谢益辉

Bookdown: 使用 *R Markdown* 创作书籍和技术文档

To Shao Yong (邵雍),
for sharing a secret joy with simple words;

月到天心处，风来水面时。
一般清意味，料得少人知。

and

To Hongzhi Zhengjue (宏智禅师),
for sharing the peace of an ending life with simple words.

梦幻空华，六十七年；
白鸟淹没，秋水连天。

目录

插图

表格

翻译与排版说明

本书的简体中文版还没有影子，所以笔者先用自己早就遗忘的文学素养进行翻译，使用与原书相同的技术栈生成中文翻译版本。

翻译说明

英译中的技术书籍的翻译是个痛苦的过程，难以避免地会遇到一些没有公认翻译方式的英文词汇，或者是有公认的中文翻译词汇，但该词过于口语化，或者不能很好地反映英文词汇的意思。这些英文词汇以及它们在书中的对应中文词汇将在下方列出，欢迎大家批评。

英文词汇	中文翻译	原因
package	软件包，程序包	r pkgs 是一组用来完成特定任务的程序，作为 R 的补充，符合 Software Package 的定义。
hardcopy	实体书，书的实体版本	原意为“硬拷贝”，指信息被储存并显示在物体实体上，这里采用符合常用语境的翻译。

英文词汇	中文翻译	原因
page margin	页边空白	直译为页面外边距（区域），是页面各边边线离矩形文字区域的垂直距离，四边共同组成了边框形状的区域，通常为空白部分。
typewriter font	老式打字机字体	直译为“打字机字体”，也就是类似于二十世纪七八十年代的铅字打字机的字体，为突出其独特性而强调了“老式”
R plots	（暂无）	（暂无）
personal access token	个人访问令牌	参照国内计网教科书对于 token 的翻译进行的直译。
command-line	命令行（hang，第二声）、命令提示符	笔者对于“line”所指的概念不明确，因此参照国内流行的翻译，称为“命令行”。
key	字段、配置项	指的是 YAML 文件中的配置项，位于冒号：左侧。由于是“键值对”的形式，因此用了 key 一词，但依照语境翻译为“字段”或是“配置项”。
LaTeX preamble	LaTeX preamble、LaTeX 导言	在 `\begin{document}` 之前的命令称为“preamble”（导言），preamble 中通常定义文档的格式、语言等。

英文词汇	中文翻译	原因
Small Capitals	小型大写字母	西文字体设计中的一种字符形式，其大写字母的字高一般与 'x' 等高，并在笔画上做一定的修正，保持更宽的纵横比以保证可读性。
dedication page	献词页	在一些书中，作者想要把这本书献给某人，献词通常写在前几页。
quote	引用环境（文段）	LaTeX 中的 Quote 环境，放置引用于其他文献中的文段。
copyeditor	定稿编辑	文稿最后付印时按照印刷出版要求进行排版、校正文字和格式错误的编辑。
typeface	字型	typeface 与 font 有着微妙的区别，本书中将前者翻译为“字型”，后者翻译为“字体”。且前者多指代印刷用字体。
help desk	帮助中心	直译为“帮助台”，是用来解决用户的 IT 服务问题，降低处置时间的一个服务。
demo	样例	为与 example 区分，demo 翻译为样例，example 翻译为示例。

英文词汇	中文翻译	原因
final words	结语	翻译为中文书籍中常见的“结语”。
index	主页	当描述对象为网页时，翻译为“主页”，指网站的入口点。
in this case	使用这种方法时	将原意“在这种情况下”翻译得更加具体一些。
render	编译、呈现为、转化为	在图形学中一直被翻译为“渲染”，但用在本书中并不合适。考虑到本书中书籍是由源文档转化为多种格式的书籍，其过程涉及源文档的转译（Markdown to LaTeX）与编码，因此翻译为“编译”。另外，它也有“呈现”的意思，在本书中的一些语境下适用。
knit	生成、“编织”	这个词是对于将代码和文字交织在一起的文学编程的形象描述，笔者暂且找不到一个好的词来准确描述该过程，因此使用“生成”或加了双引号的“编织”一词来翻译。
isolate	剔除	作“分离”、“剔除”解释。

英文词汇	中文翻译	原因
side-effects	副作用	这里指程序设计中的“副作用”。如果一个函数修改了自己范围之外的资源，例如读取文件、调用其它有副作用的函数，则该函数称作是有副作用的。
upgrade/update	升级/更新	
serve (v.)	通过 HTTP 服务	直译并不贴切，因此用其行为和原理来解释这个词。
daemonized server	守护进程（?）	中文互联网上少见“守护服务器”这一词汇，因此将“server”作“服务程序”解，翻译为“守护进程”，日后再行修订。

排版说明

由于书中不可避免地会同时出现中文和英文，因此原书的排版并不完全适用于中文翻译版。为了在尊重原书的基础上使页面变得美观，约定如下排版要求：

1. 英文单词、标点符号和数字各具有 1 个前导空格和 1 个后导空格。例如：软件包的名称是 bookdown 吗。

- 英文单词、标点和数字的一侧为标点符号时，该侧无空格。例如：使用 Leading and Trailing Spaces。

2. 需要展示并链接 URL 时，将其放入尖括号内 <>。
3. 小括号内的文本包含中文时，使用中文小括号 （）；如果全是英文文本，则使用英文小括号 ()，并各具有 1 个前导和后导空格。
4. 书中某些操作中带有选项、菜单等名称，在实际操作时不具有中文翻译，此时列出该单词的中文翻译，后跟括号，括号内展示原英文单词。中文翻译便于读者查询相关资料，原英文单词便于按图索骥地进行操作。

翻译进度

常言道，人生未填之坑十之八九。笔者学业繁忙，只能利用空闲时间翻译本书。因此在这里记录一下翻译进度，欢迎加入本项目提交 Pull Request。

章节	是否翻译	是否润色
preface	√	×
Author	√	×
Introduction	√	×
Components	×	×
Output Formats	×	×
Customization	×	×
Editing	√	×
Publishing	√	×

章节	是否翻译	是否润色
Appendix	√	×
References	√	×

前言

这本短小精悍的书籍介绍了一个 R 软件包 **bookdown**，它能够改变你创作书籍的流程。写一本书在技术上要容易，看书时在视觉上要舒适愉悦，与书互动时要有趣，总览全书要方便，读者能够直截了当地为书籍内容做出贡献，或是给作者留下反馈。最重要的是，作者不应该总是被排版细节分散注意力。

Bookdown 是构建在 R Markdown (http://rmarkdown.rstudio.com) 之上的一个拓展包，它继承了 Markdown 语法的简单性（你能够在 5 分钟内学会基础内容；请看第 2.1 节），同时也继承了以多种格式 (PDF/HTML/Word/...) 进行输出的可能性。同时，它添加了多页 HTML 输出、图/表/节/方程的编号与交叉引用、插入多章组成的部分/附录等功能，并导入了 GitBook 样式 (https://www.gitbook.com) 以创建优雅迷人的 HTML 书页。这本书本身就是一个教你如何从一系列 R Markdown 文档中生成一本书籍的例子，并且其印刷版与在线版都能够有专业的观感。你能够在 https://bookdown.org 上找到更多的例子。

尽管名称中包含了 “Book” 一词，但 **Bookdown** 软件包并不仅仅适用于写书。“书” 可以是任何能够按照线性顺序阅读的一系列 R Markdown 文档，例如课程讲义、学习笔记、软件使用手册、论文，甚至可以是日记。事实上，许多 **bookdown** 特性也适用于单个 R Markdown 文档（请见第 3.4 节）。

本书的在线版本依据 Creative Commons Attribution-

NonCommercial-ShareAlike 4.0 International License[1] 许可证进行授权。另外，你能够在 Chapman & Hall[2] 或者亚马逊上购买本书的实体版本。

为什么要阅读这本书

我们能够只使用一种源文档格式编写书籍，但能生成多种格式的输出文档吗？书籍传统上通常是使用 LaTeX 或者 Microsoft Word 进行编写的。但不论是哪种工具都会使得写书变成一趟单程旅行，你无法回头：如果选择 LaTeX，你通常只会得到一个 PDF 文档；如果使用 Word，你可能不得不永远挣扎在 Word 的泥潭中，而且可能会错过许多有用的特性以及来自 LaTeX 的漂亮的 PDF 输出。

我们能够专注于书写内容而不用太担心排版吗？内容和外观之间似乎有着天然的矛盾，我们总是要平衡花费在这两方面上的时间。鱼和熊掌不可兼得，但这并不意味着我们不能吃到半条鱼和半块熊掌。我们希望我们的书看起来美观，我们也希望把注意力集中在内容上。一种选择是暂时放弃 PDF 输出，作为回报，你可能会得到一个相当不错的 HTML 网页预览版。LaTeX 是一个非常好的排版工具，但是在编写书籍的过程中，你很容易沉浸于大量的 LaTeX 命令和排版细节。我们很难避免通过 PDF 预览正在编写的书籍，然而不幸的是，我们经常会发现某些单词超出了页边空白，某些图片浮动到随机的页面上，一章末尾的五到六个零星的单词骄傲地占据了一个全新的页面……如果书籍要印刷，我们最终将不得不处理这些问题，但当你在创作书的内容时，不值得一次又一次为此分心。事实上，Markdown 语法比 LaTeX 更加

[1] http://creativecommons.org/licenses/by-nc-sa/4.0/

[2] https://www.crcpress.com/product/isbn/9781138700109

简单，功能更少，这有助于你专注于书的内容。真的有必要自己定义一个像 `\myprecious{}` 一样的新命令来将 `\textbf{\textit{\textsf{}}}` 应用到文本上吗？当读者能够轻易地理解字母“R”代表 R 语言时，字母“R”是否有必要包括在 `\proglang{}` 中？如果读者需要关注书籍的每一处细节，那这和读者什么都不关注有什么区别呢？因此好的书籍创作技术应该帮助作者自动解决对于内容不重要的细节，让作者关注重点内容。

读者能和我们的书籍中的例子进行互动吗？如果书籍是打印在纸上的，答案当然是不能。但如果你的书籍有 HTML 版本，并包含了在线示例，例如 Shiny 应用 (https://shiny.rstudio.com) 或 HTML 组件 (https://htmlwidgets.org)，那么读者阅读时就可能能够与书进行互动。例如，读者能够立刻知道如果他们改变了统计模型的某些参数后会发生什么。

我们能够在创作书籍时得到来自读者的反馈，甚至是内容贡献吗？传统上，编辑会找一小部分匿名评审员来审查你的书。评审员往往很有帮助，但你仍然可能错过来自更有代表性的读者的智慧。如果读者只有等到第一版印刷发布之后才能够看到你的书，那可能已经太迟了，他们可能需要等待好几年才能看到增订修改后的第二版。有一些网络平台，人们可以轻松地利用它们提供反馈并为你的项目做出贡献。GitHub (https://github.com) 就是一个突出的例子。如果有人在你的书里发现一个拼写错误，他/她能够简单地进行在线更正，并将更改提交给你供你审阅批准。你只需要点击一个按钮来合并更改，无需询问任何问题或来回发送邮件。为了能够使用这些平台，你需要学习 GIT 等版本控制系统，并且你的书籍源文件应该是纯文本。

R (https://www.r-project.org)、Markdown 和 Pandoc (http://pandoc.org) 的组合使得将文档从一种简单的源格式 (R Markdown) 转换为多种格式（PDF、HTML、EPUB 和 Word……）成为可能。**bookdown** 软件包的功能基于 R Markdown 实现，并为书籍和长篇文章提

供输出格式，其中包括 GitBook 格式，它是一种多页面 HTML 输出格式，有着实用且美观的用户界面。用 HTML 进行排版比用 LaTeX 轻松得多，因此你能够经常使用 HTML 预览你的书籍，并且在内容基本完成后再转为 PDF 进行调整。可运行示例很容易就能够插入 HTML 中，它可以使得书籍更具有吸引力和实用性。R Markdown 是一种纯文本格式，因此你也能享受版本控制的优势，例如在 GitHub 上协作创作。我们还努力将一些重要特性从 LaTeX 移植到 HTML 和其它输出格式上，例如图/表编号和交叉引用。

简单来说，你只需要准备一些 R Markdown 格式的书籍章节文档，然后 **bookdown** 就能帮助你将它们转变成一本漂亮的书。

本书的结构

第 1 和 2 章介绍了基础用法和语法，对大多数读者来说应该足够让他们开始创作书籍。第 3 和 4 章是为了那些想要微调书籍外观的读者准备的。如果你不熟悉 HTML/CSS 和 LaTeX，这部分内容可能看起来很技术化。当你第一次阅读本书时，不必非常仔细地阅读这两章。你可以先学习书籍外观的哪些部分可以被改变，之后再回来了解它们是如何被改变的。对于第 5 章，里面的技术细节并不重要，除非你不使用 RStudio IDE（第 5.4 节）。同样地，你可能会对第 6 章中用于发布书籍的命令感到不知所措，但我们仍然可以通过 RStudio IDE 简化你在线发布书籍的流程。自定义命令和函数仅适用于那些选择不使用 RStudio 的服务或想要明白技术细节的读者。

综上所述，本书是对 **bookdown** 程序包的综合参考书。你在阅读

时可以遵循 80/20 法则[3]。有些章节的存在是为了内容的完整性，并不是所有章节都对你想写的书同样有用。

软件信息与一些约定

本书内容主要关于 R 的软件包 **bookdown**，因此你至少需要安装 R 和 **bookdown** 软件包。不过，你的书籍根本不必与 R 语言相关。你可以使用其它计算语言（C++、SQL、Python 等；详情请见附录 B），甚至可以与计算完全无关（例如，你可以创作小说或者是诗集）。附录 A 介绍了创作并构建一本书籍所需的软件工具。

编译本书时的 R Session 信息如下所示：

```
sessionInfo()
```

```
## R version 4.0.3 (2020-10-10)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19042)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=Chinese (Simplified)_China.936
## [2] LC_CTYPE=Chinese (Simplified)_China.936
## [3] LC_MONETARY=Chinese (Simplified)_China.936
```

[3] https://en.wikipedia.org/wiki/Pareto_principle

```
## [4] LC_NUMERIC=C
## [5] LC_TIME=Chinese (Simplified)_China.936
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets
## [6] methods   base
##
## loaded via a namespace (and not attached):
## [1] bookdown_0.22     miniUI_0.1.1.1
## [3] rmarkdown_2.9     tools_4.0.3
## [5] shiny_1.6.0       htmltools_0.5.1.1
## [7] knitr_1.33
```

我们在本书的源代码中没有添加提示符（> 和 +），默认情况下我们使用两个 ## 标签注释掉文本输出，就像你在上面的 R Session 信息中看到的那样。这样做是为了让你能够方便地复制和运行代码（由于文本输出被注释掉了，因此执行代码时会被忽略）。程序包名称以粗体显示（例如，**rmarkdown**），行内代码和文件名用老式打字机字体进行格式化（例如，`knitr::knit('foo.Rmd')`）。函数名称后跟括号（例如，`bookdown::render_book()`）。双冒号操作符 :: 表示从软件包的命名空间对其中的对象进行访问。

致谢

First I'd like to thank my employer, RStudio, for providing me the opportunity to work on this exciting project. I was hoping to work on it when I first saw the GitBook project in 2013, because I immediately

realized it was a beautiful book style and there was a lot more power we could add to it, judging from my experience of writing the **knitr** book (Xie, 2015) and reading other books. R Markdown became mature after two years, and luckily, **bookdown** became my official job in late 2015. There are not many things in the world better than the fact that your job happens to be your hobby (or vice versa). I totally enjoyed messing around with JavaScript libraries, LaTeX packages, and endless regular expressions in R. Honestly I should also thank Stack Overflow (https://stackoverflow.com), and I believe you all know what I mean,[4] if you have ever written any program code.

This project is certainly not a single person's effort. Several colleagues at RStudio have helped me along the way. Hadley Wickham provided a huge amount of feedback during the development of **bookdown**, as he was working on his book *R for Data Science* with Garrett Grolemund. JJ Allaire and Jonathan McPherson provided a lot of technical help directly to this package as well as support in the RStudio IDE. Jeff Allen, Chaita Chaudhari, and the RStudio Connect team have been maintaining the https://bookdown.org website. Robby Shaver designed a nice cover image for this book. Both Hadley Wickham and Mine Cetinkaya-Rundel reviewed the manuscript and gave me a lot of helpful comments. Tareef Kawaf tried his best to help me become a professional software engineer. It is such a blessing to work in this company with enthusiastic and smart people. I remember once I told Jonathan, "hey I found a problem in caching HTML widgets dependencies and finally figured out a possible solution". Jonathan grabbed his beer and said, "I already solved it." "Oh, nice, nice."

I also received a lot of feedback from book authors outside RStudio,

[4] http://bit.ly/2cWbiAp

including Jan de Leeuw, Jenny Bryan, Dean Attali, Rafael Irizarry, Michael Love, Roger Peng, Andrew Clark, and so on. Some users also contributed code to the project and helped revise the book. Here is a list of all contributors: https://github.com/rstudio/bookdown/graphs/contributors. It feels good when you invent a tool and realize you are also the beneficiary of your own tool. As someone who loves the GitHub pull request model, I wished readers did not have to email me there was a typo or obvious mistake in my book, but could just fix it via a pull request. This was made possible in **bookdown**. You can see how many pull requests on typos I have merged: https://github.com/rstudio/bookdown/pulls. It is nice to have so many outsourced careful human spell checkers. It is not that I do not know how to use a real spell checker, but I do not want to do this before the book is finished, and the evil Yihui also wants to leave a few simple tasks to the readers to engage them in improving the book.

Callum Webb kindly designed a nice hexbin sticker for **bookdown**.

The **bookdown** package is not possible without a few open-source software packages. In particular, Pandoc, GitBook, jQuery, and the dependent R packages, not to mention R itself. I thank the developers of these packages.

I moved to Omaha, Nebraska, in 2015, and enjoyed one year at Steeplechase Apartments, where I lived comfortably while developing the **bookdown** package, thanks to the extremely friendly and helpful staff. Then I met a professional and smart realtor, Kevin Schaben, who found a fabulous home for us in an amazingly short period of time, and I finished this book in our new home.

John Kimmel, the editor from Chapman & Hall/CRC, helped me publish my first book. It is my pleasure to work with him again. He gener-

ously agreed to let me keep the online version of this book for free, so I can continue to update it after it is printed and published (i.e., you do not have to wait for years for the second edition to correct mistakes and introduce new features). I wish I could be as open-minded as he is when I'm his age. Rebecca Condit and Suzanne Lassandro proofread the manuscript, and their suggestions were professional and helpful. Shashi Kumar solved some of my technical issues with the publisher's LaTeX class (`krantz.cls`) when I was trying to integrate it with **bookdown**. I also appreciate the very helpful comments from the reviewers Jan de Leeuw, Karl Broman, Brooke Anderson, Michael Grayling, Daniel Kaplan, and Max Kuhn.

Lastly I want to thank my family, in particular, my wife and son, for their support. The one-year-old has discovered that my monitor will light up when he touches my keyboard, so occasionally he just creeps into my office and presses randomly on the keyboard when I'm away. I'm not sure if this counts as his contribution to the book… @)!%)&@*

Yihui Xie
Elkhorn, Nebraska

关于作者

谢益辉 (http://yihui.org) 是 RStudio 的软件工程师 (http://www.rstudio.com)。他在爱荷华州立大学统计系获得了博士学位。他对交互式统计图形和统计计算感兴趣。作为一个活跃的 R 语言用户，他编写了多个 R 包，如 **knitr**、**bookdown**、**blogdown**、**animation**、**DT**、**tinytex**、**tufte**、**formatR**、**fun**、**mime**、**highr**、**servr**、**Rd2roxygen** 等，其中 **animation** 软件包荣获 2009 年的 John M. Chambers Statistical Software Award (ASA)。他还与人合著了其他一些 R 软件包，包括 **shiny**、**rmarkdown** 和 **sliple**。

2006 年，他创立了统计之都 (https://cosx.org)，它已经发展成为中国的一个大型统计网络社区。他于 2008 年发起了中国 R 语言会议，并从那时起参与了在中国组织 R 语言会议。在爱荷华州立大学攻读博士学位期间，他获得了 Vince Sposito Statistical Computing Award (2011) 和统计部的 Snedecor Award (2012)。

他偶尔在推特上咆哮 (https://twitter.com/xieyihui)，大多数时候你都可以在 GitHub 上找到他 (https://github.com/yihui).

他喜欢中国古典文学，同样喜欢辛辣的食物。

1

简介

这本书是使用 R Markdown (Allaire et al., 2021a) 和 R 软件包 **bookdown** (Xie, 2021a) 创作书籍和技术文档的指南。它侧重于创作书籍、长篇文章或报告所需要使用的功能，例如：

- 公式、定理、图表的排版和交叉引用；
- 如何为一本书生成多种输出格式，例如 HTML、PDF 和电子书；
- 怎样自定义书本模板并为书中不同元素设置样式；
- 编辑器支持（尤其是 RStudio IDE）；
- 怎样发布书籍；

这不是对 R Markdown 或 **knitr** 软件包 (Xie, 2021b) 的全面介绍，尽管 **bookdown** 就是在这个软件包的基础上构建的。要了解有关 R Markdown 的更多信息，请查看联机文档 http://rmarkdown.rstudio.com。关于 **knitr**，请参阅 Xie (2015)。你不必是 R 语言 (R Core Team, 2020) 的专家就可以阅读这本书，但是你应该对 R Markdown 和 **knitr** 有一些基本的了解。对于初学者，你可以从 https://www.rstudio.com/resources/cheatsheets/ 上的备忘单开始学习。本书的附录包含对这些软件包的简要介绍。为了能够自定义书籍模板和主题，你应该熟悉 LaTeX、HTML 和 CSS。

1.1 开发动机

Markdown 是一种很好的语言，可以编写相对简单的文档，其中包含诸如节、段落、列表、链接和图像等元素。Pandoc (http://pandoc.org) 极大地扩展了原始 Markdown 语法[1]，并增加了不少有用的新功能，如脚注、引文和表格。更重要的是，Pandoc 可以从 Markdown 生成多种格式的输出文档，包括 HTML、LaTeX/PDF、Word 和幻灯片。

目前 Pandoc 的 Markdown 还缺少一些有用的功能，这些功能对于编写一个相对复杂的文档（比如一本书）是必要的，比如 HTML 输出中的图表自动编号、图表的交叉引用以及对图表外观的精细控制（例如，目前无法使用 Markdown 语法指定图像的对齐方式。这些是我们在 **bookdown** 软件包中解决的一些问题。

在我们想要以多种输出格式构建书籍的限制下，几乎不可能涵盖这些不同输出格式的所有可能的特性。例如，使用 (R) Markdown 语法在 HTML 输出中重新创建某个复杂的 LaTeX 环境可能很困难。我们的主要目标不是用 Markdown 来代替一切，而是涵盖编写一个相对复杂的文档所需的大多数常见功能，并使这些功能的语法在所有输出格式下保持一致。这样你只需要学习一种技术，它就可以用于所有输出格式。

这个项目的另一个目标是使得构建令人赏心悦目的书籍变得更加容易。一些不错的现有例子包括 GitBook (https://www.gitbook.com)、tufte CSS (http://edwardtufte.github.io/tufte-css/) 和 Tufte-LaTeX (https://tufte-latex.github.io/tufte-latex/)。我们希望将这些主题和样式集成到 **bookdown** 中，这样作者就不必深入研究如何使用某个 LaTeX 类或如何为 HTML 输出配置 CSS 等细节。

[1] http://daringfireball.net/projects/markdown/

1.2 开始

对于初学者来说，使用 R Markdown 和 **bookdown** 开始创作书籍的最简单的方法是 GitHub 上的示例 bookdown-demo：

1. 下载 GitHub 存储库 https://github.com/rstudio/bookdown-demo 作为 Zip 文件[2]，然后在本地解压该文件。
2. 安装 RStudio IDE。注意，你需要版本号高于 1.0.0 的 RStudio。如果你的 RStudio 版本低于 1.0.0，请下载最新版本[3]。
3. 安装 R 软件包 **bookdown**：

```
# 安装 CRAN 上的稳定版本
install.packages('bookdown')
# 或者安装 GitHub 上的开发版本
# remotes::install_github('rstudio/bookdown')
```

4. 在 RStudio 中点击 bookdown-demo.Rproj 打开你下载的 bookdown-demo 储存库。
5. 打开 R Markdown 文件 index.Rmd，然后点击 RStudio 里位于 Biold 选项卡中的按钮 Build Book。

[2] https://github.com/rstudio/bookdown-demo/archive/master.zip

[3] https://www.rstudio.com/products/rstudio/download/

如果你打算把你的书打印成 PDF 格式，你将需要一个 LaTeX 发行版。我们建议你安装 TinyTeX（包含 Xeletex）：https://yihui.org/tinytex/。

现在你应该可以在 RStudio viewer 中看到本书样例的索引页。你可以添加或更改 R Markdown 文件，然后再次点击 Knit 按钮预览书籍。如果你不想使用 RStudio，也可以通过命令行编译书籍。详见下一节。

尽管你在 bookdown-demo 示例中看到了不少文件，但大多数文件对于一本书来说并不是必需的。如果你对巨大的文件数量感到不知所措，可以使用这个最小的示例，它实际上是一个文件 index.Rmd：https://github.com/yihui/bookdown-minimal。bookdown-demo 示例包含一些你之后可能需要学习的高级设置，例如如何自定义 LaTeX 序言 (preamble)、调整 CSS 以及在 GitHub 上构建图书等。

1.3 使用方法

有典型的 **bookdown** 书籍包括多个章节，并且每一章放在一个 R Markdown 文件中，文件的拓展名为 .Rmd。每一个 R Markdown 文件必须直接以本章标题作为开头，并使用一级标题，例如 # Chapter Title。全部 R Markdown 文件必须使用 UTF-8 编码，特别是当他们包含某些多字节字符时，例如中文、日文和韩文。以下是一个例子（the bullets are the filenames, followed by the file content):

- index.Rmd

```
# 前言 {-}

在本书中，我们将会介绍一种有趣的方法。
```

- 01-intro.Rmd

```
# 简介

本章是我们提出的用来解决一个 ** 重要问题 ** 的方法的概述。
```

- 02-literature.Rmd

```
# 文献

下面是对现有方法的回顾。
```

- 03-method.Rmd

```
# 方法

我们在本章介绍了我们提出的方法。
```

- 04-application.Rmd

```
# 应用

本章中展示了一些 _ 重要的 _ 应用。

## 示例 1

## 示例 2
```

- 05-summary.Rmd

```
# 结语

我们完成了一本好书。
```

默认情况下，**bookdown** 按文件名的顺序合并所有 Rmd 文件，例如，`01-intro.Rmd` 将出现在 `02-literature.Rmd` 之前。以下划线 `_` 开头的文件名将被跳过。如果存在名为 `index.Rmd` 的 Rmd 文件，则在合并所有 Rmd 文件时，它将始终被视为首个文件。使用这种特殊处理的原因是，从 `index.Rmd` 生成的 HTML 文件 `index.HTML` 通常是你查看网站时的默认主页，例如，当你打开 http://yihui.org/ 时，你实际上正在浏览 http://yihui.org/index.html。

你能够通过在书籍目录中包含一个名为 `_bookdown.yml` 的配置文件来覆盖程序的上述行为。它是一个 YAML 文件 (https://en.wikipedia.org/wiki/YAML)，R Markdown 用户应该对这种格式很熟悉，因为它也被用来在 R Markdown 文档开头编写元数据（你能够在第 B.2 节了解有关 YAML 的更多信息）。你可以使用一个名为 `rmd_files` 的字段来定义你自己的书籍文件列表与 Rmd 文件顺序。例如：

```
rmd_files: ["index.Rmd", "abstract.Rmd", "intro.Rmd"]
```

使用上述方法时，**bookdown** 将会使用你在这个 YAML 字段（如果文件 index.Rmd 存在，它将会被添加进文件列表，并且以下划线命名的文件名将会被忽略）中定义的文件列表。如果你希望同时输出 HTML 和 LaTeX/PDF 文档，并且对于 HTML 和 LaTeX 输出使用不同的 Rmd 文件，你可以分别为这两种输出格式指定不同的文件列表，例如，

```
rmd_files:
  html: ["index.Rmd", "abstract.Rmd", "intro.Rmd"]
  latex: ["abstract.Rmd", "intro.Rmd"]
```

尽管我们一直在谈论 R Markdown 文件，但章节文件实际上不必是 R Markdown 文件。它们可以是普通的 Markdown 文件（.md），并且完全不需要包含 R 代码块。你当然可以使用 **bookdown** 来创作小说和诗歌。

目前，你可能会使用的主要的输出格式包括 bookdown::pdf_book、bookdown::gitbook、bookdown::html_book 和 bookdown::epub_book。软件包中有一个类似于 rmarkdown::render() 的函数 bookdown::render_book()，但它是为了使用输出格式函数将多个 Rmd 文档呈现在一本书中。你可以直接从命令行调用这个函数，或者点击 RStudio IDE 中的相关按钮。下面是一些命令行示例：

```
bookdown::render_book('foo.Rmd', 'bookdown::gitbook')
bookdown::render_book('foo.Rmd', 'bookdown::pdf_book')
```

```
bookdown::render_book('foo.Rmd', bookdown::gitbook(lib_dir = 'libs'))
bookdown::render_book('foo.Rmd', bookdown::pdf_book(keep_tex = TRUE))
```

为了在 RStudio IDE 中使用 `render_book` 和输出格式函数，可以定义一个名为 `site` 的 YAML 字段，其值为 `bookdown::bookdown_site`，[4]并且输出格式函数可以在 `output` 字段中使用，例如：

```
---
site: "bookdown::bookdown_site"
output:
  bookdown::gitbook:
    lib_dir: "book_assets"
  bookdown::pdf_book:
    keep_tex: yes
---
```

然后你可以点击 RStudio 中 `Build` 选项卡下的 `Build Book` 按钮来将 Rmd 文件编译为一本书，或者点击工具栏中的 `Knit` 按钮来预览当前章节。

更多在 `_bookdown.yml` 中的 **bookdown** 设置将会在第 4.4 节中介绍。除了这些配置，你还能够在书籍的第一个 Rmd 文件中的 YAML 元数据里指定一些 Pandoc 相关的配置，例如标题、作者以及书籍付梓日期等。例如：

[4]这个函数会调用 `bookdown::render_book()`。

```
---
title: "Authoring A Book with R Markdown"
author: "Yihui Xie"
date: "`r Sys.Date()`"
site: "bookdown::bookdown_site"
output:
  bookdown::gitbook: default
documentclass: book
bibliography: ["book.bib", "packages.bib"]
biblio-style: apalike
link-citations: yes
---
```

1.4 两种呈现方法

将所有章节合并到一个 Rmd 文件中，这是在 **bookdown** 中呈现书籍的一种方法。实际上还有另一种方法：你可以在一个单独的会话中生成每一章，**bookdown** 将合并所有章节的 Markdown 输出文档来呈现书籍。我们将这两种方法分别称为“合并与生成”(M-K) 以及“生成与合并”(K-M)。它们之间的差异可能看起来很微妙，但根据你的用例不同可能会变得相当重要。

- 二者最显著的差异是：M-K 在相同的 R session 中运行所有代码块，而 K-M 对于每一个独立的章节都会使用单独的 R session。对于 M-K 来说，来自前几章的 R session 状态将会转移到之后的章节（例如，前几章中创建的对象可用于后几章，除非你故意删除了它们）；对于

K-M 来说，所有的章节都是相互隔离的。[5]如果你希望每一章都在一个干净的状态下进行编译，那么就使用 K-M 方法。如果你使用 M-K 方法，那么将一个正在运行中的 R session 恢复到完全干净的状态是非常棘手和困难的。例如，即便你 detach/unload 上一章中加载的软件包，R 也不会清除由这些软件包注册的 S3 方法。

- 因为 **knitr** 不允许在一个源文档中出现重复的代码块标签，因此当你使用 M-K 方法时，需要确保在书籍各章节中没有重复的标签，否则 **knitr** 在生成合并后的 Rmd 文件时会发出错误信号。而 K-M 方法只需要在任何单个 Rmd 文件中没有重复的标签。
- K-M 方法不允许 Rmd 文件位于子目录中，而 M-K 方法允许。

bookdown 中的默认方法时 M-K。如果想要转为 K-M 方法，可以在调用 `render_book()` 时使用参数 `new_session = TRUE`，或者在配置文件 `_bookdown.yml` 中设置 `new_session: yes`。

对于 K-M 方法，你可以在 `_bookdown.yml` 中配置 `book_filename` 选项，但是它应该是一个 Markdown 文件的名称，例如 `_main.md`。不过文件扩展名并不重要，你甚至可以省略扩展名，例如，只需设置为 `book_filename: _main` 即可。其它配置都适用于 M-K 和 K-M。

1.5 一些提示

分页限制下的排版（例如对于 LaTeX/PDF 输出）可能是一项非常繁琐和耗时的工作。我建议你不要经常查看 PDF 输出，因为大多数情况下你不太可能满意：文本可能超出页边空白，图片可能浮动得太

[5]当然，没有人能阻止你在一个章节中写出一些文件，然后在另一个章节中呈现它们。剔除这些副作用是很困难的。

远等等。不要试图立即使事情看起来很好，因为当你不断修改书籍时，你可能会一次又一次地失望。即使你只是做了一些小的改动，事情也可能会再次变得一团糟（参见 http://bit.ly/tbrLtx，这是一个很好的例子）。

如果想要预览书籍，请预览 HTML 输出。在完成了书籍的内容，并且非常确定不需要进行重大修订后再使用 PDF 版本。

如果 R Markdown 文档中的某些代码块运行起来很费时，你可以通过在块头部添加块选项 cache=TRUE 来缓存这一个代码块的输出，并且也建议你标记这些代码块，例如：

```
```{r important-computing, cache=TRUE}
```

在第 5 章，我们将会讨论如何在你编辑时快速地预览书籍。简单来说，你可以使用 preview_chapter() 函数来编译单个章节，而不是编译整本书。函数 serve_book() 能够让你轻松实现实时预览 HTML 书页：每当你修改 Rmd 文件时，书籍都可以重新编译，浏览器也能相应地自动刷新。
```

2

Components

This chapter demonstrates the syntax of common components of a book written in **bookdown**, including code chunks, figures, tables, citations, math theorems, and equations. The approach is based on Pandoc, so we start with the syntax of Pandoc's flavor of Markdown.

2.1 Markdown syntax

In this section, we give a very brief introduction to Pandoc's Markdown. Readers who are familiar with Markdown can skip this section. The comprehensive syntax of Pandoc's Markdown can be found on the Pandoc website `http://pandoc.org`.

2.1.1 Inline formatting

You can make text *italic* by surrounding it with underscores or asterisks, e.g., `_text_` or `*text*`. For **bold** text, use two underscores (`__text__`) or asterisks (`**text**`). Text surrounded by `~` will be converted to a subscript (e.g., `H~2~SO~4~` renders H_2SO_4), and similarly, two carets (`^`) produce a superscript (e.g., `Fe^2+^` renders Fe^{2+}). To mark text as `inline code`, use a pair of backticks,

e.g., `` `code` ``.[1] Small caps can be produced by the HTML tag `span`, e.g., `<span style="font-variant:small-caps;">Small Caps</span>` renders Small Caps. Links are created using `[text](link)`, e.g., `[RStudio](https://www.rstudio.com)`, and 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.]`. We will talk about citations in Section 2.8.

2.1.2 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 `{-}` 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 by four spaces, e.g.,

[1] To include literal backticks, use more backticks outside, e.g., you can use two backticks to preserve one backtick inside: `` `` `code` `` ``.

```
- one item
- one item
- one item
    - one item
    - one item
```

The output is:

- one item
- one item
- one item
    - one item
    - one item

Ordered list items start with numbers (the rule for nested lists is the same as above), e.g.,

```
1. the first item
2. the second item
3. the third 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

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
```

2.1.3 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: $f(k) = \binom{n}{k} p^k (1-p)^{n-k}$); 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:

$$f\left(k\right) = \binom{n}{k} p^k \left(1-p\right)^{n-k}$$

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

```
$$\begin{array}{ccc}
x_{11} & x_{12} & x_{13}\\
x_{21} & x_{22} & x_{23}
\end{array}$$
```

$$\begin{array}{ccc}
x_{11} & x_{12} & x_{13}\\
x_{21} & x_{22} & x_{23}
\end{array}$$

```
$$X = \begin{bmatrix}1 & x_{1}\\
1 & x_{2}\\
1 & x_{3}
\end{bmatrix}$$
```

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \end{bmatrix}$$

```
$$\Theta = \begin{pmatrix}\alpha & \beta\\
\gamma & \delta
\end{pmatrix}$$
```

$$\Theta = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$$

```
$$\begin{vmatrix}a & b\\
c & d
\end{vmatrix}=ad-bc$$
```

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

2.2 Markdown extensions by bookdown

Although Pandoc's Markdown is much richer than the original Markdown syntax, it still lacks a number of things that we may need for academic writing. For example, it supports math equations, but you cannot number and reference equations in multi-page HTML or EPUB output. We have provided a few Markdown extensions in **bookdown** to fill the gaps.

2.2.1 Number and reference equations

To number and refer to equations, put them in the equation environments and assign labels to them using the syntax `(\#eq:label)`, e.g.,

```
\begin{equation}
  f\left(k\right) = \binom{n}{k} p^k\left(1-p\right)^{n-k}
  (\#eq:binom)
\end{equation}
```

It renders the equation below:

$$f\left(k\right) = \binom{n}{k} p^k \left(1-p\right)^{n-k} \tag{2.1}$$

You may refer to it using `\@ref(eq:binom)`, e.g., see Equation (2.1).

Equation labels must start with the prefix `eq:` in **bookdown**. All labels in **bookdown** must only contain alphanumeric characters, `:`, `-`, and/or `/`. Equation references work best for LaTeX/PDF output, and they are not well supported in Word output or e-books. For HTML output, **bookdown** can only number the equations with labels. Please make sure equations without labels are not numbered by either using the `equation*` environment or adding `\nonumber` or `\notag` to your equations. The same rules apply to other math environments, such as `eqnarray`, `gather`, `align`, and so on (e.g., you can use the `align*` environment).

We demonstrate a few more math equation environments below. Here is an unnumbered equation using the `equation*` environment:

```
\begin{equation*}
\frac{d}{dx}\left( \int_{a}^{x} f(u)\,du\right)=f(x)
\end{equation*}
```

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

Below is an `align` environment (2.2):

```
\begin{align}
g(X_{n}) &= g(\theta)+g'({\tilde{\theta}})(X_{n}-\theta) \notag \\
\sqrt{n}[g(X_{n})-g(\theta)] &= g'\left({\tilde{\theta}}\right)
  \sqrt{n}[X_{n}-\theta ] (\#eq:align)
\end{align}
```

$$\begin{aligned} g(X_n) &= g(\theta) + g'(\tilde{\theta})(X_n - \theta) \\ \sqrt{n}[g(X_n) - g(\theta)] &= g'\left(\tilde{\theta}\right)\sqrt{n}[X_n - \theta] \end{aligned} \tag{2.2}$$

You can use the `split` environment inside `equation` so that all lines share the same number (2.3). By default, each line in the `align` environment will be assigned an equation number. We suppressed the number of the first line in the previous example using `\notag`. In this example, the whole `split` environment was assigned a single number.

```
\begin{equation}
\begin{split}
```

```
\mathrm{Var}(\hat{\beta}) & =\mathrm{Var}((X'X)^{-1}X'y)\\
 & =(X'X)^{-1}X'\mathrm{Var}(y)((X'X)^{-1}X')'\\
 & =(X'X)^{-1}X'\mathrm{Var}(y)X(X'X)^{-1}\\
 & =(X'X)^{-1}X'\sigma^{2}IX(X'X)^{-1}\\
 & =(X'X)^{-1}\sigma^{2}
\end{split}
(\#eq:var-beta)
\end{equation}
```

$$
\begin{aligned}
\mathrm{Var}(\hat{\beta}) &= \mathrm{Var}((X'X)^{-1}X'y) \\
&= (X'X)^{-1}X'\mathrm{Var}(y)((X'X)^{-1}X')' \\
&= (X'X)^{-1}X'\mathrm{Var}(y)X(X'X)^{-1} \\
&= (X'X)^{-1}X'\sigma^2 IX(X'X)^{-1} \\
&= (X'X)^{-1}\sigma^2
\end{aligned} \tag{2.3}
$$

2.2.2 Theorems and proofs

Theorems and proofs are commonly used in articles and books in mathematics. However, please do not be misled by the names: a "theorem" is just a numbered/labeled environment, and it does not have to be a mathematical theorem (e.g., it can be an example irrelevant to mathematics). Similarly, a "proof" is an unnumbered environment. In this section, we always use the *general* meanings of a "theorem" and "proof" unless explicitly stated.

In **bookdown**, the types of theorem environments supported are in Table 2.1. To write a theorem, you can use the syntax below:

表 **2.1:** Theorem environments in **bookdown**.

Environment	Printed Name	Label Prefix
theorem	Theorem	thm
lemma	Lemma	lem
corollary	Corollary	cor
proposition	Proposition	prp
conjecture	Conjecture	cnj
definition	Definition	def
example	Example	exm
exercise	Exercise	exr
hypothesis	Hypothesis	hyp

```
::: {.theorem}
This is a `theorem` environment that can contain **any**
_Markdown_ syntax.
:::
```

This syntax is based on Pandoc's fenced `Div` blocks[2] and can already be used in any R Markdown document to write custom blocks.[3] **Bookdown** only offers special handling for theorem and proof environments. Since this uses the syntax of Pandoc's Markdown, you can write any valid Markdown text inside the block.

To write other theorem environments, replace `::: {.theorem}` with other environment names in Table 2.1, e.g., `::: {.lemma}`.

[2] https://pandoc.org/MANUAL.html#divs-and-spans

[3] https://bookdown.org/yihui/rmarkdown-cookbook/custom-blocks.html

A theorem can have a `name` attribute so its name will be printed. For example,

```
::: {.theorem name="Pythagorean theorem"}
For a right triangle, if $c$ denotes the length of the hypotenuse
and $a$ and $b$ denote the lengths of the other two sides, we have
$$a^2 + b^2 = c^2$$
:::
```

If you want to refer to a theorem, you should label it. The label can be provided as an ID to the block of the form `#label`. For example,

```
::: {.theorem #foo}
A labeled theorem here.
:::
```

After you label a theorem, you can refer to it using the syntax `\@ref(prefix:label)`. See the column `Label Prefix` in Table 2.1 for the value of `prefix` for each environment. For example, we have a labeled and named theorem below, and `\@ref(thm:pyth)` gives us its theorem number 2.1:

```
::: {.theorem #pyth name="Pythagorean theorem"}
For a right triangle, if $c$ denotes the length of the hypotenuse
and $a$ and $b$ denote the lengths of the other two sides, we have

$$a^2 + b^2 = c^2$$
:::
```

Theorem 2.1 (Pythagorean theorem). *For a right triangle, if c denotes the length of the hypotenuse and a and b denote the lengths of the other two sides, we have*

$$a^2 + b^2 = c^2$$

The proof environments currently supported are `proof`, `remark`, and `solution`. The syntax is similar to theorem environments, and proof environments can also be named using the `name` attribute. The only difference is that since they are unnumbered, you cannot reference them, even if you provide an ID to a proof environment.

We have tried to make all these theorem and proof environments work out of the box, no matter if your output is PDF or HTML. If you are a LaTeX or HTML expert, you may want to customize the style of these environments anyway (see Chapter 4). Customization in HTML is easy with CSS, and each environment is enclosed in `<div></div>` with the CSS class being the environment name, e.g., `<div class="lemma"></div>`. For LaTeX output, we have predefined the style to be `definition` for environments `definition`, `example`, `exercise`, and `hypothesis`, and `remark` for environments `proof` and `remark`. All other environments use the `plain` style. The style definition is done through the `\theoremstyle{}` command of the **amsthm** package. If you do not want the default theorem definitions to be automatically added by **bookdown**, you can set `options(bookdown.theorem.preamble = FALSE)`. This can be useful, for example, to avoid conflicts in single documents (Section 3.4) using the output format `bookdown::pdf_book` with a `base_format` that has already included **amsmath** definitions.

Theorems are numbered by chapters by default. If there are no chapters in your document, they are numbered by sections instead. If

the whole document is unnumbered (the output format option `number_sections = FALSE`), all theorems are numbered sequentially from 1, 2, ..., N. LaTeX supports numbering one theorem environment after another, e.g., let theorems and lemmas share the same counter. This is not supported for HTML/EPUB output in **bookdown**. You can change the numbering scheme in the LaTeX preamble by defining your own theorem environments, e.g.,

```
\newtheorem{theorem}{Theorem}
\newtheorem{lemma}[theorem]{Lemma}
```

When **bookdown** detects `\newtheorem{theorem}` in your LaTeX preamble, it will not write out its default theorem definitions, which means you have to define all theorem environments by yourself. For the sake of simplicity and consistency, we do not recommend that you do this. It can be confusing when your Theorem 18 in PDF becomes Theorem 2.4 in HTML.

Below we show more examples[4] of the theorem and proof environments, so you can see the default styles in **bookdown**.

Definition 2.1. The characteristic function of a random variable X is defined by

$$\varphi_X(t) = \mathrm{E}\left[e^{itX}\right], \ t \in \mathcal{R}$$

Example 2.1. We derive the characteristic function of $X \sim U(0, 1)$ with the probability density function $f(x) = \mathbf{1}_{x\in[0,1]}$.

[4] Some examples are adapted from the Wikipedia page https://en.wikipedia.org/wiki/Characteristic_function_(probability_theory)

$$
\begin{aligned}
\varphi_X(t) &= \mathrm{E}\left[e^{itX}\right] \\
&= \int e^{itx} f(x) dx \\
&= \int_0^1 e^{itx} dx \\
&= \int_0^1 \left(\cos(tx) + i\sin(tx)\right) dx \\
&= \left(\frac{\sin(tx)}{t} - i\frac{\cos(tx)}{t}\right)\Bigg|_0^1 \\
&= \frac{\sin(t)}{t} - i\left(\frac{\cos(t)-1}{t}\right) \\
&= \frac{i\sin(t)}{it} + \frac{\cos(t)-1}{it} \\
&= \frac{e^{it}-1}{it}
\end{aligned}
$$

Note that we used the fact $e^{ix} = \cos(x) + i\sin(x)$ twice.

Lemma 2.1. *For any two random variables X_1, X_2, they both have the same probability distribution if and only if*

$$\varphi_{X_1}(t) = \varphi_{X_2}(t)$$

Theorem 2.2. *If $X_1, ..., X_n$ are independent random variables, and $a_1, ..., a_n$ are some constants, then the characteristic function of the linear combination $S_n = \sum_{i=1}^n a_i X_i$ is*

$$\varphi_{S_n}(t) = \prod_{i=1}^n \varphi_{X_i}(a_i t) = \varphi_{X_1}(a_1 t) \cdots \varphi_{X_n}(a_n t)$$

Proposition 2.1. *The distribution of the sum of independent Poisson random variables $X_i \sim \mathrm{Pois}(\lambda_i),\ i = 1, 2, \cdots, n$ is $\mathrm{Pois}(\sum_{i=1}^n \lambda_i)$.*

证明. The characteristic function of $X \sim \text{Pois}(\lambda)$ is $\varphi_X(t) = e^{\lambda(e^{it}-1)}$. Let $P_n = \sum_{i=1}^n X_i$. We know from Theorem 2.2 that

$$\begin{aligned}\varphi_{P_n}(t) &= \prod_{i=1}^n \varphi_{X_i}(t) \\ &= \prod_{i=1}^n e^{\lambda_i(e^{it}-1)} \\ &= e^{\sum_{i=1}^n \lambda_i(e^{it}-1)}\end{aligned}$$

This is the characteristic function of a Poisson random variable with the parameter $\lambda = \sum_{i=1}^n \lambda_i$. From Lemma 2.1, we know the distribution of P_n is $\text{Pois}(\sum_{i=1}^n \lambda_i)$.

□

Remark. In some cases, it is very convenient and easy to figure out the distribution of the sum of independent random variables using characteristic functions.

Corollary 2.1. *The characteristic function of the sum of two independent random variables X_1 and X_2 is the product of characteristic functions of X_1 and X_2, i.e.,*

$$\varphi_{X_1+X_2}(t) = \varphi_{X_1}(t)\varphi_{X_2}(t)$$

Exercise 2.1 (Characteristic Function of the Sample Mean)**.** Let $\bar{X} = \sum_{i=1}^n \frac{1}{n} X_i$ be the sample mean of n independent and identically distributed random variables, each with characteristic function φ_X. Compute the characteristic function of $\bar{X}$.

Solution. Applying Theorem 2.2, we have

$$\varphi_{\bar{X}}(t) = \prod_{i=1}^n \varphi_{X_i}\left(\frac{t}{n}\right) = \left[\varphi_X\left(\frac{t}{n}\right)\right]^n.$$

Hypothesis 2.1 (Riemann hypothesis)**.** The Riemann Zeta-function is defined as

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$$

for complex values of s and which converges when the real part of s is greater than 1. The Riemann hypothesis is that the Riemann zeta function has its zeros only at the negative even integers and complex numbers with real part $1/2$.

2.2.2.1 A note on the old syntax

For older versions of **bookdown** (before v0.21), a `theorem` environment could be written like this:

````
```{theorem pyth, name="Pythagorean theorem"}
For a right triangle, if $c$ denotes the length of the hypotenuse
and $a$ and $b$ denote the lengths of the other two sides, we have

$$a^2 + b^2 = c^2$$
```
````

This syntax still works, but we do not recommend it since the new syntax allows you to write richer content and has a cleaner implementation. However, note that the old syntax has to be used if you want the environment to work with output formats in addition to HTML and PDF, such as EPUB. The fenced `Div` syntax only works for HTML and PDF output at the moment, and we will try to improve it in the future.

This conversion between the two syntaxes is straightforward. The above theorem could be rewritten in this way:

```
::: {.theorem #pyth name="Pythagorean theorem"}
For a right triangle, if $c$ denotes the length of the hypotenuse
and $a$ and $b$ denote the lengths of the other two sides, we have

$$a^2 + b^2 = c^2$$
:::
```

You can use the helper function `bookdown::fence_theorems()` to convert a whole file or a piece of text. This is a one-time operation. We have tried to do the conversion from old to new syntax safely, but we might have missed some edge cases. To make sure you do not overwrite the `input` file by accident, you can write the converted source to a new file, e.g.,

```
bookdown::fence_theorems("01-intro.Rmd", output = "01-intro-new.Rmd")
```

Then double check the content of `01-intro-new.Rmd`. Using `output = NULL` will print the result of conversion in the R console, and is another way to check the conversion. If you are using a control version tool, you can set `output` to be the same as `input`, as it should be safe and easy for you to revert the change if anything goes wrong.

2.2.3 Special headers

There are a few special types of first-level headers that will be processed differently in **bookdown**. The first type is an unnumbered header that starts with the token `(PART)`. This kind of headers are translated to part titles. If you are familiar with LaTeX, this basically means

`\part{}`. When your book has a large number of chapters, you may want to organize them into parts, e.g.,

```
# (PART) Part I {-}

# Chapter One

# Chapter Two

# (PART) Part II {-}

# Chapter Three
```

A part title should be written right before the first chapter title in this part, both title in the same document. You can use `(PART\*)` (the backslash before `*` is required) instead of `(PART)` if a part title should not be numbered.

The second type is an unnumbered header that starts with `(APPENDIX)`, indicating that all chapters after this header are appendices, e.g.,

```
# Chapter One

# Chapter Two

# (APPENDIX) Appendix {-}

# Appendix A

# Appendix B
```

The numbering style of appendices will be automatically changed in LaTeX/PDF and HTML output (usually in the form A, A.1, A.2, B, B.1, ...). This feature is not available to e-books or Word output.

2.2.4 Text references

You can assign some text to a label and reference the text using the label elsewhere in your document. This can be particularly useful for long figure/table captions (Section 2.4 and 2.5), in which case you normally will have to write the whole character string in the chunk header (e.g., `fig.cap = "A long long figure caption."`) or your R code (e.g., `kable(caption = "A long long table caption.")`). It is also useful when these captions contain special HTML or LaTeX characters, e.g., if the figure caption contains an underscore, it works in the HTML output but may not work in LaTeX output because the underscore must be escaped in LaTeX.

The syntax for a text reference is `(ref:label) text`, where `label` is a unique label[5] throughout the document for `text`. It must be in a separate paragraph with empty lines above and below it. The paragraph must not be wrapped into multiple lines, and should not end with a white space. For example,

```
(ref:foo) Define a text reference **here**.
```

Then you can use `(ref:foo)` in your figure/table captions. The text can contain anything that Markdown supports, as long as it is one single paragraph. Here is a complete example:

[5] You may consider using the code chunk labels.

````
A normal paragraph.

(ref:foo) A scatterplot of the data `cars` using **base** R graphics.

```{r foo, fig.cap='(ref:foo)'}
plot(cars)  # a scatterplot
```
````

Text references can be used anywhere in the document (not limited to figure captions). It can also be useful if you want to reuse a fragment of text in multiple places.

2.3 R code

There are two types of R code in R Markdown/**knitr** documents: R code chunks, and inline R code. The syntax for the latter is `` `r R_CODE` ``, and it can be embedded inline with other document elements. R code chunks look like plain code blocks, but have `{r}` after the three backticks and (optionally) chunk options inside `{}`, e.g.,

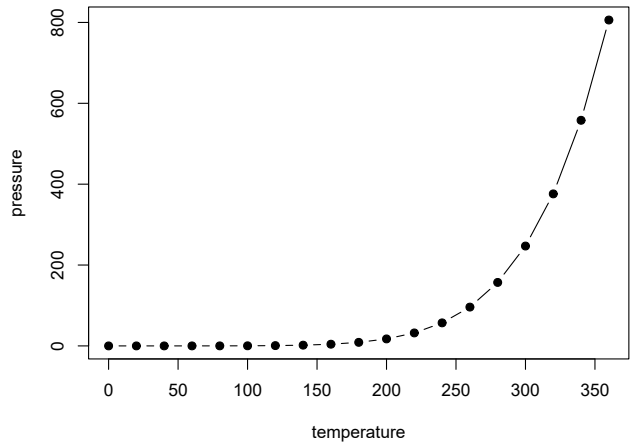
````
```{r chunk-label, echo = FALSE, fig.cap = 'A figure caption.'}
1 + 1
rnorm(10)  # 10 random numbers
plot(dist ~ speed, cars)  # a scatterplot
```
````

To learn more about **knitr** chunk options, see Xie (2015) or the web page `http://yihui.org/knitr/options`. For books, additional R code can be executed before/after each chapter; see `before_chapter_script` and `after_chapter_script` in Section 4.4.

2.4 Figures

By default, figures have no captions in the output generated by **knitr**, which means they will be placed wherever they were generated in the R code. Below is such an example.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, pch = 19, type = 'b')
```



The disadvantage of typesetting figures in this way is that when there is not enough space on the current page to place a figure, it may either reach the bottom of the page (hence exceeds the page margin), or

be pushed to the next page, leaving a large white margin at the bottom of the current page. That is basically why there are "floating environments" in LaTeX: elements that cannot be split over multiple pages (like figures) are put in floating environments, so they can float to a page that has enough space to hold them. There is also a disadvantage of floating things forward or backward, though. That is, readers may have to jump to a different page to find the figure mentioned on the current page. This is simply a natural consequence of having to typeset things on multiple pages of fixed sizes. This issue does not exist in HTML, however, since everything can be placed continuously on one single page (presumably with infinite height), and there is no need to split anything across multiple pages of the same page size.

If we assign a figure caption to a code chunk via the chunk option `fig.cap`, R plots will be put into figure environments, which will be automatically labeled and numbered, and can also be cross-referenced. The label of a figure environment is generated from the label of the code chunk, e.g., if the chunk label is `foo`, the figure label will be `fig:foo` (the prefix `fig:` is added before `foo`). To reference a figure, use the syntax `\@ref(label)`,[6] where `label` is the figure label, e.g., `fig:foo`.

To take advantage of Markdown formatting *within* the figure caption, you will need to use text references (see Section 2.2.4). For example, a figure caption that contains `_italic text_` will not work when the output format is LaTeX/PDF, since the underscore is a special character in LaTeX, but if you use text references, `_italic text_` will be translated to LaTeX code when the output is LaTeX.

[6] Do not forget the leading backslash! And also note the parentheses `()` after `ref`; they are not curly braces `{}`.

If you want to cross-reference figures or tables generated from a code chunk, please make sure the chunk label only contains *alphanumeric* characters (a-z, A-Z, 0-9), slashes (/), or dashes (-).

The chunk option `fig.asp` can be used to set the aspect ratio of plots, i.e., the ratio of figure height/width. If the figure width is 6 inches (`fig.width = 6`) and `fig.asp = 0.7`, the figure height will be automatically calculated from `fig.width * fig.asp = 6 * 0.7 = 4.2`. Figure 2.1 is an example using the chunk options `fig.asp = 0.7`, `fig.width = 6`, and `fig.align = 'center'`, generated from the code below:

```
par(mar = c(4, 4, .1, .1))
plot(pressure, pch = 19, type = 'b')
```

The actual size of a plot is determined by the chunk options `fig.width` and `fig.height` (the size of the plot generated from a graphical device), and we can specify the output size of plots via the chunk options `out.width` and `out.height`. The possible value of these two options depends on the output format of the document. For example, `out.width = '30%'` is a valid value for HTML output, but not for LaTeX/PDF output. However, **knitr** will automatically convert a percentage value for `out.width` of the form `x%` to `(x / 100) \linewidth`, e.g., `out.width = '70%'` will be treated as `.7\linewidth` when the output format is LaTeX. This makes it possible to specify a relative width of a plot in a consistent manner. Figure 2.2 is an example of `out.width = 70%`.

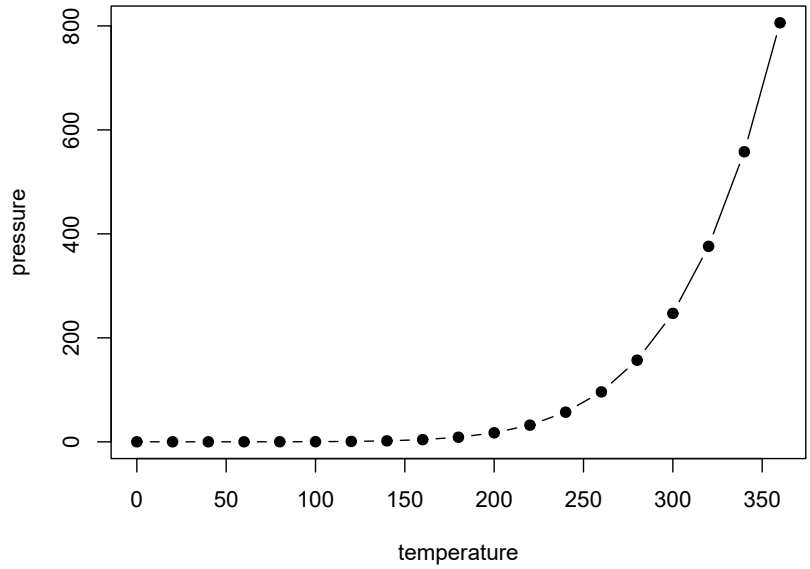


图 **2.1:** A figure example with the specified aspect ratio, width, and alignment.

```
par(mar = c(4, 4, .1, .1))
plot(cars, pch = 19)
```

If you want to put multiple plots in one figure environment, you must use the chunk option `fig.show = 'hold'` to hold multiple plots from a code chunk and include them in one environment. You can also place plots side by side if the sum of the width of all plots is smaller than or equal to the current line width. For example, if two plots have the same width `50%`, they will be placed side by side. Similarly, you can specify `out.width = '33%'` to arrange three plots on one line. Figure 2.3 is an example of two plots, each with a width of `50%`.

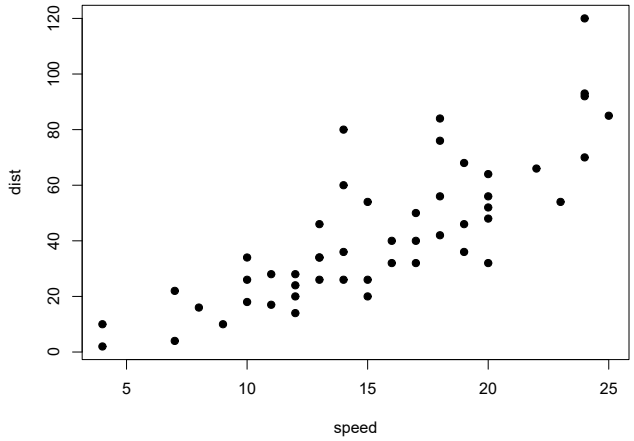


图 **2.2:** A figure example with a relative width 70%.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, pch = 19, type = 'b')
plot(cars, pch = 19)
```

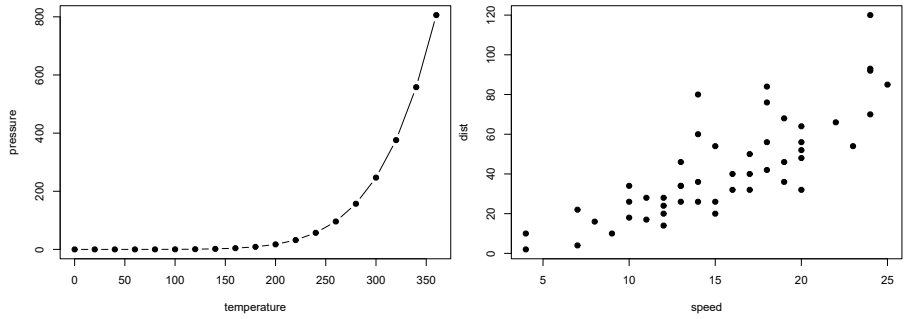


图 **2.3:** Two plots placed side by side.

Sometimes you may have certain images that are not generated from R code, and you can include them in R Markdown via the function `knitr::include_graphics()`. Figure 2.4 is an example of three **knitr** logos included in a figure environment. You may pass one or multiple

image paths to the `include_graphics()` function, and all chunk options that apply to normal R plots also apply to these images, e.g., you can use `out.width = '33%'` to set the widths of these images in the output document.

```
knitr::include_graphics(rep('images/knit-logo.png', 3))
```

图 **2.4:** Three knitr logos included in the document from an external PNG image file.

There are a few advantages of using `include_graphics()`:

1. You do not need to worry about the document output format, e.g., when the output format is LaTeX, you may have to use the LaTeX command `\includegraphics{}` to include an image, and when the output format is Markdown, you have to use `![]()`. The function `include_graphics()` in **knitr** takes care of these details automatically.
2. The syntax for controlling the image attributes is the same as when images are generated from R code, e.g., chunk options `fig.cap`, `out.width`, and `fig.show` still have the same meanings.
3. `include_graphics()` can be smart enough to use PDF graphics automatically when the output format is LaTeX and the

PDF graphics files exist, e.g., an image path `foo/bar.png` can be automatically replaced with `foo/bar.pdf` if the latter exists. PDF images often have better qualities than raster images in LaTeX/PDF output. To make use of this feature, set the argument `auto_pdf = TRUE`, or set the global option `options(knitr.graphics.auto_pdf = TRUE)` to enable this feature globally in an R session.

4. You can easily scale these images proportionally using the same ratio. This can be done via the `dpi` argument (dots per inch), which takes the value from the chunk option `dpi` by default. If it is a numeric value and the chunk option `out.width` is not set, the output width of an image will be its actual width (in pixels) divided by `dpi`, and the unit will be inches. For example, for an image with the size 672 x 480, its output width will be 7 inches (`7in`) when `dpi = 96`. This feature requires the package **png** and/or **jpeg** to be installed. You can always override the automatic calculation of width in inches by providing a non-NULL value to the chunk option `out.width`, or use `include_graphics(dpi = NA)`.

2.5 Tables

For now, the most convenient way to generate a table is the function `knitr::kable()`, because there are some internal tricks in **knitr** to make it work with **bookdown** and users do not have to know anything about these implementation details. We will explain how to use other packages and functions later in this section.

表 **2.2:** A table of the first 10 rows of the mtcars data.

	mpg	cyl	disp	hp	drat	wt	qsec	vs
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1

Like figures, tables with captions will also be numbered and can be referenced. The `kable()` function will automatically generate a label for a table environment, which is the prefix `tab:` plus the chunk label. For example, the table label for a code chunk with the label `foo` will be `tab:foo`, and we can still use the syntax `\@ref(label)` to reference the table. Table 2.2 is a simple example.

```
knitr::kable(
  head(mtcars[, 1:8], 10), booktabs = TRUE,
  caption = 'A table of the first 10 rows of the mtcars data.'
)
```

If you want to put multiple tables in a single table environment, wrap the data objects (usually data frames in R) into a list. See Table 2.3 for

表 **2.3:** A Tale of Two Tables.

Sepal.Length	Sepal.Width
5.1	3.5
4.9	3.0
4.7	3.2

	mpg	cyl	disp
Mazda RX4	21.0	6	160
Mazda RX4 Wag	21.0	6	160
Datsun 710	22.8	4	108
Hornet 4 Drive	21.4	6	258
Hornet Sportabout	18.7	8	360

an example. Please note that this feature is only available in HTML and PDF output.

```
knitr::kable(
  list(
    head(iris[, 1:2], 3),
    head(mtcars[, 1:3], 5)
  ),
  caption = 'A Tale of Two Tables.', booktabs = TRUE
)
```

When you do not want a table to float in PDF, you may use the LaTeX package **longtable**,[7] which can break a table across multiple pages. To use **longtable**, pass `longtable = TRUE` to `kable()`, and make sure to include `\usepackage{longtable}` in the LaTeX preamble (see Section 4.1 for how to customize the LaTeX preamble). Of course, this is irrelevant to HTML output, since tables in HTML do not need to float.

[7] https://www.ctan.org/pkg/longtable

```
knitr::kable(
  iris[1:55, ], longtable = TRUE, booktabs = TRUE,
  caption = 'A table generated by the longtable package.'
)
```

表 **2.4:** A table generated by the longtable package.

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa

5.1	3.8	1.5	0.3	setosa
5.4	3.4	1.7	0.2	setosa
5.1	3.7	1.5	0.4	setosa
4.6	3.6	1.0	0.2	setosa
5.1	3.3	1.7	0.5	setosa
4.8	3.4	1.9	0.2	setosa
5.0	3.0	1.6	0.2	setosa
5.0	3.4	1.6	0.4	setosa
5.2	3.5	1.5	0.2	setosa
5.2	3.4	1.4	0.2	setosa
4.7	3.2	1.6	0.2	setosa
4.8	3.1	1.6	0.2	setosa
5.4	3.4	1.5	0.4	setosa
5.2	4.1	1.5	0.1	setosa
5.5	4.2	1.4	0.2	setosa
4.9	3.1	1.5	0.2	setosa
5.0	3.2	1.2	0.2	setosa
5.5	3.5	1.3	0.2	setosa
4.9	3.6	1.4	0.1	setosa
4.4	3.0	1.3	0.2	setosa
5.1	3.4	1.5	0.2	setosa
5.0	3.5	1.3	0.3	setosa
4.5	2.3	1.3	0.3	setosa
4.4	3.2	1.3	0.2	setosa
5.0	3.5	1.6	0.6	setosa
5.1	3.8	1.9	0.4	setosa
4.8	3.0	1.4	0.3	setosa

5.1	3.8	1.6	0.2	setosa
4.6	3.2	1.4	0.2	setosa
5.3	3.7	1.5	0.2	setosa
5.0	3.3	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.9	3.1	4.9	1.5	versicolor
5.5	2.3	4.0	1.3	versicolor
6.5	2.8	4.6	1.5	versicolor

Pandoc supports several types of Markdown tables,[8] such as simple tables, multiline tables, grid tables, and pipe tables. What `knitr::kable()` generates is a simple table like this:

```
Table: A simple table in Markdown.

 Sepal.Length   Sepal.Width   Petal.Length   Petal.Width
-------------  ------------  -------------  ------------
          5.1           3.5            1.4           0.2
          4.9           3.0            1.4           0.2
          4.7           3.2            1.3           0.2
          4.6           3.1            1.5           0.2
          5.0           3.6            1.4           0.2
          5.4           3.9            1.7           0.4
```

You can use any types of Markdown tables in your document. To be

[8] http://pandoc.org/MANUAL.html#tables

able to cross-reference a Markdown table, it must have a labeled caption of the form `Table: (\#label) Caption here`, where `label` must have the prefix `tab:`, e.g., `tab:simple-table`.

If you decide to use other R packages to generate tables, you have to make sure the label for the table environment appears in the beginning of the table caption in the form `(\#label)` (again, `label` must have the prefix `tab:`). You have to be very careful about the *portability* of the table generating function: it should work for both HTML and LaTeX output automatically, so it must consider the output format internally (check `knitr::opts_knit$get('rmarkdown.pandoc.to')`). When writing out an HTML table, the caption must be written in the `<caption></caption>` tag. For simple tables, `kable()` should suffice. If you have to create complicated tables (e.g., with certain cells spanning across multiple columns/rows), you will have to take the aforementioned issues into consideration.

2.6 Cross-references

We have explained how cross-references work for equations (Section 2.2.1), theorems (Section 2.2.2), figures (Section 2.4), and tables (Section 2.5). In fact, you can also reference sections using the same syntax `\@ref(label)`, where `label` is the section ID. By default, Pandoc will generate an ID for all section headers, e.g., a section `# Hello World` will have an ID `hello-world`. We recommend you to manually assign an ID to a section header to make sure you do not forget to update the reference label after you change the section header. To assign an ID

to a section header, simply add `{#id}` to the end of the section header. Further attributes of section headers can be set using standard Pandoc syntax[9].

When a referenced label cannot be found, you will see two question marks like **??**, as well as a warning message in the R console when rendering the book.

You can also create text-based links using explicit or automatic section IDs or even the actual section header text.

- If you are happy with the section header as the link text, use it inside a single set of square brackets:
  - `[Section header text]`: example "A single document" via `[A single document]`
- There are two ways to specify custom link text:
  - `[link text][Section header text]`, e.g., "non-English books" via `[non-English books][Internationalization]`
  - `[link text](#ID)`, e.g., "Table stuff" via `[Table stuff](#tables)`

The Pandoc documentation provides more details on automatic section IDs[10] and implicit header references.[11]

Cross-references still work even when we refer to an item that is not on the current page of the PDF or HTML output. For example, see Equation (2.1) and Figure 2.4.

[9] http://pandoc.org/MANUAL.html#heading-identifiers

[10] http://pandoc.org/MANUAL.html#extension-auto_identifiers

[11] http://pandoc.org/MANUAL.html#extension-implicit_header_references

2.7 Custom blocks

We recommend that you use the new syntax to create custom blocks, which is introduced in Section 9.6 in the *R Markdown Cookbook*[12] (Xie et al., 2020b). We do not recommend using the `block` or `block2` engine described below. Those will be deprecated in a future version of **bookdown**. Please remember to switch to the new syntax for your content if you are still using one of these engines. Thanks for your understanding!

You can generate custom blocks using the `block` engine in **knitr**, i.e., the chunk option `engine = 'block'`, or the more compact syntax ```` ```{block} ````. This engine should be used in conjunction with the chunk option `type`, which takes a character string. When the `block` engine is used, it generates a `<div>` to wrap the chunk content if the output format is HTML, and a LaTeX environment if the output is LaTeX. The `type` option specifies the class of the `<div>` and the name of the LaTeX environment. For example, the HTML output of this chunk

````
```{block, type='FOO'}
Some text for this block.
```
````

will be this:

```
<div class="FOO">
Some text for this block.
</div>
```

and the LaTeX output will be this:

```
\begin{FOO}
Some text for this block.
\end{FOO}
```

It is up to the book author how to define the style of the block. You can define the style of the `<div>` in CSS and include it in the output via the `includes` option in the YAML metadata. Similarly, you may define the LaTeX environment via `\newenvironment` and include the definition in the LaTeX output via the `includes` option. For example, we may save the following style in a CSS file, say, `style.css`:

```
div.FOO {
  font-weight: bold;
  color: red;
}
```

And the YAML metadata of the R Markdown document can be:

```
---
output:
  bookdown::html_book:
    includes:
      in_header: style.css
---
```

We have defined a few types of blocks for this book to show notes, tips, and warnings, etc. Below are some examples:

R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under the terms of the GNU General Public License versions 2 or 3. For more information about these matters see http://www.gnu.org/licenses/.

R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under the terms of the GNU General Public License versions 2 or 3. For more information about these matters see http://www.gnu.org/licenses/.

R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under the terms of the GNU General Public License versions 2 or 3. For more information about these matters see http://www.gnu.org/licenses/.

R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under the terms of the GNU General Public License versions 2 or 3. For more information about these matters see http://www.gnu.org/licenses/.

R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under the terms of the GNU General Public License versions 2 or 3. For more information about these matters see http://www.gnu.org/licenses/.

The **knitr** `block` engine was designed to display simple content (typically a paragraph of plain text). You can use simple formatting syntax such as making certain words bold or italic, but more advanced syntax such as citations and cross-references will not work. However, there is an alternative engine named `block2` that supports arbitrary Markdown syntax, e.g.,

```
```{block2, type='FOO'}
Some text for this block [@citation-key].

- a list item
- another item

More text.
```
```

The `block2` engine should also be faster than the `block` engine if you have a lot of custom blocks in the document, but its implementation was based on a hack,[13] so we are not 100% sure if it is always going to work in the future. We have not seen problems with Pandoc v1.17.2 yet.

One more caveat for the `block2` engine: if the last element in the block is not an ordinary paragraph, you must leave a blank line at the end, e.g.,

```
```{block2, type='FOO'}
Some text for this block [@citation-key].

- a list item
- another item
- end the list with a blank line

```
```

The theorem and proof environments in Section 2.2.2 are actually implemented through the `block2` engine.

For all custom blocks based on the `block` or `block2` engine, there is one chunk option `echo` that you can use to show (`echo = TRUE`) or hide (`echo = FALSE`) the blocks.

[13] https://github.com/jgm/pandoc/issues/2453

2.8 Citations

Pandoc offers two methods for managing citations and bibliographic references in a document.

1. The default method is to use a Pandoc helper program called `pandoc-citeproc`[14], which follows the specifications of the Citation Style Language (CSL)[15] and obtains specific formatting instructions from one of the huge number of available CSL style files.[16]
2. Users may also choose to use either **natbib**[17] (based on `bibtex`) or **biblatex**[18] as a "citation package". In this case, the bibliographic data files need to be in the `bibtex` or `biblatex` format, and the document output format is limited to PDF. Again, various bibliographic styles are available (please consult the documentation of these packages).

   To use **natbib** or **biblatex** to process references, you can set the `citation_package` option of the R Markdown output format, e.g.,

```
output:
  pdf_document:
```

[14] https://github.com/jgm/pandoc-citeproc

[15] http://docs.citationstyles.org/en/1.0.1/specification.html

[16] https://www.zotero.org/styles/

[17] https://ctan.org/pkg/natbib

[18] https://ctan.org/pkg/biblatex

```
    citation_package: natbib
bookdown::pdf_book:
    citation_package: biblatex
```

Even if you choose `natbib` or `biblatex` for PDF output, all other output formats will be using `pandoc-citeproc`. If you use matching styles (e.g., `biblio-style: apa` for `biblatex` along with `csl: apa.csl` for `pandoc-citeproc`), output to PDF and to non-PDF formats will be very similar, though not necessarily identical.

For any non-PDF output format, `pandoc-citeproc` is the only available option. If consistency across PDF and non-PDF output formats is important, use `pandoc-citeproc` throughout.

The bibliographic data can be in several formats. We have only shown examples of BibTeX databases in this section, and please see the "Citations"[19] section of the Pandoc manual for other possible formats.

A BibTeX database is a plain-text file (with the conventional filename extension `.bib`) that consists of bibliography entries like this:

```
@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 = {2016},
```

[19] https://pandoc.org/MANUAL.html#citations

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

A bibliography entry starts with `@type{`, where `type` may be `article`, `book`, `manual`, and so on.[20] Then there is a citation key, like `R-base` in the above example. To cite an entry, use `@key` or `[@key]` (the latter puts the citation in braces), e.g., `@R-base` is rendered as R Core Team (2020), and `[@R-base]` generates "(R Core Team, 2020)". If you are familiar with the **natbib** package in LaTeX, `@key` is basically `\citet{key}`, and `[@key]` is equivalent to `\citep{key}`.

There are a number of fields in a bibliography entry, such as `title`, `author`, and `year`, etc. You may see `https://en.wikipedia.org/wiki/BibTeX` for possible types of entries and fields in BibTeX.

There is a helper function `write_bib()` in **knitr** to generate BibTeX entries automatically for R packages, e.g.,

```
# the second argument can be a .bib file
knitr::write_bib(c('knitr', 'stringr'), '', width = 60)
```

```
@Manual{R-knitr,
  title = {knitr: A General-Purpose Package for Dynamic
    Report Generation in R},
  author = {Yihui Xie},
  year = {2021},
```

[20] The type name is case-insensitive, so it does not matter if it is `manual`, `Manual`, or `MANUAL`.

```
  note = {R package version 1.33},
  url = {https://yihui.org/knitr/},
}

@Manual{R-stringr,
  title = {stringr: Simple, Consistent Wrappers for Common
    String Operations},
  author = {Hadley Wickham},
  year = {2019},
  note = {R package version 1.4.0},
  url = {https://CRAN.R-project.org/package=stringr},
}

@Book{knitr2015,
  title = {Dynamic Documents with {R} and knitr},
  author = {Yihui Xie},
  publisher = {Chapman and Hall/CRC},
  address = {Boca Raton, Florida},
  year = {2015},
  edition = {2nd},
  note = {ISBN 978-1498716963},
  url = {https://yihui.org/knitr/},
}

@InCollection{knitr2014,
  booktitle = {Implementing Reproducible Computational
    Research},
  editor = {Victoria Stodden and Friedrich Leisch and Roger
    D. Peng},
```

```
  title = {knitr: A Comprehensive Tool for Reproducible
    Research in {R}},
  author = {Yihui Xie},
  publisher = {Chapman and Hall/CRC},
  year = {2014},
  note = {ISBN 978-1466561595},
  url = {http://www.crcpress.com/product/isbn/
    9781466561595},
}
```

Once you have one or multiple `.bib` files, you may use the field `bibliography` in the YAML metadata of your first R Markdown document (which is typically `index.Rmd`), and you can also specify the bibliography style via `biblio-style` (this only applies to PDF output), e.g.,

```
---
bibliography: ["one.bib", "another.bib", "yet-another.bib"]
biblio-style: "apalike"
link-citations: true
---
```

The field `link-citations` can be used to add internal links from the citation text of the author-year style to the bibliography entry in the HTML output.

When the output format is LaTeX, the list of references will be automatically put in a chapter or section at the end of the document. For non-LaTeX output, you can add an empty chapter as the last chapter of your book. For example, if your last chapter is the Rmd file `06-references.Rmd`, its content can be an inline R expression:

```
`r if (knitr::is_html_output()) '# References {-}'`
```

For more detailed instructions and further examples on how to use citations, please see the "Citations" section of the Pandoc manual.

2.9 Index

Currently the index is only supported for LaTeX/PDF output. To print an index after the book, you can use the LaTeX package **makeidx** in the preamble (see Section 4.1):

```
\usepackage{makeidx}
\makeindex
```

Then insert `\printindex` at the end of your book through the YAML option `includes -> after_body`. An index entry can be created via the `\index{}` command in the book body, e.g., `\index{GIT}`.

2.10 HTML widgets

Although one of R's greatest strengths is data visualization, there are a large number of JavaScript libraries for much richer data visualization. These libraries can be used to build interactive applications that

can easily render in web browsers, so users do not need to install any additional software packages to view the visualizations. One way to bring these JavaScript libraries into R is through the **htmlwidgets**[21] package (Vaidyanathan et al., 2020).

HTML widgets can be rendered as a standalone web page (like an R plot), or embedded in R Markdown documents and Shiny applications. They were originally designed for HTML output only, and they require the availability of JavaScript, so they will not work in non-HTML output formats, such as LaTeX/PDF. Before **knitr** v1.13, you will get an error when you render HTML widgets to an output format that is not HTML. Since **knitr** v1.13, HTML widgets will be rendered automatically as screenshots taken via the **webshot** package (Chang, 2019). Of course, you need to install the **webshot** package. Additionally, you have to install PhantomJS (`http://phantomjs.org`), since it is what **webshot** uses to capture screenshots. Both **webshot** and PhantomJS can be installed automatically from R:

```
install.packages('webshot')
webshot::install_phantomjs()
```

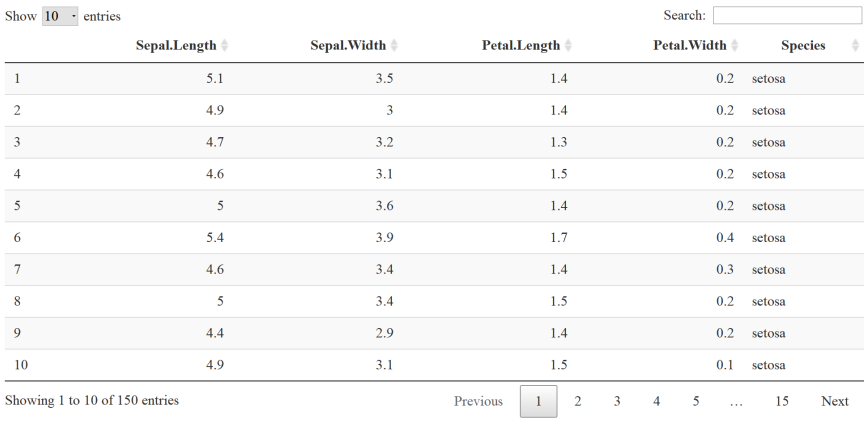
The function `install_phantomjs()` works for Windows, OS X, and Linux. You may also choose to download and install PhantomJS by yourself, if you are familiar with modifying the system environment variable PATH.

When **knitr** detects an HTML widget object in a code chunk, it either renders the widget normally when the current output format is HTML,

[21] `http://htmlwidgets.org`

or saves the widget as an HTML page and calls **webshot** to capture the screen of the HTML page when the output format is not HTML. Here is an example of a table created from the **DT** package (Xie et al., 2020a):

```
DT::datatable(iris)
```



Show 10 entries Search:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa

Showing 1 to 10 of 150 entries Previous 1 2 3 4 5 … 15 Next

图 **2.5:** A table widget rendered via the DT package.

If you are reading this book as web pages now, you should see an interactive table generated from the above code chunk, e.g., you may sort the columns and search in the table. If you are reading a non-HTML version of this book, you should see a screenshot of the table. The screenshot may look a little different with the actual widget rendered in the web browser, due to the difference between a real web browser and PhantomJS's virtual browser.

There are a number of **knitr** chunk options related to screen-capturing. First, if you are not satisfied with the quality of the automatic screenshots, or want a screenshot of the widget of a particular

state (e.g., after you click and sort a certain column of a table), you may capture the screen manually, and provide your own screenshot via the chunk option `screenshot.alt` (alternative screenshots). This option takes the paths of images. If you have multiple widgets in a chunk, you can provide a vector of image paths. When this option is present, **knitr** will no longer call **webshot** to take automatic screenshots.

Second, sometimes you may want to force **knitr** to use static screenshots instead of rendering the actual widgets even on HTML pages. In this case, you can set the chunk option `screenshot.force = TRUE`, and widgets will always be rendered as static images. Note that you can still choose to use automatic or custom screenshots.

Third, **webshot** has some options to control the automatic screenshots, and you may specify these options via the chunk option `screenshot.opts`, which takes a list like `list(delay = 2, cliprect = 'viewport')`. See the help page `?webshot::webshot` for the full list of possible options, and the package vignette[22] `vignette('intro', package = 'webshot')` illustrates the effect of these options. Here the `delay` option can be important for widgets that take long time to render: `delay` specifies the number of seconds to wait before PhantomJS takes the screenshot. If you see an incomplete screenshot, you may want to specify a longer delay (the default is 0.2 seconds).

Fourth, if you feel it is slow to capture the screenshots, or do not want to do it every time the code chunk is executed, you may use the chunk option `cache = TRUE` to cache the chunk. Caching works for both HTML and non-HTML output formats.

Screenshots behave like normal R plots in the sense that many

[22] https://cran.rstudio.com/web/packages/webshot/vignettes/intro.html

chunk options related to figures also apply to screenshots, including `fig.width`, `fig.height`, `out.width`, `fig.cap`, and so on. So you can specify the size of screenshots in the output document, and assign figure captions to them as well. The image format of the automatic screenshots can be specified via the chunk option `dev`, and possible values are `pdf`, `png`, and `jpeg`. The default for PDF output is `pdf`, and it is `png` for other types of output. Note that `pdf` may not work as faithfully as `png`: sometimes there are certain elements on an HTML page that fail to render to the PDF screenshot, so you may want to use `dev = 'png'` even for PDF output. It depends on specific cases of HTML widgets, and you can try both `pdf` and `png` (or `jpeg`) before deciding which format is more desirable.

2.11 Web pages and Shiny apps

Similar to HTML widgets, arbitrary web pages can be embedded in the book. You can use the function `knitr::include_url()` to include a web page through its URL. When the output format is HTML, an `iframe` is used;[23] in other cases, **knitr** tries to take a screenshot of the web page (or use the custom screenshot you provided). All chunk options are the same as those for HTML widgets. One option that may require your special attention is the `delay` option: HTML widgets are rendered locally, so usually they are fast to load for PhantomJS to take screenshots, but an arbitrary URL may take longer to load, so you may want

[23] An `iframe` is basically a box on one web page to embed another web page.

to use a larger `delay` value, e.g., use the chunk option `screenshot.opts = list(delay = 5)`.

A related function is `knitr::include_app()`, which is very similar to `include_url()`, and it was designed for embedding Shiny apps via their URLs in the output. Its only difference with `include_url()` is that it automatically adds a query parameter `?showcase=0` to the URL, if no other query parameters are present in the URL, to disable the Shiny showcase mode, which is unlikely to be useful for screenshots or iframes. If you do want the showcase mode, use `include_url()` instead of `include_app()`. Below is a Shiny app example (Figure 2.6):

```
knitr::include_app('https://yihui.shinyapps.io/miniUI/', height = '600px')
```

Again, you will see a live app if you are reading an HTML version of this book, and a static screenshot if you are reading other types of formats. The above Shiny app was created using the **miniUI** package (Cheng, 2018), which provides layout functions that are particularly nice for Shiny apps on small screens. If you use normal Shiny layout functions, you are likely to see vertical and/or horizontal scrollbars in the iframes because the page size is too big to fit in an iframe. When the default width of the iframe is too small, you may use the chunk option `out.width` to change it. For the height of the iframe, use the `height` argument of `include_url()`/`include_app()`.

Shiny apps may take even longer to load than usual URLs. You may want to use a conservative value for the `delay` option, e.g., 10. Needless to say, `include_url()` and `include_app()` require a working Internet connection, unless you have previously cached the chunk (but web

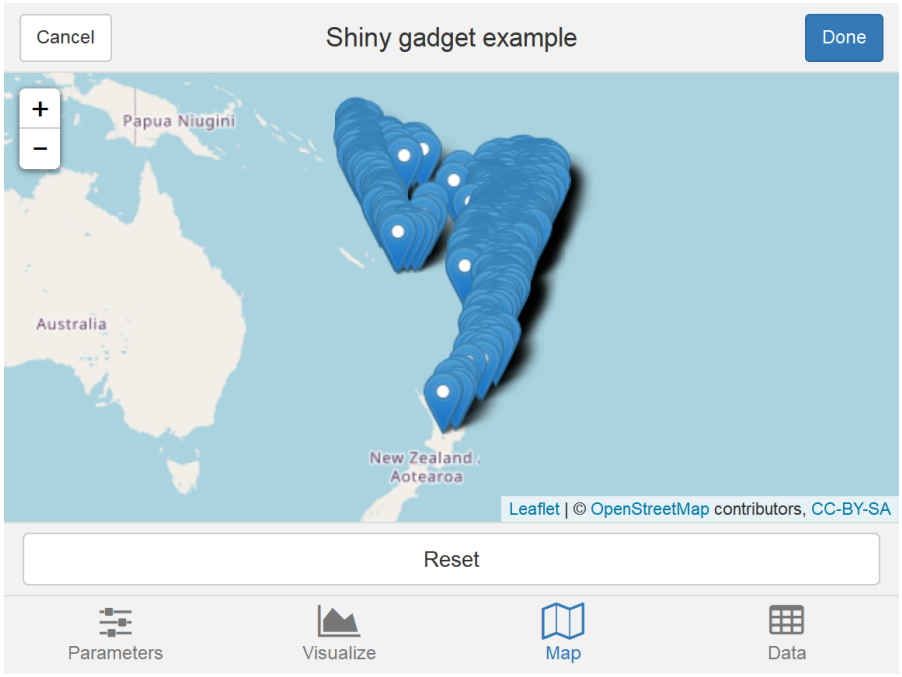


图 **2.6:** A Shiny app created via the miniUI package; you can see a live version at https://yihui.shinyapps.io/miniUI/.

pages inside iframes still will not work without an Internet connection).

3

Output Formats

The **bookdown** package primarily supports three types of output formats: HTML, LaTeX/PDF, and e-books. In this chapter, we introduce the possible options for these formats. Output formats can be specified either in the YAML metadata of the first Rmd file of the book, or in a separate YAML file named `_output.yml` under the root directory of the book. Here is a brief example of the former (output formats are specified in the `output` field of the YAML metadata):

```
---
title: "An Impressive Book"
author: "Li Lei and Han Meimei"
output:
  bookdown::gitbook:
    lib_dir: assets
    split_by: section
    config:
      toolbar:
        position: static
  bookdown::pdf_book:
    keep_tex: yes
  bookdown::html_book:
```

```
    css: toc.css
documentclass: book
---
```

Here is an example of `_output.yml`:

```
bookdown::gitbook:
  lib_dir: assets
  split_by: section
  config:
    toolbar:
      position: static
bookdown::pdf_book:
  keep_tex: yes
bookdown::html_book:
  css: toc.css
```

In this case, all formats should be at the top level, instead of under an `output` field. You do not need the three dashes `---` in `_output.yml`.

3.1 HTML

The main difference between rendering a book (using **bookdown**) with rendering a single R Markdown document (using **rmarkdown**) to HTML is that a book will generate multiple HTML pages by default — normally one HTML file per chapter. This makes it easier to bookmark

a certain chapter or share its URL with others as you read the book, and faster to load a book into the web browser. Currently we have provided a number of different styles for HTML output: the GitBook style, the Bootstrap style, and the Tufte style.

3.1.1 GitBook style

The GitBook style was borrowed from GitBook, a project launched by Friendcode, Inc. (https://www.gitbook.com) and dedicated to helping authors write books with Markdown. It provides a beautiful style, with a layout consisting of a sidebar showing the table of contents on the left, and the main body of a book on the right. The design is responsive to the window size, e.g., the navigation buttons are displayed on the left/right of the book body when the window is wide enough, and collapsed into the bottom when the window is narrow to give readers more horizontal space to read the book body.

We have made several improvements over the original GitBook project. The most significant one is that we replaced the Markdown engine with R Markdown v2 based on Pandoc, so that there are a lot more features for you to use when writing a book:

- You can embed R code chunks and inline R expressions in Markdown, and this makes it easy to create reproducible documents and frees you from synchronizing your computation with its actual output (**knitr** will take care of it automatically).
- The Markdown syntax is much richer: you can write anything that Pandoc's Markdown supports, such as LaTeX math expressions and citations.

- You can embed interactive content in the book (for HTML output only), such as HTML widgets and Shiny apps.

We have also added some useful features in the user interface that we will introduce in detail soon. The output format function for the GitBook style in **bookdown** is `gitbook()`. Here are its arguments:

```
gitbook(fig_caption = TRUE, number_sections = TRUE,
  self_contained = FALSE, anchor_sections = TRUE,
  lib_dir = "libs", pandoc_args = NULL, ...,
  template = "default",
  split_by = c("chapter", "chapter+number", "section", "section+number", "rmd", "none"),
  split_bib = TRUE, config = list(), table_css = TRUE)
```

Most arguments are passed to `rmarkdown::html_document()`, including `fig_caption`, `lib_dir`, and `...`. You can check out the help page of `rmarkdown::html_document()` for the full list of possible options. We strongly recommend you to use `fig_caption = TRUE` for two reasons: 1) it is important to explain your figures with captions; 2) enabling figure captions means figures will be placed in floating environments when the output is LaTeX, otherwise you may end up with a lot of white space on certain pages. The format of figure/table numbers depends on if sections are numbered or not: if `number_sections = TRUE`, these numbers will be of the format `X.i`, where `X` is the chapter number, and `i` in an incremental number; if sections are not numbered, all figures/tables will be numbered sequentially through the book from 1, 2, ..., N. Note that in either case, figures and tables will be numbered independently.

Among all possible arguments in `...`, you are most likely to use the `css`

argument to provide one or more custom CSS files to tweak the default CSS style. There are a few arguments of `html_document()` that have been hard-coded in `gitbook()` and you cannot change them: `toc = TRUE` (there must be a table of contents), `theme = NULL` (not using any Bootstrap themes), and `template` (there exists an internal GitBook template).

Please note that if you change `self_contained = TRUE` to make self-contained HTML pages, the total size of all HTML files can be significantly increased since there are many JS and CSS files that have to be embedded in every single HTML file.

Besides these `html_document()` options, `gitbook()` has three other arguments: `split_by`, `split_bib`, and `config`. The `split_by` argument specifies how you want to split the HTML output into multiple pages, and its possible values are:

- `rmd`: use the base filenames of the input Rmd files to create the HTML filenames, e.g., generate `chapter3.html` for `chapter3.Rmd`.
- `none`: do not split the HTML file (the book will be a single HTML file).
- `chapter`: split the file by the first-level headers.
- `section`: split the file by the second-level headers.
- `chapter+number` and `section+number`: similar to `chapter` and `section`, but the files will be numbered.

For `chapter` and `section`, the HTML filenames will be determined by the header identifiers, e.g., the filename for the first chapter with a chapter title `# Introduction` will be `introduction.html` by default. For `chapter+number` and `section+number`, the chapter/section numbers will be prepended to the HTML filenames, e.g., `1-introduction.html` and `2-1-literature.html`. The header identifier is automatically generated

from the header text by default,[1] and you can manually specify an identifier using the syntax `{#your-custom-id}` after the header text, e.g.,

```
# An Introduction {#introduction}


The default identifier is `an-introduction` but we changed
it to `introduction`.
```

By default, the bibliography is split and relevant citation items are put at the bottom of each page, so that readers do not have to navigate to a different bibliography page to see the details of citations. This feature can be disabled using `split_bib = FALSE`, in which case all citations are put on a separate page.

There are several sub-options in the `config` option for you to tweak some details in the user interface. Recall that all output format options (not only for `bookdown::gitbook`) can be either passed to the format function if you use the command-line interface `bookdown::render_book()`, or written in the YAML metadata. We display the default sub-options of `config` in the `gitbook` format as YAML metadata below (note that they are indented under the `config` option):

```
bookdown::gitbook:
  config:
    toc:
```

[1] To see more details on how an identifier is automatically generated, see the `auto_identifiers` extension in Pandoc's documentation http://pandoc.org/MANUAL.html#header-identifiers

```
    collapse: subsection
    scroll_highlight: yes
    before: null
    after: null
  toolbar:
    position: fixed
  edit : null
  download: null
  search:
    engine: lunr # or fuse
    # options to control/tune search engine behaviour (for
    # fuse.js, refer to https://fusejs.io/api/options.html)
    options: null
  fontsettings:
    theme: white
    family: sans
    size: 2
  sharing:
    facebook: yes
    github: no
    twitter: yes
    linkedin: no
    weibo: no
    instapaper: no
    vk: no
    whatsapp: no
    all: ['facebook', 'twitter', 'linkedin', 'weibo', 'instapaper']
  info: yes
```

The `toc` option controls the behavior of the table of contents (TOC). You can collapse some items initially when a page is loaded via the `collapse` option. Its possible values are `subsection`, `section`, `none` (or `null`). This option can be helpful if your TOC is very long and has more than three levels of headings: `subsection` means collapsing all TOC items for subsections (X.X.X), `section` means those items for sections (X.X) so only the top-level headings are displayed initially, and `none` means not collapsing any items in the TOC. For those collapsed TOC items, you can toggle their visibility by clicking their parent TOC items. For example, you can click a chapter title in the TOC to show/hide its sections.

The `scroll_highlight` option in `toc` indicates whether to enable highlighting of TOC items as you scroll the book body (by default this feature is enabled). Whenever a new header comes into the current viewport as you scroll down/up, the corresponding item in TOC on the left will be highlighted.

Since the sidebar has a fixed width, when an item in the TOC is truncated because the heading text is too wide, you can hover the cursor over it to see a tooltip showing the full text.

You may add more items before and after the TOC using the HTML tag `<li>`. These items will be separated from the TOC using a horizontal divider. You can use the pipe character `|` so that you do not need to escape any characters in these items following the YAML syntax, e.g.,

```
toc:
  before: |
    <li><a href="...">My Awesome Book</a></li>
    <li><a href="...">John Smith</a></li>
  after: |
```

```
<li><a href="https://github.com/rstudio/bookdown">
Proudly published with bookdown</a></li>
```

As you navigate through different HTML pages, we will try to preserve the scroll position of the TOC. Normally you will see the scrollbar in the TOC at a fixed position even if you navigate to the next page. However, if the TOC item for the current chapter/section is not visible when the page is loaded, we will automatically scroll the TOC to make it visible to you.

图 **3.1:** The GitBook toolbar.

The GitBook style has a toolbar (Figure 3.1) at the top of each page that allows you to dynamically change the book settings. The `toolbar` option has a sub-option `position`, which can take values `fixed` or `static`. The default is that the toolbar will be fixed at the top of the page, so even if you scroll down the page, the toolbar is still visible there. If it is `static`, the toolbar will not scroll with the page, i.e., once you scroll away, you will no longer see it.

The first button on the toolbar can toggle the visibility of the sidebar. You can also hit the s key on your keyboard to do the same thing. The GitBook style can remember the visibility status of the sidebar, e.g., if you closed the sidebar, it will remain closed the next time you open

the book. In fact, the GitBook style remembers many other settings as well, such as the search keyword and the font settings.

The second button on the toolbar is the search button. Its keyboard shortcut is `F` (Find). When the button is clicked, you will see a search box at the top of the sidebar. As you type in the box, the TOC will be filtered to display the sections that match the search keyword. Now you can use the arrow keys `Up`/`Down` to highlight the previous/next match in the search results. When you click the search button again (or hit `F` outside the search box), the search keyword will be emptied and the search box will be hidden. To disable searching, set the option `search: no` in `config`.

The third button is for font/theme settings. The reader can change the font size (bigger or smaller), the font family (serif or sans serif), and the theme (`White`, `Sepia`, or `Night`). You can set the initial value of these settings via the `fontsettings` option. Font size is measured on a scale of 0-4; the initial value can be set to 1, 2 (default), 3, or 4. The button can be removed from the toolbar by setting `fontsettings: null` (or `no`).

```
# changing the default
    fontsettings:
      theme: night
      family: serif
      size: 3
```

The `edit` option is the same as the option mentioned in Section 4.4. If it is not empty, an edit button will be added to the toolbar. This was designed for potential contributors to the book to contribute by edit-

ing the book on GitHub after clicking the button and sending pull requests. The `history` and `view` options work the same way.

If your book has other output formats for readers to download, you may provide the `download` option so that a download button can be added to the toolbar. This option takes either a character vector, or a list of character vectors with the length of each vector being 2. When it is a character vector, it should be either a vector of filenames, or filename extensions, e.g., both of the following settings are okay:

```
download: ["book.pdf", "book.epub"]
download: ["pdf", "epub", "mobi"]
```

When you only provide the filename extensions, the filename is derived from the book filename of the configuration file `_bookdown.yml` (Section 4.4). When `download` is `null`, `gitbook()` will look for PDF, EPUB, and MOBI files in the book output directory, and automatically add them to the `download` option. If you just want to suppress the download button, use `download: no`. All files for readers to download will be displayed in a drop-down menu, and the filename extensions are used as the menu text. When the only available format for readers to download is PDF, the download button will be a single PDF button instead of a drop-down menu.

An alternative form for the value of the `download` option is a list of length-2 vectors, e.g.,

```
download: [["book.pdf", "PDF"], ["book.epub", "EPUB"]]
```

You can also write it as:

```
download:
  - ["book.pdf", "PDF"]
  - ["book.epub", "EPUB"]
```

Each vector in the list consists of the filename and the text to be displayed in the menu. Compared to the first form, this form allows you to customize the menu text, e.g., you may have two different copies of the PDF for readers to download and you will need to make the menu items different.

On the right of the toolbar, there are some buttons to share the link on social network websites such as Twitter, Facebook, and Linkedin. You can use the `sharing` option to decide which buttons to enable. If you want to get rid of these buttons entirely, use `sharing: null` (or `no`).

Another button shown on the toolbar is the information ('i') button that lists keyboard shortcuts available to navigate the document. This button can be hidden by setting `info: no`.

Finally, there are a few more top-level options in the YAML metadata that can be passed to the GitBook HTML template via Pandoc. They may not have clear visible effects on the HTML output, but they may be useful when you deploy the HTML output as a website. These options include:

- `description`: A character string to be written to the `content` attribute of the tag `<meta name="description" content="">` in the HTML head (if missing, the title of the book will be used). This can be useful for

search engine optimization (SEO). Note that it should be plain text without any Markdown formatting such as `_italic_` or `**bold**`.

- `url`: The URL of book's website, e.g., `https\://bookdown.org/yihui/bookdown/`.[2]
- `github-repo`: The GitHub repository of the book of the form `user/repo`.
- `cover-image`: The path to the cover image of the book.
- `apple-touch-icon`: A path to an icon (e.g., a PNG image). This is for iOS only: when the website is added to the Home screen, the link is represented by this icon.
- `apple-touch-icon-size`: The size of the icon (by default, 152 x 152 pixels).
- `favicon`: A path to the "favorite icon". Typically this icon is displayed in the browser's address bar, or in front of the page title on the tab if the browser support tabs.

Below we show some sample YAML metadata (again, please note that these are *top-level* options):

```yaml
---
title: "An Awesome Book"
author: "John Smith"
description: "This book introduces the ABC theory, and ..."
url: 'https\://bookdown.org/john/awesome/'
github-repo: "john/awesome"
cover-image: "images/cover.png"
```

[2]The backslash before : is due to a technical issue: we want to prevent Pandoc from translating the link to HTML code `<a href="..."></a>`. More details at https://github.com/jgm/pandoc/issues/2139.

```
apple-touch-icon: "touch-icon.png"
apple-touch-icon-size: 120
favicon: "favicon.ico"
---
```

A nice effect of setting `description` and `cover-image` is that when you share the link of your book on some social network websites such as Twitter, the link can be automatically expanded to a card with the cover image and description of the book.

3.1.2 Bootstrap style

If you have used R Markdown before, you should be familiar with the Bootstrap style (http://getbootstrap.com), which is the default style of the HTML output of R Markdown. The output format function in **rmarkdown** is `html_document()`, and we have a corresponding format `html_book()` in **bookdown** using `html_document()` as the base format. In fact, there is a more general format `html_chapters()` in **bookdown** and `html_book()` is just its special case:

```
html_chapters(toc = TRUE, number_sections = TRUE,
  fig_caption = TRUE, lib_dir = "libs",
  template = bookdown_file("templates/default.html"),
  pandoc_args = NULL, ...,
  base_format = rmarkdown::html_document,
  split_bib = TRUE, page_builder = build_chapter,
  split_by = c("section+number", "section", "chapter+number", "chapter", "rmd", "none"))
```

Note that it has a `base_format` argument that takes a base output format function, and `html_book()` is basically `html_chapters(base_format = rmarkdown::html_document)`. All arguments of `html_book()` are passed to `html_chapters()`:

```
html_book(...)
```

That means that you can use most arguments of `rmarkdown::html_document`, such as `toc` (whether to show the table of contents), `number_sections` (whether to number section headings), and so on. Again, check the help page of `rmarkdown::html_document` to see the full list of possible options. Note that the argument `self_contained` is hard-coded to `FALSE` internally, so you cannot change the value of this argument. We have explained the argument `split_by` in the previous section.

The arguments `template` and `page_builder` are for advanced users, and you do not need to understand them unless you have strong need to customize the HTML output, and those many options provided by `rmarkdown::html_document()` still do not give you what you want.

If you want to pass a different HTML template to the `template` argument, the template must contain three pairs of HTML comments, and each comment must be on a separate line:

- `<!--bookdown:title:start-->` and `<!--bookdown:title:end-->` to mark the title section of the book. This section will be placed only on the first page of the rendered book;
- `<!--bookdown:toc:start-->` and `<!--bookdown:toc:end-->` to mark the table of contents section, which will be placed on all HTML pages;

- `<!--bookdown:body:start-->` and `<!--bookdown:body:end-->` to mark the HTML body of the book, and the HTML body will be split into multiple separate pages. Recall that we merge all R Markdown or Markdown files, render them into a single HTML file, and split it.

You may open the default HTML template to see where these comments were inserted:

```
bookdown:::bookdown_file('templates/default.html')
# you may use file.edit() to open this file
```

Once you know how **bookdown** works internally to generate multiple-page HTML output, it will be easier to understand the argument `page_builder`, which is a function to compose each individual HTML page using the HTML fragments extracted from the above comment tokens. The default value of `page_builder` is a function `build_chapter` in **bookdown**, and its source code is relatively simple (ignore those internal functions like `button_link()`):

```
build_chapter = function(
  head, toc, chapter, link_prev, link_next, rmd_cur, html_cur, foot
) {
  # add a has-sub class to the <li> items that has sub lists
  toc = gsub('^(<li>)(.+<ul>)$', '<li class="has-sub">\\2', toc)
  paste(c(
    head,
    '<div class="row">',
```

```
    '<div class="col-sm-12">',
    toc,
    '</div>',
    '</div>',
    '<div class="row">',
    '<div class="col-sm-12">',
    chapter,
    '<p style="text-align: center;">',
    button_link(link_prev, 'Previous'),
    source_link(rmd_cur, type = 'edit'),
    source_link(rmd_cur, type = 'history'),
    source_link(rmd_cur, type = 'view'),
    button_link(link_next, 'Next'),
    '</p>',
    '</div>',
    '</div>',
    foot
  ), collapse = '\n')
}
```

Basically, this function takes a number of components like the HTML head, the table of contents, the chapter body, and so on, and it is expected to return a character string which is the HTML source of a complete HTML page. You may manipulate all components in this function using text-processing functions like `gsub()` and `paste()`.

What the default page builder does is to put TOC in the first row, the body in the second row, navigation buttons at the bottom of the body, and concatenate them with the HTML head and foot. Here is a sketch

of the HTML source code that may help you understand the output of `build_chapter()`:

```
<html>
  <head>
    <title>A Nice Book</title>
  </head>
  <body>

    <div class="row">TOC</div>

    <div class="row">
      CHAPTER BODY
      <p>
        <button>PREVIOUS</button>
        <button>NEXT</button>
      </p>
    </div>

  </body>
</html>
```

For all HTML pages, the main difference is the chapter body, and most of the rest of the elements are the same. The default output from `html_book()` will include the Bootstrap CSS and JavaScript files in the `<head>` tag.

The TOC is often used for navigation purposes. In the GitBook style, the TOC is displayed in the sidebar. For the Bootstrap style, we did not

apply a special style to it, so it is shown as a plain unordered list (in the HTML tag `<ul>`). It is easy to turn this list into a navigation bar with some CSS techniques. We have provided a CSS file `toc.css` in this package that you can use, and you can find it here: https://github.com/rstudio/bookdown/blob/master/inst/examples/css/toc.css

You may copy this file to the root directory of your book, and apply it to the HTML output via the `css` option, e.g.,

```
---
output:
  bookdown::html_book:
    toc: yes
    css: toc.css
---
```

There are many possible ways to turn `<ul>` lists into navigation menus if you do a little bit searching on the web, and you can choose a menu style that you like. The `toc.css` we just mentioned is a style with white menu texts on a black background, and supports sub-menus (e.g., section titles are displayed as drop-down menus under chapter titles).

As a matter of fact, you can get rid of the Bootstrap style in `html_document()` if you set the `theme` option to `null`, and you are free to apply arbitrary styles to the HTML output using the `css` option (and possibly the `includes` option if you want to include arbitrary content in the HTML head/foot).

3.1.3 Tufte style

Like the Bootstrap style, the Tufte style is provided by an output format `tufte_html_book()`, which is also a special case of `html_chapters()` using `tufte::tufte_html()` as the base format. Please see the **tufte** package (Xie and Allaire, 2020) if you are not familiar with the Tufte style. Basically, it is a layout with a main column on the left and a margin column on the right. The main body is in the main column, and the margin column is used to place footnotes, margin notes, references, and margin figures, and so on.

All arguments of `tufte_html_book()` have exactly the same meanings as `html_book()`, e.g., you can also customize the CSS via the `css` option. There are a few elements that are specific to the Tufte style, though, such as margin notes, margin figures, and full-width figures. These elements require special syntax to generate; please see the documentation of the **tufte** package. Note that you do not need to do anything special to footnotes and references (just use the normal Markdown syntax `^[footnote]` and `[@citation]`), since they will be automatically put in the margin. A brief YAML example of the `tufte_html_book` format:

```
---
output:
  bookdown::tufte_html_book:
    toc: yes
    css: toc.css
---
```

3.2 LaTeX/PDF

We strongly recommend that you use an HTML output format instead of LaTeX when you develop a book, since you will not be too distracted by the typesetting details, which can bother you a lot if you constantly look at the PDF output of a book. Leave the job of careful typesetting to the very end (ideally after you have really finished the content of the book).

The LaTeX/PDF output format is provided by `pdf_book()` in **bookdown**. There is not a significant difference between `pdf_book()` and the `pdf_document()` format in **rmarkdown**. The main purpose of `pdf_book()` is to resolve the labels and cross-references written using the syntax described in Sections 2.4, 2.5, and 2.6. If the only output format that you want for a book is LaTeX/PDF, you may use the syntax specific to LaTeX, such as `\label{}` to label figures/tables/sections, and `\ref{}` to cross-reference them via their labels, because Pandoc supports LaTeX commands in Markdown. However, the LaTeX syntax is not portable to other output formats, such as HTML and e-books. That is why we introduced the syntax `(\#label)` for labels and `\@ref(label)` for cross-references.

There are some top-level YAML options that will be applied to the LaTeX output. For a book, you may change the default document class to `book` (the default is `article`), and specify a bibliography style required by your publisher. A brief YAML example:

```
---
documentclass: book
bibliography: [book.bib, packages.bib]
biblio-style: apalike
---
```

There are a large number of other YAML options that you can specify for LaTeX output, such as the paper size, font size, page margin, line spacing, font families, and so on. See http://pandoc.org/MANUAL.html#variables-for-latex for a full list of options.

The `pdf_book()` format is a general format like `html_book()`, and it also has a `base_format` argument:

```
pdf_book(toc = TRUE, number_sections = TRUE,
  fig_caption = TRUE, pandoc_args = NULL, ...,
  base_format = rmarkdown::pdf_document,
  toc_unnumbered = TRUE, toc_appendix = FALSE,
  toc_bib = FALSE, quote_footer = NULL,
  highlight_bw = FALSE)
```

You can change the `base_format` function to other output format functions, and **bookdown** has provided a simple wrapper function `tufte_book2()`, which is basically `pdf_book(base_format = tufte::tufte_book)`, to produce a PDF book using the Tufte PDF style (again, see the **tufte** package).

3.3 E-Books

Currently **bookdown** provides two e-book formats, EPUB and MOBI. Books in these formats can be read on devices like smartphones, tablets, or special e-readers such as Kindle.

3.3.1 EPUB

To create an EPUB book, you can use the `epub_book()` format. It has some options in common with `rmarkdown::html_document()`:

```
epub_book(fig_width = 5, fig_height = 4, dev = "png",
  fig_caption = TRUE, number_sections = TRUE,
  toc = FALSE, toc_depth = 3, stylesheet = NULL,
  cover_image = NULL, metadata = NULL,
  chapter_level = 1, epub_version = c("epub3", "epub"),
  md_extensions = NULL, pandoc_args = NULL,
  template = "default")
```

The option `toc` is turned off because the e-book reader can often figure out a TOC automatically from the book, so it is not necessary to add a few pages for the TOC. There are a few options specific to EPUB:

- `stylesheet`: It is similar to the `css` option in HTML output formats, and you can customize the appearance of elements using CSS.
- `cover_image`: The path to the cover image of the book.
- `metadata`: The path to an XML file for the metadata of the book (see Pandoc documentation for more details).

- `chapter_level`: Internally an EPUB book is a series of "chapter" files, and this option determines the level by which the book is split into these files. This is similar to the `split_by` argument of HTML output formats we mentioned in Section 3.1, but an EPUB book is a single file, and you will not see these "chapter" files directly. The default level is the first level, and if you set it to 2, it means the book will be organized by section files internally, which may allow the reader to load the book more quickly.
- `epub_version`: Version 3 or 2 of EPUB.

An EPUB book is essentially a collection of HTML pages, e.g., you can apply CSS rules to its elements, embed images, insert math expressions (because MathML is partially supported), and so on. Figure/table captions, cross-references, custom blocks, and citations mentioned in Chapter 2 also work for EPUB. You may compare the EPUB output of this book to the HTML output, and you will see that the only major difference is the visual appearance.

There are several EPUB readers available, including Calibre (`https://www.calibre-ebook.com`), Apple's iBooks, and Google Play Books.

3.3.2 MOBI

MOBI e-books can be read on Amazon's Kindle devices. Pandoc does not support MOBI output natively, but you may use third-party tools to convert EPUB to MOBI. One possible tool is Calibre. Calibre is open-source and free, and supports conversion among many more formats. For example, you can convert HTML to EPUB, Word documents to MOBI, and so on. The function `calibre()` in **bookdown** is a wrapper function of the command-line utility `ebook-convert` in Calibre. You

need to make sure that the executable `ebook-convert` can be found via the environment variable `PATH`. If you use macOS, you can install Calibre with Homebrew (https://brew.sh) via the command `brew cask install calibre`, so you do not need to worry about the `PATH` issue.

3.4 A single document

Sometimes you may not want to write a book, but a single long-form article or report instead. Usually what you do is call `rmarkdown::render()` with a certain output format. The main features missing there are the automatic numbering of figures/tables/equations, and cross-referencing figures/tables/equations/sections. We have factored out these features from **bookdown**, so that you can use them without having to prepare a book of multiple Rmd files.

The functions `html_document2()`, `tufte_html2()`, `pdf_document2()`, `word_document2()`, `tufte_handout2()`, and `tufte_book2()` are designed for this purpose. If you render an R Markdown document with the output format, say, `bookdown::html_document2`, you will get figure/table numbers and be able to cross-reference them in the single HTML page using the syntax described in Chapter 2.

Below are a few examples of these output formats in the YAML metadata of a single Rmd file (you can also add these formats to the `_output.yml` file):

```
output:
  bookdown::html_document2: default
  bookdown::pdf_document2:
    keep_tex: true
  bookdown::word_document2:
    toc: true
```

The above HTML and PDF output format functions are basically wrappers of output formats `bookdown::html_book` and `bookdown::pdf_book`, in the sense that they changed the `base_format` argument. For example, you can take a look at the source code of `pdf_document2`:

```
bookdown::pdf_document2
```

```
## function (...)
## {
##     pdf_book(..., base_format = rmarkdown::pdf_document)
## }
## <bytecode: 0x000000001cd59b60>
## <environment: namespace:bookdown>
```

After you know this fact, you can apply the same idea to other output formats by using the appropriate `base_format`. For example, you can port the **bookdown** features to the `jss_article` format in the **rticles** package (Allaire et al., 2021b) by using the YAML metadata:

```
output:
  bookdown::pdf_book:
    base_format: rticles::jss_article
```

Then you will be able to use all features we introduced in Chapter 2.

Although the `gitbook()` format was designed primarily for books, you can actually also apply it to a single R Markdown document. The only difference is that there will be no search button on the single page output, because you can simply use the searching tool of your web browser to find text (e.g., press `Ctrl + F` or `Command + F`). You may also want to set the option `split_by` to `none` to only generate a single output page, in which case there will not be any navigation buttons, since there are no other pages to navigate to. You can still generate multiple-page HTML files if you like. Another option you may want to use is `self_contained = TRUE` when it is only a single output page.

4

Customization

As we mentioned in the very beginning of this book, you are expected to have some basic knowledge about R Markdown, and we have been focusing on introducing the **bookdown** features instead of **rmarkdown**. In fact, R Markdown is highly customizable, and there are many options that you can use to customize the output document. Depending on how much you want to customize the output, you may use some simple options in the YAML metadata, or just replace the entire Pandoc template.

4.1 YAML options

For most types of output formats, you can customize the syntax highlighting styles using the `highlight` option of the specific format. Currently, the possible styles are `default`, `tango`, `pygments`, `kate`, `monochrome`, `espresso`, `zenburn`, `haddock`, and `breezedark`. For example, you can choose the `tango` style for the `gitbook` format:

```
---
output:
```

```
  bookdown::gitbook:
    highlight: tango
---
```

For HTML output formats, you are most likely to use the `css` option to provide your own CSS stylesheets to customize the appearance of HTML elements. There is an option `includes` that applies to more formats, including HTML and LaTeX. The `includes` option allows you to insert arbitrary custom content before and/or after the body of the output. It has three sub-options: `in_header`, `before_body`, and `after_body`. You need to know the basic structure of an HTML or LaTeX document to understand these options. The source of an HTML document looks like this:

```
<html>

  <head>
  <!-- head content here, e.g. CSS and JS -->
  </head>

  <body>
  <!-- body content here -->
  </body>

</html>
```

The `in_header` option takes a file path and inserts it into the `<head>` tag.

The `before_body` file will be inserted right below the opening `<body>` tag, and `after_body` is inserted before the closing tag `</body>`.

A LaTeX source document has a similar structure:

```
\documentclass{book}

% LaTeX preamble
% insert in_header here

\begin{document}
% insert before_body here

% body content here

% insert after_body here
\end{document}
```

The `includes` option is very useful and flexible. For HTML output, it means you can insert arbitrary HTML code into the output. For example, when you have LaTeX math expressions rendered via the MathJax library in the HTML output, and want the equation numbers to be displayed on the left (default is on the right), you can create a text file that contains the following code:

```
<script type="text/x-mathjax-config">
MathJax.Hub.Config({
  TeX: { TagSide: "left" }
```

```
});
</script>
```

Let's assume the file is named `mathjax-number.html`, and it is in the root directory of your book (the directory that contains all your Rmd files). You can insert this file into the HTML head via the `in_header` option, e.g.,

```
---
output:
  bookdown::gitbook:
    includes:
      in_header: mathjax-number.html
---
```

Another example is to enable comments or discussions on your HTML pages. There are several possibilities, such as Disqus (https://disqus.com) or Hypothesis (https://hypothes.is). These services can be easily embedded in your HTML book via the `includes` option (see Section **??** for details).

Similarly, if you are familiar with LaTeX, you can add arbitrary LaTeX code to the preamble. That means you can use any LaTeX packages and set up any package options for your book. For example, this book used the `in_header` option to use a few more LaTeX packages like **booktabs** (for better-looking tables) and **longtable** (for tables that span across multiple pages), and applied a fix to an XeLaTeX problem that links on graphics do not work:

```
\usepackage{booktabs}
\usepackage{longtable}

\ifxetex
  \usepackage{letltxmacro}
  \setlength{\XeTeXLinkMargin}{1pt}
  \LetLtxMacro\SavedIncludeGraphics\includegraphics
  \def\includegraphics#1#{% #1 catches optional stuff (star/opt. arg.)
    \IncludeGraphicsAux{#1}%
  }%
  \newcommand*{\IncludeGraphicsAux}[2]{%
    \XeTeXLinkBox{%
      \SavedIncludeGraphics#1{#2}%
    }%
  }%
\fi
```

The above LaTeX code is saved in a file `preamble.tex`, and the YAML metadata looks like this:

```
---
output:
  bookdown::pdf_book:
    includes:
      in_header: preamble.tex
---
```

4.2 Theming

Sometimes you may want to change the overall theme of the output, and usually this can be done through the `in_header` option described in the previous section, or the `css` option if the output is HTML. Some output formats have their unique themes, such as `gitbook`, `tufte_html_book`, and `tufte_book2`, and you may not want to customize these themes too much. By comparison, the output formats `html_book()` and `pdf_book()` are not tied to particular themes and more customizable.

As mentioned in Section 3.1.2, the default style for `html_book()` is the Bootstrap style. The Bootstrap style actually has several built-in themes that you can use, including `default`, `cerulean`, `journal`, `flatly`, `darkly`, `readable`, `spacelab`, `united`, `cosmo`, `lumen`, `paper`, `sandstone`, `simplex`, and `yeti`. You can set the theme via the `theme` option, e.g.,

```
---
output:
  bookdown::html_book:
    theme: united
---
```

If you do not like any of these Bootstrap styles, you can set `theme` to `null`, and apply your own CSS through the `css` or `includes` option.

For `pdf_book()`, besides the `in_header` option mentioned in the previous section, another possibility is to change the document class. There are many possible LaTeX classes for

books, such as **memoir** (https://www.ctan.org/pkg/memoir), **amsbook** (https://www.ctan.org/pkg/amsbook), KOMA-Script (https://www.ctan.org/pkg/koma-script) and so on. Here is a brief sample of the YAML metadata specifying the `scrbook` class from the KOMA-Script package:

```
---
documentclass: scrbook
output:
  bookdown::pdf_book:
    template: null
---
```

Some publishers (e.g., Springer and Chapman & Hall/CRC) have their own LaTeX style or class files. You may try to change the `documentclass` option to use their document classes, although typically it is not as simple as that. You may end up using `in_header`, or even design a custom Pandoc LaTeX template to accommodate these document classes.

Note that when you change `documentclass`, you are likely to specify an additional Pandoc argument `--top-level-division=chapter` so that Pandoc knows the first-level headers should be treated as chapters instead of sections (this is the default when `documentclass` is `book`), e.g.,

```
documentclass: krantz
output:
  bookdown::pdf_book:
    pandoc_args: --top-level-division=chapter
```

4.3 Templates

When Pandoc converts Markdown to another output format, it uses a template under the hood. The template is a plain-text file that contains some variables of the form `$variable$`. These variables will be replaced by their values generated by Pandoc. Below is a very brief template for HTML output:

```
<html>
  <head>
    <title>$title$</title>
  </head>

  <body>
  $body$
  </body>
</html>
```

It has two variables `title` and `body`. The value of `title` comes from the `title` field of the YAML metadata, and `body` is the HTML code generated from the body of the Markdown input document. For example, suppose we have a Markdown document:

```
---
title: A Nice Book
---
```

```
# Introduction

This is a **nice** book!
```

If we use the above template to generate an HTML document, its source code will be like this:

```
<html>
  <head>
    <title>A Nice Book</title>
  </head>

  <body>

  <h1>Introduction</h1>

  <p>This is a <strong>nice</strong> book!</p>

  </body>
</html>
```

The actual HTML, LaTeX, and EPUB templates are more complicated, but the idea is the same. You need to know what variables are available: some variables are built-in Pandoc variables, and some can be either defined by users in the YAML metadata, or passed from the command-line option `-V` or `--variable`. Some variables only make sense in specific output formats, e.g., the `documentclass` variable is only used in LaTeX

output. Please see the documentation of Pandoc to learn more about these variables, and you can find all default Pandoc templates in the GitHub repository https://github.com/jgm/pandoc-templates.

Note that for HTML output, **bookdown** requires some additional comment tokens in the template, and we have explained them in Section 3.1.2.

4.4 Configuration

We have mentioned `rmd_files` in Section 1.3, and there are more (optional) settings you can configure for a book in `_bookdown.yml`:

- `book_filename`: the filename of the main Rmd file, i.e., the Rmd file that is merged from all chapters; by default, it is named `_main.Rmd`.
- `delete_merged_file`: whether to delete the main Rmd file after the book is successfully rendered.
- `before_chapter_script`: one or multiple R scripts to be executed before each chapter, e.g., you may want to clear the workspace before compiling each chapter, in which case you can use `rm(list = ls(all = TRUE))` in the R script.
- `after_chapter_script`: similar to `before_chapter_script`, and the R script is executed after each chapter.
- `edit`: a link that collaborators can click to edit the Rmd source document of the current page; this was designed primarily for GitHub repositories, since it is easy to edit arbitrary plain-text files on GitHub even in other people's repositories (if you do not have write access to the repository, GitHub will automatically fork it and let

you submit a pull request after you finish editing the file). This link should have `%s` in it, which will be substituted by the actual Rmd filename for each page.

- `history`: similar to `edit`, a link to the edit/commit history of the current page.
- `view`: similar to `edit`, a link to source code of the current page.
- `rmd_subdir`: whether to search for book source Rmd files in subdirectories (by default, only the root directory is searched). This may be either a boolean (e.g. `true` will search for book source Rmd files in the project directory and all subdirectories) or list of paths if you want to search for book source Rmd files in a subset of subdirectories.
- `output_dir`: the output directory of the book (`_book` by default); this setting is read and used by `render_book()`.
- `clean`: a vector of files and directories to be cleaned by the `clean_book()` function.

Here is a sample `_bookdown.yml`:

```
book_filename: "my-book.Rmd"
delete_merged_file: true
before_chapter_script: ["script1.R", "script2.R"]
after_chapter_script: "script3.R"
view: https://github.com/rstudio/bookdown-demo/blob/master/%s
edit: https://github.com/rstudio/bookdown-demo/edit/master/%s
output_dir: "book-output"
clean: ["my-book.bbl", "R-packages.bib"]
```

4.5 Internationalization

If the language of your book is not English, you will need to translate certain English words and phrases into your language, such as the words "Figure" and "Table" when figures/tables are automatically numbered in the HTML output. Internationalization may not be an issue for LaTeX output, since some LaTeX packages can automatically translate these terms into the local language, such as the **ctexcap** package for Chinese.

For non-LaTeX output, you can set the `language` field in the configuration file `_bookdown.yml`. Currently the default settings are:

```
language:
  label:
    fig: 'Figure '
    tab: 'Table '
    eq: 'Equation '
    thm: 'Theorem '
    lem: 'Lemma '
    cor: 'Corollary '
    prp: 'Proposition '
    cnj: 'Conjecture '
    def: 'Definition '
    exm: 'Example '
    exr: 'Exercise '
    hyp: 'Hypothesis '
```

```
    proof: 'Proof. '
    remark: 'Remark. '
    solution: 'Solution. '
  ui:
    edit: Edit
    chapter_name: ''
    appendix_name: ''
```

For example, if you want `FIGURE x.x` instead of `Figure x.x`, you can change `fig` to `"FIGURE "`:

```
language:
  label:
    fig: "FIGURE "
```

The fields under `ui` are used to specify some terms in the user interface. The `edit` field specifies the text associated with the `edit` link in `_bookdown.yml` (Section 4.4). The fields `chapter_name`, `appendix_name`, `fig`, `tab` and `eq` can be either a character string to be prepended to chapter (e.g., `'CHAPTER '`) or reference number (e.g., `'FIGURE '`), or an R function that takes a number (chapter or reference number) as the input and returns a string. (e.g., `!expr function(i) paste('Chapter', i)`). Here is an example for Hungarian:

```
language:
  label:
    fig: !expr function(i) paste(i, 'ábra')
```

```
ui:
  chapter_name: !expr function(i) paste0(i, '. fejezet')
```

For `chapter_name` and `appendix_name` only, if it is a character vector of length 2, the chapter title prefix will be `paste0(chapter_name[1], i, chapter_name[2])`, where `i` is the chapter number.

There is one caveat when you write in a language that uses multibyte characters, such as Chinese, Japanese, and Korean (CJK): Pandoc cannot generate identifiers from section headings that are pure CJK characters, so you will not be able to cross-reference sections (they do not have labels), unless you manually assign identifiers to them by appending `{#identifier}` to the section heading, where `identifier` is an identifier of your choice.

5

编辑

在本章中，我们将解释如何在本地编辑、构建、预览和通过 HTTP 服务预览书籍。你可以使用任何文本编辑器来编辑书籍，不过我们将展示一些使用 RStudio IDE 的技巧。在介绍编辑器之前，我们将介绍用于构建、预览和通过 HTTP 服务预览书籍的底层 R 函数，以便你真正了解在 RStudio IDE 中单击某个按钮时在幕后发生的事情，并且还可以自定义调用这些函数的其它编辑器。

5.1 构建书籍

要将所有 Rmd 文件构建到一本书中，可以在 **bookdown** 中调用 `render_book()`函数。下面是 `render_book()` 的参数：

```
render_book(input = ".", output_format = NULL, ...,
  clean = TRUE, envir = parent.frame(),
  clean_envir = !interactive(), output_dir = NULL,
  new_session = NA, preview = FALSE,
  config_file = "_bookdown.yml")
```

最重要的参数是 `output_format`，它可以接受表示输出格式的字符

串（例如 'bookdown::gitbook'）。你可以将此参数留空，默认输出格式是在第一个 Rmd 文件的 YAML 元数据中指定的第一个输出格式，或者在单独的 YAML 文件 _output.yml 中，如 4.4 部分所述。如果你计划为书籍生成多种输出格式，建议在 _output.yml 中指定所有格式。

一旦在 _output.yml 中指定了所有格式，就很容易编写一个 R 或 Shell 脚本或 Makefile 来编译书籍。下面是一个使用 Shell 脚本将书籍编译为 HTML（GitBook 样式）和 PDF 的简单示例：

```
#!/usr/bin/env Rscript

bookdown::render_book("index.Rmd", "bookdown::gitbook")
bookdown::render_book("index.Rmd", "bookdown::pdf_book")
```

Shell 脚本在 Windows 上不起作用（虽然严格来说并不是真的），但希望你能理解。

译者注：Windows 的 cmd 和 powershell 都无法运行 bash shell 脚本，但能够通过其它一些方式运行。

参数 ... 被传递给输出格式函数。参数 clean 和 envir 被传递给 rmarkdown::render()，分别决定是否清理中间文件以及指定运行 R 代码的环境。

书籍的输出目录可以通过 output_dir 参数指定。默认情况下，书籍生成到 _book 目录。这也可以通过配置文件 _bookdown.yml 中的 output_dir 字段进行更改，这样就不必多次指定它来将书籍编译为多种输

出格式。new_session 参数已在第 1.4 节中进行了解释。当设置 preview = TRUE 时，只编译 input 参数中指定的 Rmd 文件，这在预览某一章节时很方便，因为你不重新编译整本书，但在发布图书时，该参数肯定应该设置为 FALSE。

render_book() 将生成许多输出文件。有时你可能需要清除书籍输出目录并重新开始生成书籍。例如，删除由 **knitr** 自动生成的图形和缓存文件。函数 clean_book() 就是为此而设计的。默认情况下，它告诉你可以删除哪些输出文件。如果你查看了它输出的可删除文件列表，并且确定没有文件被错误地标识为输出文件（你当然不想删除手动创建的输入文件），则可以使用 bookdown::clean_book(TRUE) 删除所有输出文件。由于删除文件是一个相对危险的操作，我们建议你通过版本控制工具（如 GIT）或支持备份和还原的服务来维护你的书籍，这样，如果你错误地删除某些文件，文件将不会永远丢失。

5.2 预览单个章节

当书籍项目的大小很大时，构建整本书可能会很慢。有两件事会影响一本书的构建速度：R 代码块的计算以及使用 Pandoc 将 Markdown 转换为其他格式的过程。前者可以通过使用 chunk 选项 cache = TRUE 在 **knitr** 中启用缓存来改进，但创作者并没有什么办法能使后者更快。不过，你可以选择使用 **bookdown** 中的 preview_chapter() 函数一次只编译一个章节，通常这比编译整本书要快得多。只有传递到 preview_chapter() 的 Rmd 文件才会被编译。

当你只关注当前章节时，预览当前章节会很有帮助，因为可以在添加更多内容或修改章节时立即看到实际输出。尽管预览功能适用于所有输出格式，但我们建议你预览 HTML 输出。

预览单个章节的一个缺点是，对其他章节的交叉引用将不起作用，因为在这种情况下，**bookdown** 对其他章节一无所知。这对于速度的提高来说是一个相当小的代价。由于预览章节只会呈现该特定章节的输出，因此不应期望其他章节的内容也能正确呈现。例如，当你导航到其他章节时，实际上你正在查看该章节的旧输出（甚至可能不存在该章节的输出）。

5.3 使用 HTTP 服务预览书籍

相比于重复运行 `render_book()` 或 `preview_chapter()` 来预览章节，你实际上可以在 web 浏览器中实时预览书籍，你只需要保存 Rmd 文件即可。**bookdown** 中的函数 `serve_book()` 可以基于 **servr** 软件包 (Xie, 2020) 启动本地 web 服务器，提供 HTML 输出的在线预览服务。

```
serve_book(dir = ".", output_dir = "_book",
  preview = TRUE, in_session = TRUE, quiet = FALSE,
  ...)
```

将书籍的根目录传递给 `dir` 参数，上述函数将启动本地 web 服务器，以便你可以使用服务器查看书籍输出。访问书籍输出页面的默认 URL 是 `http://127.0.0.1:4321`。如果在交互式 R session 中运行此功能，此 URL 将自动在 web 浏览器中打开。如果你在 RStudio IDE 中，RStudio viewer 将用作默认的 web 浏览器，因此你可以在相同的环境中编写 Rmd 源文件并预览输出（例如，在左侧编写源文件，在右侧查看输出文件）。

服务器将侦听书籍根目录中的更改：每当修改书籍目录中的任

何文件，serve_book() 都可以检测到更改，重新编译对应的 Rmd 文件，并自动刷新 web 浏览器。如果修改的文件不包括 Rmd 文件，它只会刷新浏览器（例如只更新了某个 CSS 文件）。这意味着一旦启动了服务器，接下来所要做的就是编写书籍并保存文件。在保存文件时，编译和预览将自动进行。

如果真的不需要太多时间来重新编译整本书，可以设置参数 preview = FALSE，这样每次更新这本书时，整本书都会重新编译，否则只有修改过的章节会通过 preview_chapter() 重新编译。

... 里的参数都会传递给 servr::httw()，请参阅其帮助页面以查看所有可能的选项，例如 daemon 和 port。使用 in_session = TRUE 或 FALSE 有其优缺点：

- 对于 in_session = TRUE，你可以在当前 R session 中访问在书籍中创建的所有对象；如果使用守护进程（通过参数 daemon = TRUE），你可以在当前 R session 不忙碌时检查对象；否则必须先停止服务器，然后才能检查对象。这当你需要以交互方式探索书中的 R 对象时会很有用。in_session = TRUE 的缺点是输出内容可能与从新的 R session 编译的书不同，因为当前 R session 的状态可能不干净。
- 对于 in_session = FALSE，你不能从当前 R session 访问书籍中的对象，但其输出内容更有可能是可复制的，因为所有内容都是从新的 R session 中创建的。由于此函数仅用于预览目的，因此 R session 是否干净可能不是一个大问题。

根据具体的用例，你可以选择 in_session = TRUE 或 FALSE。最后，你应该从一个新的 R session 开始运行 render_book() 以生成一个可靠的书籍副本。

5.4 RStudio IDE

如果你的 RStudio IDE 版本低于 1.0.0，我们建议你进行升级[1]。正如第 1.3 节所述，所有 R Markdown 文件都必须用 UTF-8 编码。这一点非常重要，尤其是当文件包含多字节字符时。要使用 UTF-8 编码保存文件，可以使用菜单 File -> Save with Encoding，然后选择 UTF-8。

当你单击 Knit 按钮在 RStudio IDE 中编译 R Markdown 文档时，RStudio 调用的默认函数是 rmarkdown::render()，这不是我们想要的编译书籍的函数。要调用函数 bookdown::render_book()，可以在 R Markdown 文档 index.Rmd 的 YAML 元数据中将 site 字段设置为 bookdown::bookdown_site，例如：

```
---
title: "A Nice Book"
site: bookdown::bookdown_site
output:
  bookdown::gitbook: default
---
```

当你在 index.Rmd 中设置 site: bookdown::bookdown_site 后，RStudio 将能够发现作为书籍源目录的目录，[2]，并且你将在 Build 窗格中看到 Build book 按钮。你可以通过单击按钮以不同的格式构建整本书，另外，如果单击工具栏上的 Knit 按钮，RStudio 将自动预览当前章节，而不需要显式使用 preview_chapter()。

[1] https://www.rstudio.com/products/rstudio/download/

[2] 这个目录必须是一个 RStudio 项目。

bookdown 软件包附带了几个 RStudio 插件。如果你不熟悉 RStudio 插件，可以在以下位置查看文档：http://rstudio.github.io/rstudioaddins/。当你安装了 **bookdown** 软件包并正在使用 RStudio v0.99.878 或更高版本时，打开菜单，你将在工具栏上看到名为“Addins”的下拉菜单和 “Preview Book” 以及 “Input LaTeX Math” 菜单项。

“Preview Book” 插件调用 `bookdown::serve_Book()` 来编译书籍并通过 HTTP 服务提供书籍预览。它将阻塞当前的 R session，即当 `serve_book` 正在运行时，你将无法在 R console 中执行任何操作。为了避免阻塞 R session，你可以使用 `bookdown::serve_book(daemon = TRUE)` 在服务器上启动守护进程。请注意，在 RStudio 中打开的当前文档位于书籍的根目录下时，才能够使用这个加载项，否则 `serve_book()` 可能无法找到书籍源文档。

“Input LaTeX Math” 插件本质上是一个小的 Shiny 应用程序，它提供一个文本框来帮助输入 LaTeX 数学表达式（图 5.1）。输入表达式时，你将可以预览数学表达式并看到其源代码。这将使输入数学表达式时更不容易出错——当在没有预览的情况下输入一个长的 LaTeX 数学表达式时很容易犯错误，例如，输入 `X_ij` 时，可能想要输入的是 `X_{ij}`，或者输入时忽略了右括号。如果在单击加载项之前在 RStudio 编辑器中选择了 LaTeX 数学表达式，则该表达式将自动加载并呈现在文本框中。这个插件是建立在 MathQuill 库 (http://mathquill.com) 之上的。它并不是要为所有用于数学表达式的 LaTeX 命令提供完整支持，而是致力于帮助输入一些常见的数学表达式。

还有其他 R 软件包提供了插件来帮助你编写书籍。**citr** 软件包 (Aust, 2019) 提供了一个名为 “Insert citations” 的加载项，可以很容易地将引用文献插入到 R Markdown 文档中。它会扫描你的文献数据库，并在下拉菜单中显示所有引用文献，因此你可以直接从列表中选择，而无需记住哪个引文键对应于哪个引文项（图 5.2）。

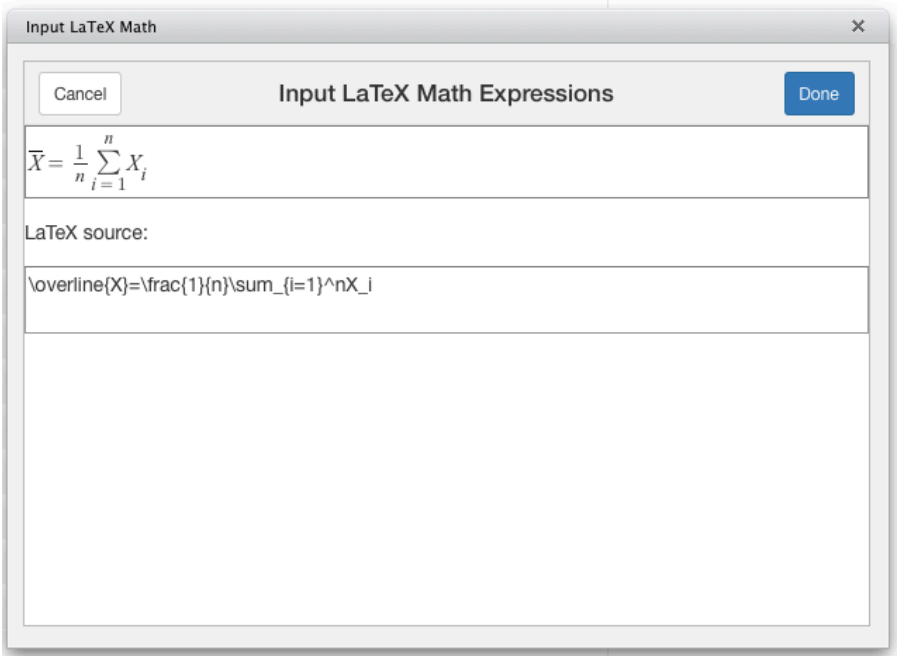


图 **5.1:** 辅助输入 LaTeX 数学公式的 RStudio 插件

5.5 协同工作

今后创作书籍几乎肯定会涉及到多个人。你可能会和共同作者一起工作，以及不时提供反馈的读者。

由于所有书籍章节都是纯文本文件，因此它们非常适合用于应用版本控制工具。因此，如果你的所有合著者和合作者都具备 GIT 等版本控制工具的基本知识，你就可以使用这些工具与他们协作处理书籍内容。事实上，即使他们不知道如何使用 GIT，通过 GIT 进行协作也是可以的，因为 GitHub能够让使用者在 web 浏览器中在线创建和编辑文件。只需要有一个人必须熟悉 GIT，并且能够设置书籍存储库。

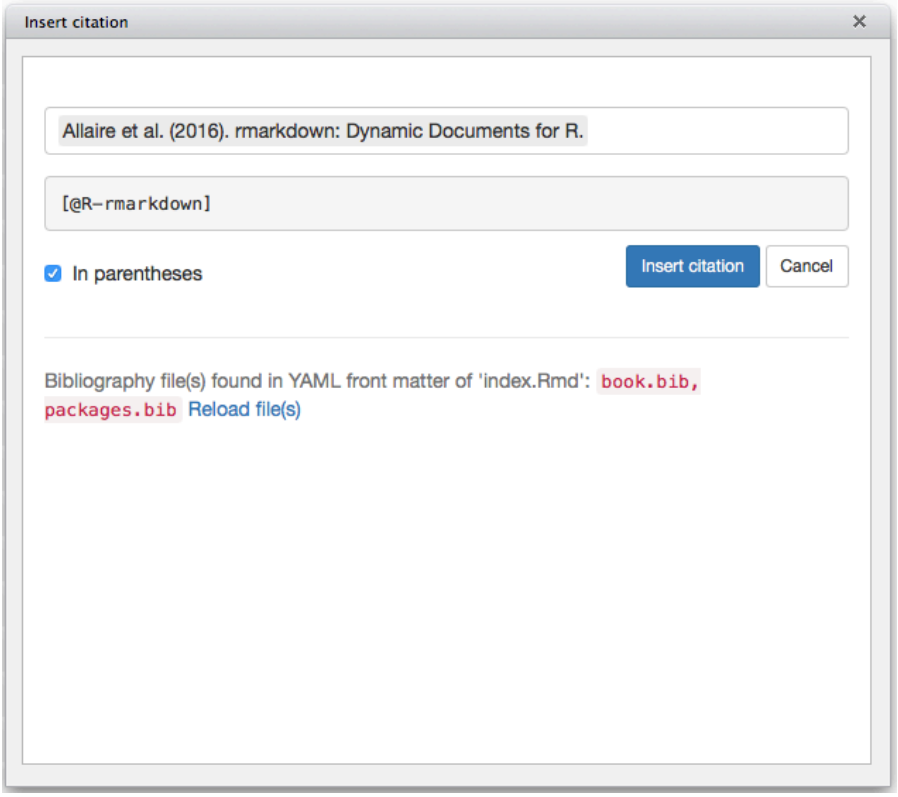


图 **5.2:** 帮助插入引用文献的 RStudio 插件

其他合作者可以在线贡献内容，不过如果他们知道在本地使用 GIT 工作的基础用法，他们会有更多的自由。

读者可以通过两种方式做出贡献。一种方式是直接贡献内容，如果你的书源托管在 GitHub 上，最简单的方法是通过 GitHub pull requests[3]。基本上，任何 GitHub 用户都可以单击 Rmd 源文件页面上的编辑 (edit) 按钮，编辑内容，并将更改提交给你进行审批。如果你对提交的更改感到满意（你可以清楚地看到更改的内容），可以单击“合并 (Merge)”按钮来合并更改。如果你不满意，可以在 pull request 中提供反馈，以便读者根据要求进一步修改。我们在第 **??** 节中提到了

[3] https://help.github.com/articles/about-pull-requests/

GitBook 样式中的编辑 (edit) 按钮。该按钮链接到每个页面的 Rmd 源文件，可以引导你创建 pull request。没有必要来回收发电子邮件来交流简单的更改内容，比如修改打字错误。

读者对你的书作出贡献的另一种方式是留下评论。评论可以以多种形式留下：电子邮件、GitHub issues 或 HTML 页面评论。这里我们使用 Disqus（参见第 4.1 节）作为示例。Disqus 是一种在网页上嵌入讨论区的服务，可以通过 JavaScript 加载。当你注册并在 Disqus 上创建一个新的论坛后，你可以找到如下所示的 JavaScript 代码：

```
<div id="disqus_thread"></div>
<script>
(function() { // DON'T EDIT BELOW THIS LINE
var d = document, s = d.createElement('script');
s.src = '//yihui.disqus.com/embed.js';
s.setAttribute('data-timestamp', +new Date());
(d.head || d.body).appendChild(s);
})();
</script>
<noscript>Please enable JavaScript to view the
<a href="https://disqus.com/?ref_noscript">
  comments powered by Disqus.</a></noscript>
```

请注意，你需要用自己的论坛名称替换名称 `yihui`（创建新论坛时必须提供此名称）。你可以将代码保存到一个名为 `disqus.html` 的 HTML 文件中。然后可以通过 `after_body` 选项将其嵌入到每页的末尾（图 5.3 显示了讨论区的外观）：

```
---
output:
  bookdown::gitbook:
    includes:
      after_body: disqus.html
---
```

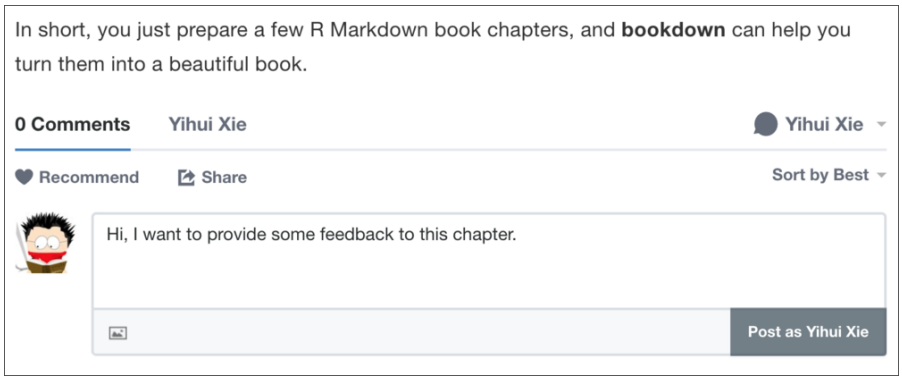


图 **5.3:** 有讨论区的书页。

6

发布与出版

当你在创作书籍时，你可以将书稿提供给公众，例如，将书稿发布到网站上，以便从读者那里得到早期反馈。当你完成书籍的创作后，你需要考虑正式出版书籍的方式，可以通过印刷本也可以通过电子书进行出版。

6.1 RStudio Connect

理论上，你可以自己编译书籍然后将其发布到你想要的任何地方。例如，你可以在自己的 Web 服务器上托管书籍的 HTML 版本。不过我们在 **bookdown** 中提供了一个函数 `publish_book()`，它能够让你很轻松地将书籍上传至 https://bookdown.org。这是一个由 RStudio 提供的网站，用于免费托管你的书籍。这个网站建立在“RStudio Connect”，[1] 之上，它是 RStudio 提供的产品之一，能够让你将各种与 R 相关的应用部署到服务器上，包括 R Markdown 文档、Shiny 应用、R plots 等。

你不必了解太多 RStudio Connect 就能够将你的书籍发布到 bookdown.org。你需要在 https://bookdown.org/connect/ 注册，之后当你第一次尝试运行 `bookdown::publish_book()` 时，系统将要求你授权 **bookdown** 发布到你的 bookdown.org 账户。以后使用时，只需要再

[1] https://www.rstudio.com/products/connect/

次调用 `publish_book()` 即可，**bookdown** 将不会要求你进行任何其他的操作。

```
publish_book(name = NULL, account = NULL,
  server = NULL, render = c("none", "local", "server"))
```

你需要接触的 `publish_book()` 的唯一参数是 `render`。它决定了在发布之前是否编译书籍。如果你之前已经运行过 `render_book()`，就不需要改变这个参数，否则你可能需要将其设置为 `'local'`：

```
bookdown::publish_book(render = 'local')
```

如果你已经配置好了自己的 RStudio Connect 服务器，那么当然可以将书籍发布到你自己的服务器，而不必上传至 bookdown.org。

6.2 GitHub

你可以使用 GitHub Pages (https://pages.github.com) 在 GitHub 上免费托管你的书籍。GitHub 支持 Jekyll (http://jekyllrb.com)，它是一个静态网站生成器，能够将一系列 Markdown 文件转换为网站。这可能是 GitHub Pages 最常见的用法，但是 GitHub 还支持任意静态 HTML 文件，因此你可以在 GitHub 上托管书籍的 HTML 输出文件。其关键是创建一个隐藏文件 `.nojekyll`，它告诉 GitHub 你的网站不是通过 Jekyll 构建的，因为 **bookdown** 的 HTML 输出文件已经是一个独立的网站。

```
# 假设你已经初始化了一个 git 储存库，并且已经在书籍储存库的目录下

# 创建一个隐藏文件 .nojekyll
touch .nojekyll
# 将其添加入 git 版本控制中，这样她就不会再 RStudio 中显示
git add .nojekyll
```

如果你使用 Windows，那么可能没有 touch 命令，这时你可以在 R 中使用 file.create('.nojekyll') 来创建一个文件。

发布书籍的一种方法将书籍的 HTML 文件放入 master 分支中的 /docs 文件夹，然后从该文件夹将书籍作为 GitHub Pagse 站点发布，就像 GitHub Help[2] 中描述的那样。首先，在配置文件 _bookdown.yml 中添加一行 output_dir:"docs"，将书籍的输出目录设置为 /docs。然后，将更改推送到 GitHub，再转到存储库的设置，在 “GitHub Pages” 配置项下将 “Source” 选项更改为 “master branch/docs folder”。使用该方法时，.nojekyll 文件必须位于 /docs 文件夹中。

另一种方法是先在存储库中创建一个 gh-pages 分支，再构建书籍，将 HTML 输出（包括图像、CSS 和 JavaScript 文件等所有外部资源）放入该分支，然后将该分支推送到远程存储库。如果你的书籍存储库没有 gh-pages 分支，可以使用以下命令创建一个分支：

```
# 假设你已经初始化了一个 git 储存库，并且已经在书籍储存库的目录下

# 创建一个名为 gh-pages 的分支，并清除全部文件
git checkout --orphan gh-pages
```

[2] http://bit.ly/2cvloKV

```
git rm -rf .

# 创建一个隐藏文件 .nojekyll
touch .nojekyll
git add .nojekyll

git commit -m "Initial commit"
git push origin gh-pages
```

设置好GIT之后，剩下的工作可以通过脚本（Shell、R或Makefile，取决于你的偏好）实现自动化。总的来说，首先将书籍编译为HTML，然后运行git命令将文件推送到GitHub，但你可能不希望在本地反复手动执行这些操作，这时可以通过云来完成。由于在云上实现发布过程的完全自动化将会非常方便，因此一旦设置正确，接下来所要做的就是编写书籍并将Rmd源文件推送到GitHub，你创作的书将始终在服务器端自动构建和发布。

你可以选择使用的一个云服务是Travis CI (https://travis-ci.com)。它对于GitHub上的共有储存库提供的服务是免费的，并且是为软件包的持续集成(CI)而设计的。Travis CI能够连接到GitHub，即每当你推送更改到GitHub时，Travis能够被触发在最新版本的储存库上运行某些命令/脚本。[3]这些命令储存在你的储存库根目录下名为`.travis.yml`的YAML文件中。它们通常用于测试软件，但实际上它们的用途十分开放，这意味着你可以在Travis（虚拟）机器上运行任意命令。也就是说，你当然可以在Travis上运行自己的脚本来构建书籍。注意，Travis目前仅支持Ubuntu和Mac OS X，因此你应该对于Linux/Unix命令有一些基本的了解。

[3]你需要先在GitHub上为你的储存库授权Travis CI服务。有关如何开始使用Travis CI，请参阅https://docs.travis-ci.com/user/getting-started/。

下一个问题是，怎样将在 Travis 中构建的书籍发布到 GitHub？大体上说，你需要授予 Travis 对你的 GitHub 储存库的写访问权限。该授权能够通过集中方式完成，对于初学者来说最简单的方式是个人访问令牌 (personal access token)。以下是你可以遵循的几个操作步骤：

1. 对于你在 GitHub 上的账户创建一个个人访问令牌 (personal access token)[4]（请确保启用 “repo” 作用域 (“repo” scope)，以便可以通过此令牌写入你的 GitHub 储存库）。
2. 通过命令 `travis encrypt` 将其加密并放在环境变量 `GITHUB_PAT` 里，然后储存在 `.travis.yml` 文件中。例如 `travis encrypt GITHUB_PAT=TOKEN`。如果你不知道如何安装或使用 Travis 命令行工具，只需要将这个环境变量通过 https://travis-ci.com/user/repo/settings 储存起来，其中 `user` 是你的 GitHub ID，`repo` 是储存库的名称。
3. 你可以使用你的 GitHub 令牌在 Travis 上克隆先前创建的这个 `gh-pages` 分支，向其添加从 R Markdown 转换而来的 HTML 输出文件（不要忘记添加图片和 CSS 样式文件），然后推送到远程储存库。

假设你正在 `master` 分支中（你存放 Rmd 源文件的分支），并且已经编译好书籍，放在 `_book` 目录中。接下来你可以在 Travis 中做的是：

```
# 如果你还没有进行配置的话，设置好你的用户名和邮箱
git config --global user.email "you@example.com"
git config --global user.name "Your Name"

# 将储存库克隆到书籍输出目录 book-output
```

[4] http://bit.ly/2cEBYWB

```
git clone -b gh-pages \
  https://${GITHUB_PAT}@github.com/${TRAVIS_REPO_SLUG}.git \
  book-output
cd book-output
git rm -rf *
cp -r ../_book/* ./
git add --all *
git commit -m"Update the book"
git push -q origin gh-pages
```

变量名 GITHUB_PAT 和目录名 book-output 可以是任意名称。只要名称没有与已经存在的变量名或目录名冲突，你就可以使用喜欢的任何名字。上述脚本与我们在第 ?? 节提到的书籍构建脚本可以放在 master 分支作为 as Shell 脚本。例如，你可以将它们命名为 _build.sh 和 _deploy.sh。那么，你的 .travis.yml 文件可能是这样的：

```
language: r
pandoc_version: 1.19.2.1

env:
  global:
    - secure: A_LONG_ENCRYPTED_STRING

before_script:
  - chmod +x ./_build.sh
  - chmod +x ./_deploy.sh
```

```
script:
  - ./_build.sh
  - ./_deploy.sh
```

language 选项告诉 Travis 需要使用安装了 R 的虚拟机。secure 字段是加密的个人访问令牌 (personal access token)。如果你已经使用 Travis 上的 Web 界面而不是命令行工具 travis encrypt 保存了 GITHUB_PAT 变量，则可以忽略这项设置。

由于 Travis 服务主要用于检查 R 软件包，因此还需要一个（假的）DESCRIPTION 文件，使得书籍存储库像是一个 R 软件包一样。这个文件中唯一一个真正重要的是软件包依赖项这一配置。所有依赖项都将通过 **devtools** 包安装。如果依赖项在 CRAN 或 BioConductor 上，只需在 DESCRIPTION 文件的 Imports 字段中列出即可。如果它在 GitHub 上，您可以使用 Remotes 字段列出它的存储库名称。下面展示了一个例子：

```
Package: placeholder
Type: Book
Title: Does not matter.
Version: 0.0.1
Imports: bookdown, ggplot2
Remotes: rstudio/bookdown
```

如果你使用 Travis 的 container-based infrastructure[5]，你可以在 .travis.yml 中使用 sudo: false 启用缓存。通常你至少需要缓存两类目录：图片目录（例如 _main_files）以及缓存目录（例如 _main_cache）。如果你指定了 **knitr** 代码块选项 fig.path 和 cache.path，这些目录的名

[5] https://docs.travis-ci.com/user/workers/container-based-infrastructure/

称可能不同，但是我强烈建议不要改变这些设置。图片和缓存目录都存放在书籍根目录中的 _bookdown_files 目录下。启用了 **knitr** 图片和缓存的 .travis.yml 文件可能具有如下的 sudo 和 cache 附加配置：

```
sudo: false

cache:
  packages: yes
  directories:
    - $TRAVIS_BUILD_DIR/_bookdown_files
```

如果构建书籍非常耗时，你可以在 Travis 上使用上面的配置来节省时间。注意，packages:yes 表示安装在 Travis 上的 R 包也被缓存。

以上所有脚本和配置都可以在 bookdown-demo 存储库中找到：https://github.com/rstudio/bookdown-demo/。如果你将它们复制到自己的存储库中，请记住使用自己的加密变量 GITHUB_PAT 更改 .travis.yml 文件中的 secure 字段。

GitHub 和 Travis CI 当然不是构建和出版你的书籍的唯一选择。你可以在自己的服务器上自由地存储和发布这本书。

6.3 出版商

除了在网上发布你的书之外，你还可以考虑通过出版商出版你的书籍。例如，本书是由 Chapman & Hall/CRC 出版的，在 https://bookdown.org/yihui/bookdown/ 也有免费的在线版本（与出版商达成了协议）。如果你不想与的发布者合作，你还可以考虑自主出版 (https:

//en.wikipedia.org/wiki/Self-publishing)。Pablo Casas 写了两篇你可能会觉得有用的博客文章："How to self-publish a book"[6] 和 "How to self-publish a book: customizing bookdown"[7]。

如果你选择的出版商支持 LaTeX，那么出版用 **bookdown** 编写的书会容易得多。例如，Chapman & Hall 提供了一个名为 `krantz.cls` 的 LaTeX 类，Springer 提供的是 `svmono.cls`。如果要将这些 LaTeX 类应用于 PDF 书籍，请将 `index.Rmd` 的 YAML 元数据中的 `documentclass` 设置为 LaTeX 类文件名（不带扩展名 `.cls`）。

LaTeX 类是 YAML 元数据中最重要的设置。它控制了 PDF 书籍的整体样式。还有一些其他设置是你经常需要调整的，下面我们将展示有关本书的一些详细信息。

本书的 YAML 元数据包含以下设置：

```
documentclass: krantz
lot: yes
lof: yes
fontsize: 12pt
monofont: "Source Code Pro"
monofontoptions: "Scale=0.7"
```

字段 `lot:yes` 表示我们需要表格列表；类似地，`lof` 表示图片列表。基础字体大小是'12pt'，我们使用了 Source Code Pro[8] 作为等宽（固定宽度）字体，它适用于本书中的所有程序代码。

[6] https://blog.datascienceheroes.com/how-to-self-publish-a-book/

[7] https://blog.datascienceheroes.com/how-to-self-publish-a-book-customizing-bookdown/

[8] https://www.fontsquirrel.com/fonts/source-code-pro

在 LaTeX 导言 (preamble)（第 4.1 节）中，我们还有一些设置。首先，我们将主字体族设置为 Alegreya[9]，并且由于此字体没有 Small Capitals（小型大写字母）特征，我们使用 Alegreya SC 字体。

```
\setmainfont[
  UprightFeatures={SmallCapsFont=AlegreyaSC-Regular}
]{Alegreya}
```

下面的命令通过允许浮动环境占用更大部分的页面而不是浮动，从而使得它们更不太可能浮动。

```
\renewcommand{\textfraction}{0.05}
\renewcommand{\topfraction}{0.8}
\renewcommand{\bottomfraction}{0.8}
\renewcommand{\floatpagefraction}{0.75}
```

由于 krantz.cls 为引用文段提供了一个环境 VF，因此我们将标准的 quote 环境重新定义为 VF。您可以在第 2.1 节中看到它的样式。

```
\renewenvironment{quote}{\begin{VF}}{\end{VF}}
```

然后我们将超链接重新定义为脚注，因为当书印刷在纸上时，读者无法点击文本中的链接，而脚注会告诉他们实际的链接是什么。

[9] https://www.fontsquirrel.com/fonts/alegreya

```
\let\oldhref\href
\renewcommand{\href}[2]{#2\footnote{\url{#1}}}
```

我们还为 `_output.yml` 中的 `bookdown::pdf_book` 格式进行了一些设置：

```
bookdown::pdf_book:
  includes:
    in_header: latex/preamble.tex
    before_body: latex/before_body.tex
    after_body: latex/after_body.tex
  keep_tex: yes
  dev: "cairo_pdf"
  latex_engine: xelatex
  citation_package: natbib
  template: null
  pandoc_args: --top-level-division=chapter
  toc_unnumbered: no
  toc_appendix: yes
  quote_footer: ["\\VA{", "}{}"]
  highlight_bw: yes
```

我们上面提到的所有导言 (preamble) 设置都在文件 `latex/preamble.tex` 中，其中我们还指定了前言 (front matter) 的开始：

译者注：\frontmatter 通常跟在 \begin{document} 后，会关闭章节序号，页码使用罗马数字。

```
\frontmatter
```

在 latex/before_body.tex 中，我们插入了出版商要求的一些空白页，并编写了奉献页。在书的第一章之前，我们插入

```
\mainmatter
```

因此，LaTeX 知道将页码样式从罗马数字（前言所用的样式）更改为阿拉伯数字（正文所用的样式）。

我们在 latex/after_body.tex（第 2.9 节）中打印索引。

由于默认设备 pdf 不能嵌入字体，因此用于保存图片的图形设备 (dev) 被设置为 cairo_pdf，以便字体可以嵌入图片中。你的文案编辑可能会要求您嵌入 PDF 中使用的所有字体，以便该书可以完全按其电子版本的外观打印，否则某些字体可能会被替换，印刷时的字型可能无法预测。

quote_footer 字段是为了确保引用页脚右对齐：krantz.cls 提供了 LaTeX 命令 \VA{} 以包含引用页脚。

highlight_bw 选项被设置为 true，这样语法高亮显示的代码块中的颜色将转换为灰度，因为这本书将采用黑白打印。

这本书是通过 xelatex 编译成 PDF 的，以便于我们使用自定义字体。

除 VF 环境和 \VA{} 命令外，上述所有设置都可以应用于任何其他 LaTeX 文档类。

如果你也想与 Chapman & Hall 合作，你可以从我们存储库 (https://github.com/rstudio/bookdown/tree/master/inst/examples) 中的 krantz.cls 文件开始，而不使用你从编辑那里得到的副本。我们已经与 LaTeX 帮助中心合作解决了这个 LaTeX 类的许多问题，所以如果你使用 **bookdown**，希望它能很好地用于你的书。

A

软件工具

对于那些不熟悉使用 R Markdown 所需的软件包的读者，我们将简要介绍这些软件包的安装和维护。

A.1 R 和 R 软件包

R 能够从任何一个 CRAN (the Comprehensive R Archive Network) 镜像站中下载和安装，例如 https://cran.rstudio.com。请注意每年都会有一些 R 的新版本发布，你可能需要偶尔升级 R。

为了安装 **bookdown** 阮家堡，你可以在 R 中输入：

```
install.packages("bookdown")
```

这将安装所有必需的 R 软件包。如果你不太关心这些软件包是否实际用于编译你的书籍（例如 **htmlwidgets**），也可以选择安装所有可选的软件包：

```
install.packages("bookdown", dependencies = TRUE)
```

如果想体验 GitHub 上 **bookdown** 的开发版本，需要首先安装 **devtools**：

```
if (!requireNamespace('devtools')) install.packages('devtools')
devtools::install_github('rstudio/bookdown')
```

R 软件包同样也经常在 CRAN 或 GitHub 上不断更新，因此你可能需要偶尔地更新它们：

```
update.packages(ask = FALSE)
```

尽管这不是必须的，但当你在处理与 R 相关的项目时，RStudio IDE 能够使很多事情变得更加简单。RStudio IDE 可以从 https://www.rstudio.com 下载。

A.2 Pandoc

R Markdown 文档 (`*.Rmd`) 首先通过 **knitr** 软件包编译成 Markdown (`*.md`)，然后通过 Pandoc 将 Markdown 编译成其他输出格式 (如 LaTeX 或 HTML)。这个过程由 **rmarkdown** 软件包自动完成。你不需要单独安装 **knitr** 或 **rmarkdown**，因为它们是 **bookdown** 的必需软件包，安装 **bookdown** 时会自动安装。但是，Pandoc 不是 R 软件包，因此在安装 **bookdown** 时不会自动安装。你可以按照 Pandoc 主页 (http://pandoc.org) 上的安装说明安装 Pandoc，但是如果你使用 RStudio IDE，实际上不需要单独安装 Pandoc，因为 RStudio 包含一个 Pandoc 的副本。Pandoc 版本号可通过以下方式获得：

```
rmarkdown::pandoc_version()
## [1] '2.14.0.3'
```

如果你发现这个版本太低了，并且一些 Pandoc 功能特性只在更高版本中提供，你可以安装更高版本的 Pandoc，之后 **rmarkdown** 将会调用更高版本的 Pandoc，而不是内置的版本。

A.3 LaTeX

只有当你想要将你的书籍转为 PDF 时，你才需要 LaTeX。你可以查阅 https://www.latex-project.org/get/ 以获取更多关于 LaTeX 它的安装的信息，但是我们强烈推荐你安装名为 TinyTeX[1] 的轻量级跨平台 LaTeX 发行版，它是基于 TeX Live 构建的。TinyTeX 能够通过 R 软件包 **tinytex** 轻松安装（安装 **bookdown** 时将自动安装）：

```
tinytex::install_tinytex()
```

使用 TinyTeX，你将永远不会看见这样的错误信息：

```
! LaTeX Error: File `titling.sty' not found.

Type X to quit or <RETURN> to proceed,
or enter new name. (Default extension: sty)
```

[1] https://yihui.org/tinytex/

```
Enter file name:
! Emergency stop.
<read *>

l.107 ^^M

pandoc: Error producing PDF
Error: pandoc document conversion failed with error 43
Execution halted
```

上面的错误信息表示你使用了一个包含 `titling.sty` 的 LaTeX 软件包，但它并没有被安装。LaTeX 软件包名称通常是 `*.sty` 这样的文件名格式，因此在本例中，你可以尝试安装 `titling` 软件包。如果你使用带有 R Markdown 的 TinyTeX，丢失的 LaTeX 软件包将自动安装，因此你无需担心此类问题。

LaTeX 发行版和其软件包也不时会进行更新，你可以考虑更新它们，特别是当您遇到 LaTeX 问题时。你可以通过以下方式找到 LaTeX 发行版的版本：

```
system('pdflatex --version')
## pdfTeX 3.141592653-2.6-1.40.23 (TeX Live 2021/W32TeX)
## kpathsea version 6.3.3
## Copyright 2021 Han The Thanh (pdfTeX) et al.
## There is NO warranty.  Redistribution of this software is
## covered by the terms of both the pdfTeX copyright and
## the Lesser GNU General Public License.
```

```
## For more information about these matters, see the file
## named COPYING and the pdfTeX source.
## Primary author of pdfTeX: Han The Thanh (pdfTeX) et al.
## Compiled with libpng 1.6.37; using libpng 1.6.37
## Compiled with zlib 1.2.11; using zlib 1.2.11
## Compiled with xpdf version 4.03
```

你可以运行如下代码来更新 TinyTeX：

```
tinytex::tlmgr_update()
```

随着时间的推移，你可能也需要升级 TinyTeX（否则你无法安装或更新任何 LaTeX 软件包），在这种情况下你需要重新安装 TinyTeX：

```
tinytex::reinstall_tinytex()
```

B

软件使用

如第 1 章所述，这本书并不是一本全面的 **knitr** 或 **rmarkdown** 指南。在本章中，我们简要地解释了 **knitr** 和 **rmarkdown** 中的一些基本概念和语法。如果你还有什么问题，可以将它们发布到 StackOverflow (https://stackoverflow.com) 上，并用 r、knitr、rmarkdown 和/或 bookdown 等任何适合的标签标记你的问题。

B.1 knitr

knitr 软件包是基于“文学编程”(Knuth, 1984) 的思想设计的，它允许你将程序代码与源文档中的文本混合在一起。当 **knitr** 编译文档时，将提取并执行程序代码（以代码块为单位），程序输出将与输出文档中的原始文本一起显示。我们在第 2.3 节中介绍了基本的语法。

R Markdown 不是 **knitr** 支持的唯一源格式。**knitr** 的基本思想可应用于其他计算和创作语言。例如，**knitr** 还支持 R 和 LaTeX 的组合（`*.rnw` 文档），以及 R + HTML (`*.RtML`) 等。你也可以在 **knitr** 中使用其他计算语言，如 C++、Python、SQL 等。下面是一个简单的例子，你可以在 http://rmarkdown.rstudio.com/authoring_knitr_engines.html 中了解更多信息。

````
```{python}
x = 'Hello, Python World!'
print(x.split(' '))
```
````

Python 用户可能熟悉 IPython 或 Jupyter Notebooks (https://jupyter.org)。事实上，R Markdown 也可以作为笔记本使用，并有一些额外的优势；有关这方面详细信息，请参阅这篇博客文章：https://blog.rstudio.org/2016/10/05/r-notebooks/。

如果要在文档中显示文本形式的代码块，可以在块头部之前添加一个内联表达式，该表达式生成一个空字符串（`r ''`），并将代码块用在四个反引号包裹起来，[1]例如：

````
````
`r ''`{r}
a literal code chunk
```
````
````

当文档被编译后，内联表达式将会消失，你会看到：

````
```{r}
# a literal code chunk
```
````

[1]如果要在列表等其他环境中显示文字形式的代码块，请遵循缩进规则：https://pandoc.org/MANUAL.html#block-content-in-list-items
````

编译文档时通常不需要直接调用 **knitr** 函数，因为 **rmarkdown** 会调用 **knitr**。如果你希望编译源文档而不进一步将其转换为其他格式，可以使用 `knitr::knit()` 函数。

B.2 R Markdown

由于 R 和 Pandoc 的强大功能，你可以轻松地在 R Markdown 文档中进行计算，并将其转换为各种输出格式，包括 HTML/PDF/Word 文档、HTML5/Beamer 幻灯片、仪表板和网站等。R Markdown 文档通常由 YAML 元数据（可选）和文档主体组成。我们在第 2 章中介绍了编写文档主体各个组件的语法，并在本节中详细解释了 YAML 元数据。

R Markdown 的元数据可以写在文档的最开头，分别以三个短划线 `---` 开头和结尾。YAML 元数据通常由冒号分隔的标记值对组成，例如：

```
---
title: "An R Markdown Document"
author: "Yihui Xie"
---
```

对于元数据中的字符值，当其不包含特殊字符时，你可以省略引号，但如果希望它们是字符值，则使用引号更为安全。

除字符类型外，另一种元数据取值的常见类型是逻辑类型。`yes` 和 `true` 都表示 true，`no`/`false` 都表示 false，例如：

```
link-citations: yes
```

元数据取值可以是向量，并且有两种写入向量的方法。下面两种方法是等价的

```
output: ["html_document", "word_document"]
```

```
output:
  - "html_document"
  - "word_document"
```

元数据取值也可以是值的列表，只需要将其额外缩进两个空格，例如：

```
output:
  bookdown::gitbook:
    split_by: "section"
    split_bib: no
```

忘记缩进元数据取值是一个常见的错误。例如，下面的数据

```
output:
html_document:
toc: yes
```

实际上表示

```
output: null
html_document: null
toc: yes
```

而不是你可能期望的那样：

```
output:
  html_document:
    toc: yes
```

R Markdown 输出格式在 YAML 元数据中的 `output` 字段中指定，并且你需要查阅 R 帮助页面以获得可以填写的选项，例如 `?rmarkdown::html_document` 或 `?bookdown::gitbook`。YAML 中其它大多数字段的含义可以在 Pandoc 文档中找到。

rmarkdown 软件包提供了这些 R Markdown 输出格式：

- `beamer_presentation`
- `context_document`
- `github_document`
- `html_document`
- `ioslides_presentation`
- `latex_document`
- `md_document`
- `odt_document`
- `pdf_document`

- powerpoint_presentation
- rtf_document
- slidy_presentation
- word_document

在其他 R 软件包中有更多可能的输出格式，包括 **bookdown**、**tufte**、**rticles**、**flexdashboard**、**revealjs** 和 **rmdformats** 等。

C

常见问题

下面是常见问题 (FAQ) 的完整列表。是的，这里只有一个问题。我个人不喜欢 FAQs。它们通常意味着惊喜，而惊喜对软件用户来说并不好。

1. 问：**bookdown** 会不会有 X、Y 和 Z 功能？

   答：简而言之，答案是否定的。但是如果你已经问了自己多次“我真的需要这些功能吗”，而答案仍然是“是”时，请随时向我们提出功能要求 https://github.com/rstudio/bookdown/issues。

   用户要求的更多功能往往来自 LaTeX 世界。如果你的功能要求是这样的话，这个问题的答案是肯定的，因为 Pandoc 的 Markdown 支持原始 LaTeX 代码。每当你觉得 Markdown 不能为你完成这项工作时，总是可以选择在 Markdown 文档中应用一些原始的 LaTeX 代码。例如，你可以使用 **glossaries** 软件包创建术语表，或者你知道 LaTeX 语法的话，可以嵌入一个复杂的 LaTeX 表。但是请记住，LaTeX 内容不可移植。它只适用于 LaTeX/PDF 输出，在其他类型的输出中将被忽略。根据要求，我们将来可能会在 **bookdown** 中引入更多的 LaTeX 功能，但我们的基本理念是 Markdown 应该尽可能简单。

世界上最具挑战性的不是学习花哨的技术，而是控制自己的狂野之心。

参考文献

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., and Iannone, R. (2021a). *rmarkdown: Dynamic Documents for R*. R package version 2.9.

Allaire, J., Xie, Y., R Foundation, Wickham, H., Journal of Statistical Software, Vaidyanathan, R., Association for Computing Machinery, Boettiger, C., Elsevier, Broman, K., Mueller, K., Quast, B., Pruim, R., Marwick, B., Wickham, C., Keyes, O., Yu, M., Emaasit, D., Onkelinx, T., Gasparini, A., Desautels, M.-A., Leutnant, D., MDPI, Taylor and Francis, ??reden, O., Hance, D., Nüst, D., Uvesten, P., Campitelli, E., Muschelli, J., Hayes, A., Kamvar, Z. N., Ross, N., Cannoodt, R., Luguern, D., Kaplan, D. M., Kreutzer, S., Wang, S., Hesselberth, J., and Dervieux, C. (2021b). *rticles: Article Formats for R Markdown*. R package version 0.20.

Aust, F. (2019). *citr: RStudio Add-in to Insert Markdown Citations*. R package version 0.3.2.

Chang, W. (2019). *webshot: Take Screenshots of Web Pages*. R package version 0.5.2.

Cheng, J. (2018). *miniUI: Shiny UI Widgets for Small Screens*. R package version 0.1.1.1.

Knuth, D. E. (1984). Literate programming. *The Computer Journal*, 27(2):97–111.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Vaidyanathan, R., Xie, Y., Allaire, J., Cheng, J., Sievert, C., and Russell, K. (2020). *htmlwidgets: HTML Widgets for R*. R package version 1.5.2.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2020). *servr: A Simple HTTP Server to Serve Static Files or Dynamic Documents*. R package version 0.21.

Xie, Y. (2021a). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.22.

Xie, Y. (2021b). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.33.

Xie, Y. and Allaire, J. (2020). *tufte: Tufte's Styles for R Markdown Documents*. R package version 0.9.

Xie, Y., Cheng, J., and Tan, X. (2020a). *DT: A Wrapper of the JavaScript Library DataTables*. R package version 0.16.

Xie, Y., Dervieux, C., and Riederer, E. (2020b). *R Markdown Cookbook*. Chapman and Hall/CRC, Boca Raton, Florida. ISBN 9780367563837.

索引