Authoring A Book with R Markdown

Yihui Xie

2016-03-04

Contents

Chapter 1

Introduction

This book and the package **bookdown** are still under active development, and should not be considered stable at the moment. You are welcome to experiment with this package, and feedback may be sent to yihui@rstudio.com.

This book is a guide to authoring books with R Markdown (Allaire et al., 2016) and the R package **bookdown** (Xie, 2016a). It focuses on the features specific to writing books, long-form articles, or reports, such as

- How to typeset figures and tables, and cross-reference them;
- How to generate multiple output formats such as HTML, PDF, and E-Books for a single book;
- How to customize the book templates and style different elements in a book;
- The editor support (in particular, the RStudio IDE);
- How to publish a book;

It is not a comprehensive introduction to R Markdown or the **knitr** package (Xie, 2016b), on top of which **bookdown** was built. To learn more about R Markdown, please check out the online documentation http://rmarkdown.rstudio.com. For **knitr**, please see Xie (2015b). You do not have to be an expert of the R language (R Core Team, 2015) to read this book, but you are expected to have some basic knowledge about R Markdown and **knitr**. For beginners, you may get started with the cheatsheets at https://www.rstudio.com/resources/cheatsheets/. To be able to customize the book templates and themes, you should be familiar with LaTeX, HTML and CSS.

1.1 Motivation

Markdown is a wonderful language to write relatively simple documents that contain elements like sections, paragraphs, lists, links, and images, etc. Pandoc (http://pandoc.org) has greatly extended the original Markdown syntax, and added quite a few useful new features, such as footnotes, citations, and tables. More importantly, Pandoc makes it possible to generate output documents of a large variety of formats from Markdown, including HTML, LaTeX/PDF, Word, and slides.

To write a relatively complicated document like a book, there are still a few useful features missing in Pandoc's Markdown at the moment, such as automatic numbering of figures and tables in the HTML output, cross-references of figures and tables, and fine control of the appearance of figures (e.g., currently it is impossible

to specify the alignment of images using the Markdown syntax). These are some of the problems that we have addressed in the **bookdown** package.

Under the constraint that we want to produce the book in multiple output formats, it is nearly impossible to cover all possible features specific to these output formats. For example, it may be difficult to reinvent a certain complicated LaTeX environment in the HTML output using the (R) Markdown syntax. Our main goal is not to replace *everything* with Markdown, but to cover *most* common functionalities required to write a relatively complicated document, and make the syntax of such functionalities consistent across all output formats.

Another goal of this project is to make it easy to produce books that look visually pleasant. Some nice existing examples include Gitbook (https://www.gitbook.com), Tufte CSS (http://edwardtufte.github.io/tufte-css/), and Tufte-LaTeX (https://tufte-latex.github.io/tufte-latex/). We hope to integrate these themes and styles into **bookdown**, so authors do not have to dive into the details of how to use a certain LaTeX class or how to configure CSS for HTML output.

1.2 Get started

The easiest way for beginners to get started with writing a book with R Markdown and **bookdown** is through the demo `bookdown-demo` on GitHub:

1. Fork or clone the GitHub repository https://github.com/rstudio/bookdown-demo if you are familiar with GIT and GitHub, or just download it as a Zip file then unzip it locally;
2. Install the RStudio IDE (http://www.rstudio.com) if you have not done so;
3. Open the `bookdown-demo` repository you cloned or downloaded in RStudio by clicking `bookdown-demo.Rproj`;
4. Install the R package **bookdown**:

   ```
   devtools::install_github("rstudio/bookdown")
   ```

5. Open the R Markdown `index.Rmd` and click the button `Knit` on the toolbar of RStudio;

Now you should see the index page of this book demo in the RStudio Viewer. You may add or change the R Markdown files, come back to `index.Rmd`, and hit the `Knit` button again to preview the book. If you prefer not to use RStudio, you may also compile the book through command line. See the next section for details.

1.3 Usage

Normally, a book contains multiple chapters, and one chapter lives in one R Markdown file, with the filename extension `.Rmd`. Each R Markdown file must start immediately with the chapter title. Here is an example (the bullets are the filenames, followed by the file content):

- 01-intro.Rmd

  ```
  # Introduction

  This chapter is an overview of the methods that
  we propose to solve an **important problem**.
  ```

- 02-literature.Rmd

```
# Literature

Here is a review of existing methods.
```

- 03-method.Rmd

```
# Methods

We describe our methods in this chapter.
```

- 04-application.Rmd

```
# Applications

Some _significant_ applications are demonstrated
in this chapter.

## Example one

## Example two
```

- 05-summary.Rmd

```
# Final Words

We have finished a nice book.
```

By default, **bookdown** merges all Rmd files by the order of filenames, e.g., `01-intro.Rmd` will appear before `02-literature.Rmd`. Filenames that start with an underscore `_` are skipped. If there exists an Rmd file named `index.Rmd`, it will always be treated as the first file when merging all Rmd files. The reason for this special treatment is that the HTML file `index.html` to be generated from `index.Rmd` is usually the default index file when you view a website, e.g., you are actually browsing http://yihui.name/index.html when you open http://yihui.name/.

You can override the above behavior by including a configuration file named `_bookdown.yml` in the book directory. It is a YAML file (https://en.wikipedia.org/wiki/YAML), and R Markdown users should be familiar with this format since it is also used to write the metadata in the beginning of R Markdown documents. You can use a field named `rmd_files` to define your own list and order of Rmd files for the book. For example,

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

In this case, **bookdown** will just use whatever you defined in this YAML field without any special treatments of `index.Rmd` or underscores. If you want both HTML and LaTeX/PDF output from the book, and use different Rmd files for HTML and LaTeX output, you may specify these files for the two output formats separately, e.g.,

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

Because **knitr** does not allow duplicate chunk labels in a source document, you need to make sure there are no duplicate labels in your book chapters, otherwise **knitr** will signal an error when knitting the merged Rmd file.

At the moment, there are three output formats that you may use: `bookdown::pdf_book`, `bookdown::gitbook`, and `bookdown::html_chapters`. There is a `bookdown::render_book()` function similar to `rmarkdown::render()` that renders multiple Rmd documents into a book using the output format functions. You may either call this function from command line, or use it in the RStudio IDE. Here are some command line examples:

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

To use `render_book` and the output format functions in the RStudio IDE, you can define a YAML field named `knit` that takes the value `bookdown::render_book`, and the output format functions can be used in the `output` field, e.g.,

```
---
knit: "bookdown::render_book"
output:
  bookdown::gitbook:
    lib_dir: "book_assets"
  bookdown::pdf_book:
    keep_tex: yes
---
```

Then you can click the `Knit` button in RStudio to compile the Rmd files into a book.

There are more things 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`
- `chapter_name`: (for HTML output only) either a character string to be prepended to the chapter number in the chapter title (by default it is `"Chapter "`), or an R function that takes the chapter number as the input and returns a string as the new chapter number (e.g., `!expr function(i) paste('Chapter', i)`)
- `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

Here is a sample `_bookdown.yml`:

```
book_filename: "my-book.Rmd"
chapter_name: "CHAPTER "
before_chapter_script: ["script1.R", "script2.R"]
after_chapter_script: "script3.R"
```

1.4 Separate R sessions for individual chapters

Merging all chapters into one Rmd file and knitting it is one way to render the book in **bookdown**. There is actually another way: you may knit each chapter in a *separate* R session, and **bookdown** will merge the Markdown output of all chapters to render the book. We call these two approaches "Merge and Knit" (MK) and "Knit and Merge" (KM), respectively. The differences between them may seem subtle, but can be fairly important depending on your use cases.

- The most significant difference is that MK runs *all* code chunks in all chapters in the same R session, whereas KM uses separate R sessions for individual chapters. For MK, the state of the R session from previous chapters is carried over to later chapters (e.g., objects created in previous chapters are available to later chapters, unless you deliberately deleted them); for KM, all chapters are isolated from each other[1]. If you want each chapter to compile from a clean state, use the KM approach. It can be very tricky and difficult to restore a running R session to a completely clean state if you use the MK approach. For example, even you detach/unload packages loaded in a previous chapter, R will not clean up the S3 methods registered by these packages.
- One advantage of KM is that Rmd files that have not been updated since the last time the book was rendered will not be recompiled by default, unless you force all chapters to be recompiled via `render_book(force_knit = TRUE)`. This may save some time, but the speedup may not be very significant, since the major time is normally consumed by running code chunks. If time-consumed chunks are cached, the compilation time for MK and KM may be about the same.
- The KM approach will generate more files under the directory of Rmd files: each Rmd file will generate a Markdown output file (`.md`), and possibly a figure directory and a cache directory. The MK approach only renders one Rmd file, so it only has one set of output files.
- For KM, whenever you change the output format (e.g., from HTML to PDF), you must recompile all chapters (`render_book(force_knit = TRUE)`), because the Markdown output files for one format may not work for another format. There is no such issue with the MK approach.

The default approach in **bookdown** is MK. To switch to KM, you either use the argument `new_session = TRUE` when calling `render_book()`, or set `new_session: yes` in the configuration file `_bookdown.yml`.

Note the configurations `before_chapter_script` and `after_chapter_script` are ignored by KM. You can still configure `book_filename`, but it should be a Markdown filename, e.g., `_main.md`. All other configurations work for both MK and KM.

1.5 Some tips

Typesetting under the paging constraint (e.g., for LaTeX/PDF output) can be an extremely tedious and time-consuming job. I'd recommend you not to look at your PDF output frequently, since most of the time you are very unlikely to be satisfied: text may overflow into the page margin, figures may float too far away, and so on. Do not try to make things look right *immediately*, because you may be disappointed over and over again as you keep on revising the book, and things may be messed up again even if you only did some minor changes (see http://bit.ly/tbrLtx for a nice illustration).

If you want to preview the book, preview the HTML output. Work on the PDF book after you have finished the content of the book, and are very sure no major revisions will be required.

If certain code chunks in your R Markdown docuemnts are time-consuming to run, you may cache them by adding the chunk option `cache = TRUE` in the chunk header, and you are recommended to label such code chunks as well, e.g.,

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

[1] Of course, no one can stop you from writing out some files in one chapter, and reading them in another chapter.
````

Chapter 2

Components

2.1 Markdown syntax

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

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 like `^` produces a superscript (e.g., `ClO^-^` renders ClO^-). 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](http://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 2.7. Section headers can be written after a number of pound signs, e.g.,

```
# First-level header

## Second-level header

### Third-level header
```

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.,

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

The output is:

- one item

[1] To include literal backticks, just 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

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:

> "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
````

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:

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

2.2 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 has `{r}` after the three backticks and (optionally) chunk options inside `{}`, e.g.,

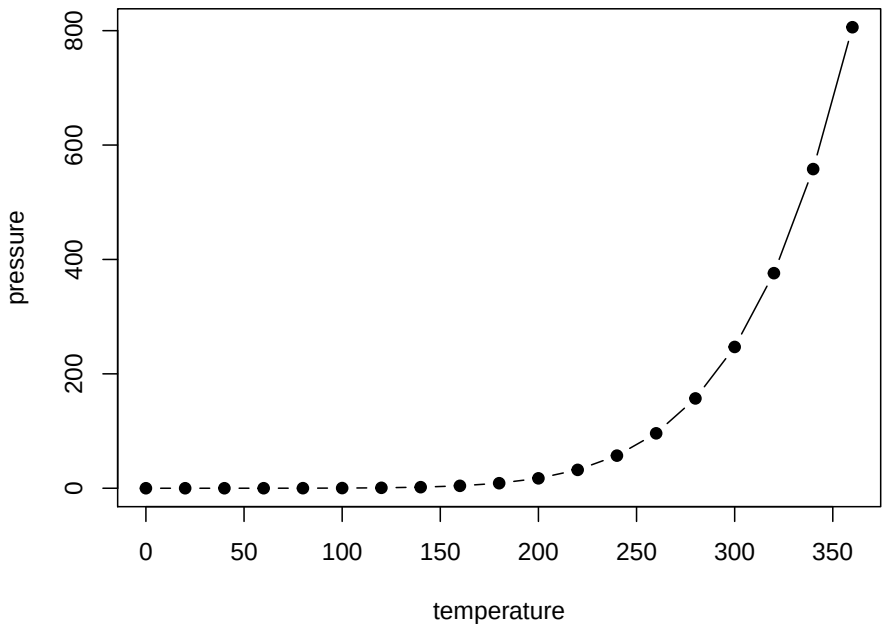
````
```{r chunk-label, echo = FALSE, fig.cap = 'A figure caption.'}
1 + 1
rnorm(10)  # 10 random numbers
plot(dist ~ speed, cars)  # a scatterplot
```
````

To learn more about **knitr** chunk options, see Xie (2015b) or http://yihui.name/knitr/options.

2.3 Figures

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

```
par(mar = c(4, 4, 0.1, 0.1))
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

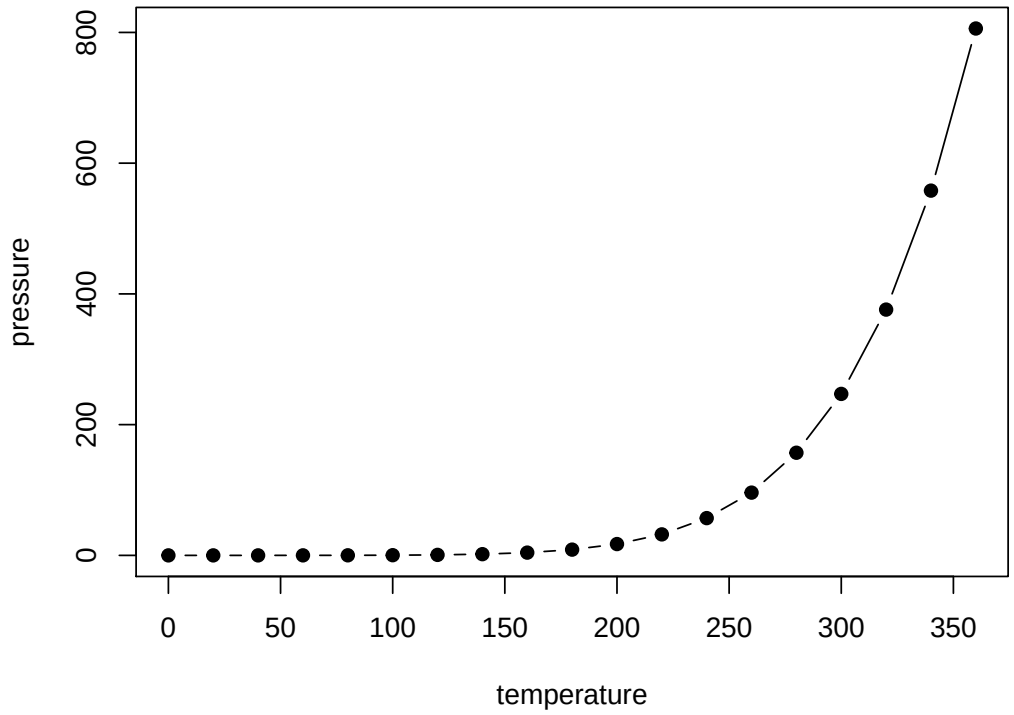


Figure 2.1: A figure example with the specified aspect ratio, width, and alignment.

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)`[2], where `label` is the figure label, e.g., `fig:foo`.

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) and dashes (-). Other characters do not qualify.

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, 0.1, 0.1))
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

[2]Do not forget the leading backslash!

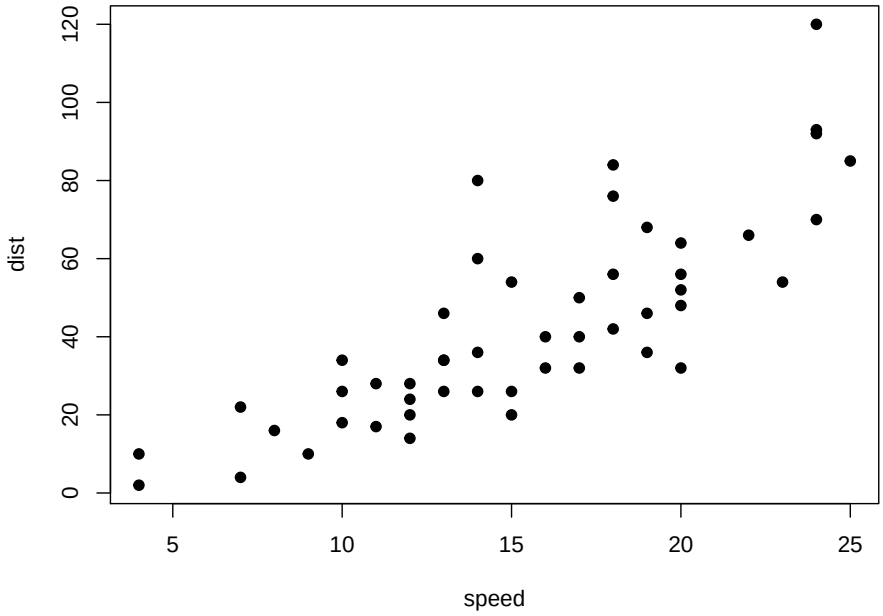


Figure 2.2: A figure example with a relative width 70%.

`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%`.

```
par(mar = c(4, 4, 0.1, 0.1))
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 `50%`.

```
par(mar = c(4, 4, 0.1, 0.1))
plot(pressure, pch = 19, type = "b")
plot(cars, pch = 19)
```

Sometimes you may have certain images that are not generated from R code, and you can include them 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))
```

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

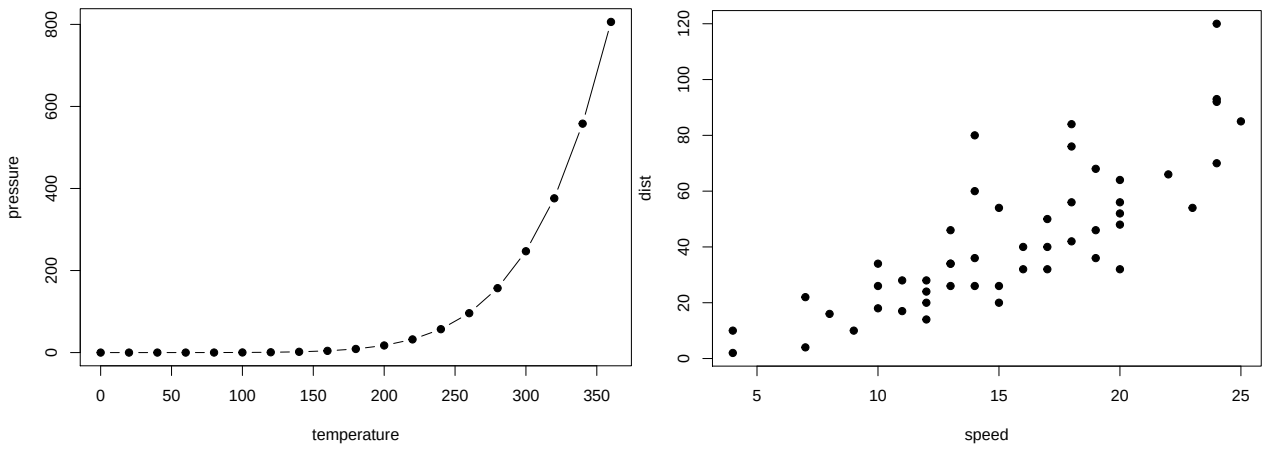


Figure 2.3: Two plots placed side by side.

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

Table 2.1: A table of the first 10 rows of the mtcars data.

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4

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()` is 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. Of course, you can disable this feature by `include_graphics(auto_pdf = FALSE)` if you do not like it.

2.4 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.

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 2.1 is a simple example.

```
knitr::kable(
  head(mtcars, 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, just wrap the data objects (usually data frames in R) into a list. See Table 2.2 for an example.

```
knitr::kable(
  list(
    head(iris[,1:2],3),
    head(mtcars[,1:3],5)
```

Table 2.2: 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

```
  ),
  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**, which can break a table across multiple pages. To use **longtable**, just 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.

```
knitr::kable(
  iris[1:100, ], longtable = TRUE, booktabs = TRUE,
  caption = 'A table generated by the longtable package.'
)
```

Table 2.3: 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
5.7	2.8	4.5	1.3	versicolor
6.3	3.3	4.7	1.6	versicolor
4.9	2.4	3.3	1.0	versicolor
6.6	2.9	4.6	1.3	versicolor
5.2	2.7	3.9	1.4	versicolor
5.0	2.0	3.5	1.0	versicolor
5.9	3.0	4.2	1.5	versicolor
6.0	2.2	4.0	1.0	versicolor
6.1	2.9	4.7	1.4	versicolor
5.6	2.9	3.6	1.3	versicolor
6.7	3.1	4.4	1.4	versicolor
5.6	3.0	4.5	1.5	versicolor
5.8	2.7	4.1	1.0	versicolor
6.2	2.2	4.5	1.5	versicolor
5.6	2.5	3.9	1.1	versicolor
5.9	3.2	4.8	1.8	versicolor
6.1	2.8	4.0	1.3	versicolor
6.3	2.5	4.9	1.5	versicolor

6.1	2.8	4.7	1.2	versicolor
6.4	2.9	4.3	1.3	versicolor
6.6	3.0	4.4	1.4	versicolor
6.8	2.8	4.8	1.4	versicolor
6.7	3.0	5.0	1.7	versicolor
6.0	2.9	4.5	1.5	versicolor
5.7	2.6	3.5	1.0	versicolor
5.5	2.4	3.8	1.1	versicolor
5.5	2.4	3.7	1.0	versicolor
5.8	2.7	3.9	1.2	versicolor
6.0	2.7	5.1	1.6	versicolor
5.4	3.0	4.5	1.5	versicolor
6.0	3.4	4.5	1.6	versicolor
6.7	3.1	4.7	1.5	versicolor
6.3	2.3	4.4	1.3	versicolor
5.6	3.0	4.1	1.3	versicolor
5.5	2.5	4.0	1.3	versicolor
5.5	2.6	4.4	1.2	versicolor
6.1	3.0	4.6	1.4	versicolor
5.8	2.6	4.0	1.2	versicolor
5.0	2.3	3.3	1.0	versicolor
5.6	2.7	4.2	1.3	versicolor
5.7	3.0	4.2	1.2	versicolor
5.7	2.9	4.2	1.3	versicolor
6.2	2.9	4.3	1.3	versicolor
5.1	2.5	3.0	1.1	versicolor
5.7	2.8	4.1	1.3	versicolor

If you decide to use other 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)`, where `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('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.

2.5 Cross-references

We have explained how cross-references work for figures (Section 2.3) and tables (Section 2.4). 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.

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.

2.6 Custom blocks

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_chapters:
    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/.

2.7 Citations

Although Pandoc supports multiple ways of writing citations, we recommend you to use BibTeX databases because they work best with LaTeX/PDF output. Pandoc can process other types of bibliography databases with the utility `pandoc-citeproc` (https://github.com/jgm/pandoc-citeproc), but it may not render certain bibliography items correctly (especially in case of multiple authors). With BibTeX databases, you will be able to define the bibiliography style if it is required by a certain publisher or journal.

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 = {2015},
  url = {https://www.R-project.org/},
}
```

A bibliography entry starts with `@type{`, where `type` may be `article`, `book`, `manual`, and so on. 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 to R Core Team (2015), and `[@R-base]` generates "(R Core Team, 2015)". 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. Note it only generates one BibTeX entry for the package itself at the moment, whereas a package may contain multiple entries in the `CITATION` file, and some entries are about the publications related to the package. These entries are ignored by `write_bib()`.

```
# the second argument can be a .bib file
knitr::write_bib(c("knitr", "stringr"), "")
```

```
@Manual{R-knitr,
  title = {knitr: A General-Purpose Package for Dynamic Report Generation in R},
  author = {Yihui Xie},
  year = {2016},
  note = {R package version 1.12.16},
  url = {http://yihui.name/knitr/},
}
@Manual{R-stringr,
  title = {stringr: Simple, Consistent Wrappers for Common String Operations},
  author = {Hadley Wickham},
  year = {2015},
  note = {R package version 1.0.0},
  url = {https://CRAN.R-project.org/package=stringr},
}
```

Once you have got one or multiple `.bib` files, you may use the field `bibliography` in the YAML metadata of your R Markdown document, and you can also specify the bibliopgraphy 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 bibilography entry in the HTML output.

2.8 HTML widgets

HTML widgets (Vaidyanathan et al., 2016) 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, 2016). 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, or save 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, 2015a):

```
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

Figure 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' 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 on 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 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 `vignette('intro', package = 'webshot')` has illustrated 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 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 `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.

2.9 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[3]; 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 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, just 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, 2016), 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 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()`.

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

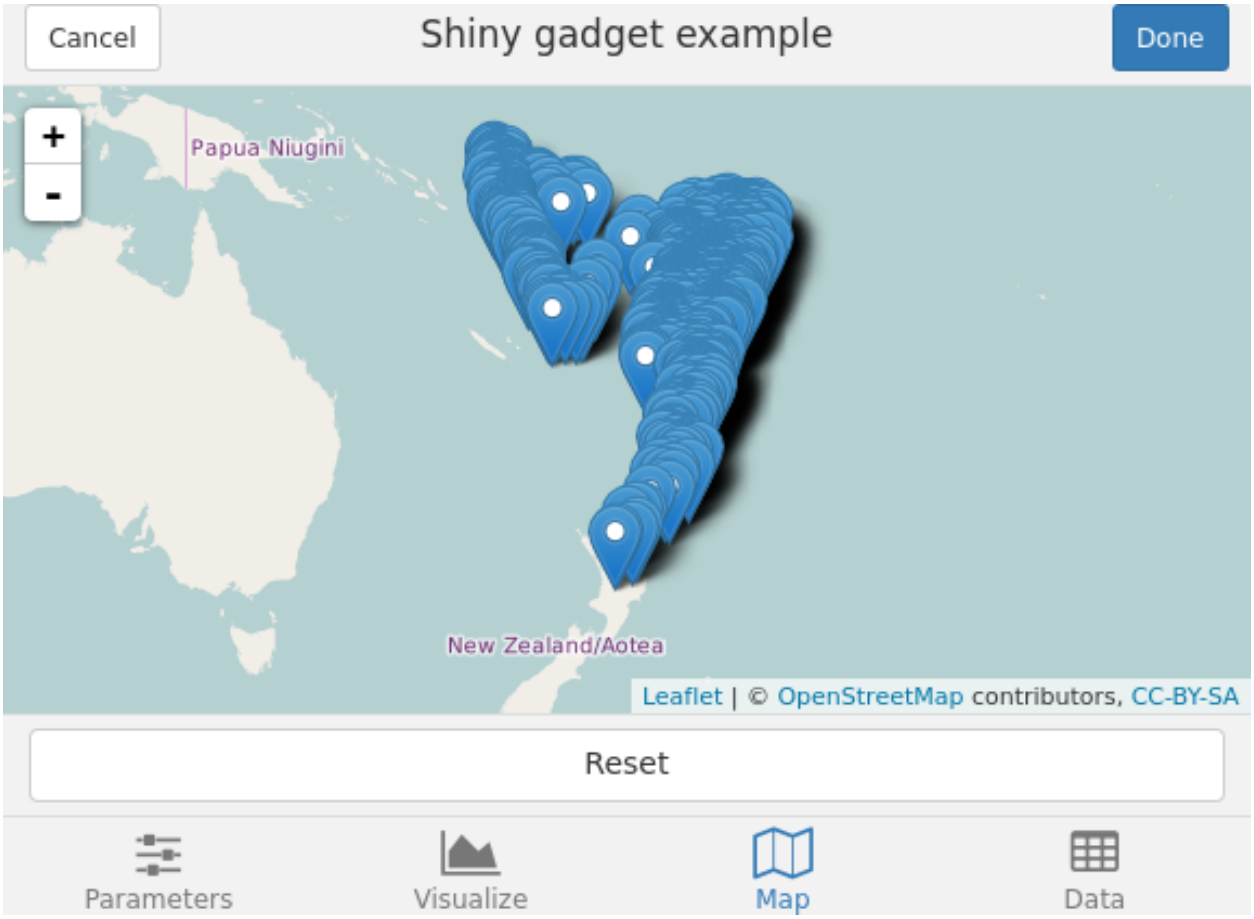


Figure 2.6: A Shiny app created via the miniUI package: https://yihui.shinyapps.io/miniUI/.

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

Chapter 3

Output Formats

3.1 HTML

multi-page HTML

3.2 LaTeX

3.3 E-Books

ePub, MOBI

Chapter 4

Customization

4.1 YAML options

includes

4.2 Theming

4.3 Templates

4.4 CJK languages

Chapter 5

Editors

5.1 RStudio

Chapter 6

Publishing

6.1 GitHub

A sketch of steps to publish to GitHub automatically:

1. Create a personal access token: https://help.github.com/articles/creating-an-access-token-for-command-line-use/
2. Encrypt it in the environment variable `GH_TOKEN` via command line `travis encrypt` and store it in `.travis.yml`, or simply save this environment variable via `https://travis-ci.org/user/repo/settings` where `user` is your GitHub ID, and `repo` is the name of the repository;
3. Create a `gh-pages` branch in your repo, and push the branch to the remote repository, e.g.,

```
git checkout --orphan gh-pages
git rm -rf .
touch .nojekyll
git add .nojekyll
git commit -m"Initial commit"
git push origin gh-pages
```

4. You can clone this `gh-pages` branch on Travis using your GitHub token, add the HTML output files from R Markdown (do not forget to add figures and CSS style files as well), and push to the remote repository, e.g.,

```
git clone -b gh-pages https://${GH_TOKEN}@github.com/${TRAVIS_REPO_SLUG}.git gh-pages
cd gh-pages
# cp ../*.html .... ./
# git add *
# git commit -m"...."
# git push origin gh-pages
```

If you use the container-based infrastructure on Travis, you can enable caching by using `sudo: false` in `.travis.yml`. Normally you should cache at least two directories: the figure directory `_main_files` and the cache directory `_main_cache`. If you have specified a different filename of the main Rmd file (Section 1.3), replace `_main` with the base name of the filename you specified. These directory names may also be different if you have specified the **knitr** chunk options `fig.path` and `cache.path`, but I'd strongly recommend you not to change these options. A `.travis.yml` file that has enabled caching of **knitr** figure and cache directories may look like this:

```
sudo: false

cache:
  directories:
  - $PWD/_main_files
  - $PWD/_main_cache
```

6.2 Publishers

6.3 Licensing

6.4 Self-publishing

Bibliography

Allaire, J., Cheng, J., Xie, Y., McPherson, J., Chang, W., Allen, J., Wickham, H., Atkins, A., and Hyndman, R. (2016). *rmarkdown: Dynamic Documents for R*. R package version 0.9.5.1.

Chang, W. (2016). *webshot: Take Screenshots of Web Pages*. R package version 0.3.

Cheng, J. (2016). *miniUI: Shiny UI Widgets for Small Screens*. R package version 0.1.1.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Vaidyanathan, R., Xie, Y., Allaire, J., Cheng, J., and Russell, K. (2016). *htmlwidgets: HTML Widgets for R*. R package version 0.6.

Xie, Y. (2015a). *DT: A Wrapper of the JavaScript Library 'DataTables'*. R package version 0.1.45.

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

Xie, Y. (2016a). *bookdown: Authoring Books with R Markdown*. R package version 0.0.41.

Xie, Y. (2016b). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.12.16.