The continuous preview is achieved via a local web server launched by the **servr** package (Xie [2019d](https://bookdown.org/yihui/rmarkdown/xaringan-preview.html#ref-R-servr)).

One distinction of xaringan::moon\_reader when compared to other R Markdown output formats is that it does not generate self-contained HTML documents by default (see Section [3.1.9](https://bookdown.org/yihui/rmarkdown/html-document.html#document-dependencies)). This means none of the external dependencies such as images or JavaScript libraries will be embedded in the HTML output file by default. Due to technical difficulties (remark.js does not use Pandoc but renders Markdown in real time in the browser), it is hard to implement the self-contained mode well. If you have to publish the slides to a web server, but it is not convenient for you to upload all the dependencies, **xaringan** may not be a good choice for you. If you use GitHub Pages or Netlify, this may not be a big problem (you commit or upload all files).

### References

Xie, Yihui. 2019d. Servr: A Simple Http Server to Serve Static Files or Dynamic Documents. [https://CRAN.R-project.org/package=servr](https://cran.r-project.org/package=servr).

## **7.5 CSS and themes**

The format xaringan::moon\_reader has a css option, to which you can pass a vector of CSS file paths, e.g.,

---

output:

xaringan::moon\_reader:

css: **[**"default"**,** "extra.css"**]**

---

In theory, the file path should contain the extension .css. If a path does not contain a filename extension, it is assumed to be a built-in CSS file in the **xaringan** package. For example, default in the above example means default.css in the package under the path xaringan:::pkg\_resource(). To see all built-in CSS files, call xaringan:::list\_css() in R.

When you only want to override a few CSS rules in the default theme, you do not have to copy the whole file default.css; instead, create a new (and hopefully smaller) CSS file that only provides new CSS rules.

Users have contributed a few themes to **xaringan**. For example, you can use the metropolis theme (<https://github.com/pat-s/xaringan-metropolis>):

---

output:

xaringan::moon\_reader:

css: **[**default**,** metropolis**,** metropolis-fonts**]**

---

To see all possible themes:

**names**(xaringan:::**list\_css**())

## [1] "chocolate-fonts" "chocolate"

## [3] "default-fonts" "default"

## [5] "duke-blue" "fc-fonts"

## [7] "fc" "hygge-duke"

## [9] "hygge" "kunoichi"

## [11] "lucy-fonts" "lucy"

## [13] "metropolis-fonts" "metropolis"

## [15] "middlebury-fonts" "middlebury"

## [17] "ninjutsu" "rladies-fonts"

## [19] "rladies" "robot-fonts"

## [21] "robot" "rutgers-fonts"

## [23] "rutgers" "shinobi"

## [25] "tamu-fonts" "tamu"

## [27] "uo-fonts" "uo"

## [29] "uol-fonts" "uol"

If you also want to contribute themes, please read the guide at <https://yihui.name/en/2017/10/xaringan-themes>.

## **7.6 Some tips**

Lastly, we present a few tips that may help you make better presentations.

### 7.6.1 Autoplay slides

Slides can be automatically played if you set the autoplay option under nature (in milliseconds). For example, the next slide can be displayed automatically every 30 seconds in a lightning talk:

---

output:

xaringan::moon\_reader:

nature:

autoplay: 30000

---

### 7.6.2 Countdown timer

A countdown timer can be added to every page of the slides using the countdown option under nature. For example, if you want to spend one minute on every page when you give the talk, you can set:

---

output:

xaringan::moon\_reader:

nature:

countdown: 60000

---

Then you will see a timer counting down from 01:00, to 00:59, 00:58, … When the time is out, the timer will continue but the time turns red.

### 7.6.3 Highlight code lines

The option highlightLines: true of nature will highlight code lines that start with \*, or are wrapped in {{ }}, or have trailing comments #<<:

---

output:

xaringan::moon\_reader:

nature:

highlightLines: true

---

Below are a few examples:

```r

if (TRUE) {

\* message("Very important!")

}

```

```{r tidy=FALSE}

if (TRUE) {

{{ message("Very important!") }}

}

```

```{r tidy=FALSE}

library(ggplot2)

ggplot(mtcars) +

aes(mpg, disp) +

geom\_point() + #<<

geom\_smooth() #<<

```

Note that the first way does not give you valid R code in the source document, but the latter two ways provide syntactically valid R code, and in the output slides, you will not see the tokens {{ }} or #<<. The lines will be highlighted with a yellow background by default.

### 7.6.4 Working offline

To make slides work offline, you need to download a copy of remark.js in advance, because **xaringan** uses the online version by default. You can use xaringan::summon\_remark() to download the latest or a specified version of remark.js. By default, it is downloaded to libs/remark-latest.min.js.

Then change the chakra option in the YAML metadata to point to this file, e.g.,

output:

xaringan::moon\_reader:

chakra: libs/remark-latest.min.js

Making the slides work offline can be tricky, since you may have other dependencies. The remark.js dependency is easy to deal with because it is a single JavaScript file; other dependencies such as MathJax can be extremely tricky. If you used Google web fonts in slides (the default theme uses Yanone Kaffeesatz, Droid Serif, and Source Code Pro), they will not work offline unless you download or install them locally. The Heroku app [google-webfonts-helper](https://google-webfonts-helper.herokuapp.com/fonts) can help you download fonts and generate the necessary CSS.

### 7.6.5 Macros

The Markdown syntax of remark.js can be amazingly extensible, because it allows users to define custom macros (JavaScript functions) that can be applied to Markdown text using the syntax ![:macroName arg1, arg2, ...] or ![:macroName arg1, arg2](this). For example, you can define a macro named scale to set the width of an image:

remark.macros.scale = **function**(w) {

**var** url = **this**;

**return** '<img src="' + url + '" style="width: ' + w + '" />';

};

Then the Markdown text

**![:scale 50%](image.jpg)**

will be translated to:

**<img** src="image.jpg" style="width: 50%" **/>**

Now you should see that you can use cleaner pseudo-Markdown syntax to generate HTML.

To insert macros in **xaringan** slides, you can save your macros in a file (e.g., macros.js), and use the option beforeInit under the option nature, e.g.,

output:

xaringan::moon\_reader:

nature:

beforeInit: "macros.js"

The beforeInit option can be used to insert arbitrary JavaScript code before remark.js initializes the slides. Inserting macros is just one of its possible applications. For example, when you embed tweets from Twitter in slides, usually you need to load https://platform.twitter.com/widgets.js, which can be loaded via the beforeInit option.

### 7.6.6 Disadvantages

The **xaringan** package was originally designed for “ninja”, meaning that if you know CSS, you will be able to freely customize the style, otherwise you can only accept the default themes. Playing with CSS can be fun and rewarding, but it can also easily waste your time. You aesthetic standards and taste may change from time to time, and you could end up tweaking the styles all the time.

The HTML output file generated from **xaringan** is not self-contained by default, as we mentioned in Section [7.4](https://bookdown.org/yihui/rmarkdown/xaringan-preview.html#xaringan-preview). If your slides must be self-contained and cannot be served through a web server, **xaringan** may not be a good option for you.

HTML widgets may not work well in **xaringan**. This might be improved in the future, but it is a little tricky technically.

When printing the slides to PDF from Google Chrome (see Section [4.1.10](https://bookdown.org/yihui/rmarkdown/ioslides-presentation.html#printing-and-pdf-output)), I recommend that you open the slides and go through all pages at least once, to make sure all content has been rendered in the browser. Without navigating through all slides manually once, some content may not be printed correctly (such as MathJax expressions and HTML widgets).