R Markdown クックブック

(著者) Xie, Yihui　　(著者) Dervieux, Christophe

(著者) Riederer, Emily　　(翻訳者) Katagiri, Satoshi (片桐 智志)[*1]

2021/09/15 14:25:50 JST, ver. 1.0.0, 本家の更新確認時刻: 2021/08/17 19:34:48 JST[*2]

[*1] twitter @ill_identified: https://twitter.com/ill_identified

[*2] https://github.com/yihui/rmarkdown-cookbook

人生で最も驚異すべき料理人, Xie Shaobai と Si Zhinan へ

—Yihui

支えてくれた妻 Caroline と生まれたばかりの愛する Axel へ

—Christophe

生涯学び続けることの楽しさを教えてくれた母へ

—Emily

目次

表目次

図目次

はじめに

本書の原著は Chapman & Hall/CRC[a] より出版されました. 本書のオンライン版は (Chapman & Hall/CRC の厚意により) ここで無料で読むことができます. 本書はクリエイティブ・コモンズ 表示 - 非営利 - 継承 4.0 国際ライセンス[b] のもとで提供されています. ご意見は GitHub で[c] いつでも受け付けています. いつもありがとうございます.

訳注

オリジナルはこちら[d] で読むことができます.
本翻訳版に関するご意見はこちら[e] で受け付けています. また, ご覧になっているのが Web 版であれば, 上部ツールバーからプルリクエストを作成することもできます. ただし修正依頼は翻訳メモ[f] の内容を確認してからしていただけると助かります.
This is an unofficial Japanese translation of the online version of "R Markdown Cookbook" by Xie, Dervieux, and Riederer, which is licensed under CC BY-NC-SA 4.0[g]. The tranlator is Katagiri, Satoshi. The original document is here[h].

[a] https://www.routledge.com/p/book/9780367563837
[b] https://creativecommons.org/licenses/by-nc-sa/4.0/deed.ja
[c] https://github.com/yihui/rmarkdown-cookbook/issues/new
[d] https://bookdown.org/yihui/rmarkdown-cookbook/
[e] https://github.com/Gedevan-Aleksizde/rmarkdown-cookbook/issues/new
[f] https://github.com/Gedevan-Aleksizde/rmarkdown-cookbook/blob/work/dev/memo.md
[g] https://creativecommons.org/licenses/by-nc-sa/4.0/
[h] https://bookdown.org/yihui/rmarkdown-cookbook/

R Markdown は分析とレポート作成を 1 つのドキュメントとして結びつけるパワフルなツールです. 2014 年初頭に **rmarkdown** パッケージ (JJ Allaire Xie McPherson, et al., 2021) が誕生して以来, R Markdown はいくつかの出力フォーマットをサポートするだけの単なるパッケージから, 書籍・ブログ・科学論文・ウェブサイト, そして講義資料の作成までをもサポートする拡張性と多様なエコシステムを持つパッケージへと成長を遂げました.

R Markdown: The Definitive Guide[*1] (Xie J.J. Allaire, and Grolemund, 2018) という, ほんの数年前に書かれた情報の詰まったガイドブックがあります. これは **rmarkdown** パッケージやその他の拡張パッケージの組み込みフォーマットのリファレンスを詳説しています. しかし読者や出版社から, 作りたい内容を実現できるのかを見つけるまでが大変なので, より実践的で, 面白く役に立つ小規模な使用例を豊富に掲載したものがあればいいのに, というコメントをいただきました (言い換えるなら, 前書は無機質すぎるということです). これが本書の生まれた経緯です.

公式ドキュメントが存在するにも関わらず, R Markdown のユーザーは有名な Q&A フォーラム『スタック・オーバーフロー』でしょっちゅう助けを求めています. 本書の執筆時点では, `r-markdown` タグのついた質問[*2] が 6,000 件以上ありました. あなたが探すべき問題が何であるか特定していないと, この膨大な件数の中ではフォーラムを利用するのが難しくなります. よって R Markdown を使ってできること, そしてどうすればできるか, の可能性の全てを把握することが難しいものとなるかもしれません. 本書の狙いはスタック・オーバーフローやその他のオンラインリソース (ブログの投稿やチュートリアル) から有名な質問を取り上げ, 多くのユーザーが毎日こぞって検索している問題に対して最新のソリューションを提供することです. 実際, 本書で扱うトピックを決めるのに役立つよう, 第二著者の Christophe はスタックオーバーフローの日々の最も人気のある投稿をスクレイピングする R Markdown のダッシュボートを作成しました. 幸運にも, 我々のクックブックはこれらの人気の投稿を含むことでより一層役に立つものになったに違いありません.

本書は R Markdown 文書の機能を活用する多くの例を掲載しています. クックブックとしてこのガイドは, R Markdown をより効率よく使いたい, そして R Markdown の力をもっと知りたい新規または初心者ユーザーにおすすめです.

本書の読み方

本書は R Markdown の基礎を理解している読者におすすめです. *R Markdown: The Definitive Guide* (Xie J.J. Allaire, and Grolemund, 2018) の Chapter 2[*3] は R Markdown の基礎を解説しており, 新規ユーザーが読むのにおすすめです. たとえば, 本書では Markdown の構文はカバーしていませんので, 読者が他の手段でそれを学んでいる想定です. 特に, 最低でも一度は Pandoc の完全なマニュアル[*4][*5] に目を通すことを強くお薦めします. このマニュアルはかなり長大ですが, 金の鉱脈のようなものでもあります. 全てを覚えなくてもかまいませんが, Markdown の機能をどう応用できるかを知っていればとても役に立つでしょう. 多くの人々が 3 連続バッククォートを verbatim なコードブロックに書こうとして失敗したり, 子要素を持つリストを作ろうとして失

[*1] https://bookdown.org/yihui/rmarkdown/

[*2] https://stackoverflow.com/questions/tagged/r-markdown

[*3] https://bookdown.org/yihui/rmarkdown/basics.html

[*4] https://pandoc.org/MANUAL.html

[*5] 訳注: 完全ではありませんが, 日本語訳が公開されています. https://pandoc-doc-ja.readthedocs.io/ja/latest/users-guide.html

敗したりするのを, 私は数え切れないほど見てきました[*6]. マニュアルに書いてある Markdown の構文を全て読まなければ,「N 連続バッククォートに対して外側に N + 1 連続でバッククォートを書く」「子要素を表現するには適切なインデントをつける」といったことに, きっと気づかないままでしょう.

このクックブックは R Markdown の技術的なリファレンスを網羅することを意図したものではありません. 本書はこれまでにある資料に対する補足となることを目的としています. よって読者は, さらに詳細な情報を知るために以下のような本を参考にすればよいでしょう.

- *R Markdown: The Definitive Guide* (Xie J.J. Allaire, and Grolemund, 2018) は **rmarkdown** パッケージやその他いくつかの拡張パッケージでの R Markdown の出力フォーマットに関する技術的資料です.
- *R for Data Science* (Wickham and Grolemund, 2016b)[*7] の Part V "Communicate".: このパートは上記の "Definitive Guide" よりも技術的なことは少ないので, より平易な R Markdown の入門になるでしょう.
- *Dynamic Documents with R and knitr* (Xie, 2015) は **knitr** パッケージ (Xie, 2021c) の網羅的な入門書です (補足しますと, R Markdown は **knitr** パッケージのサポートする文書形式の 1 つにすぎません). 短縮版を読みたい場合, Karl Broman による最小限のチュートリアル "knitr in a knutshell"[*8] が役に立つでしょう. **訳注**: これらは日本語訳が存在しませんが, Yihui 氏によるドキュメント *knitr Elegant, flexible, and fast dynamic report generation with R*[*9] の日本語訳は既に用意してあります[*10].
- *bookdown: Authoring Books and Technical Documents with R Markdown* (Xie, 2016) は **bookdown** パッケージ (Xie, 2021a) の公式ドキュメントとして書かれた小冊子です. **bookdown** パッケージは長大なフォーマットのドキュメントを R Markdown で簡単に書くために設計されました.
- *blogdown: Creating Websites with R Markdown* (Xie Hill, and Thomas, 2017) は **blogdown** パッケージ (Xie Dervieux, and Presmanes Hill, 2021) によって R Markdown でウェブサイトを作成する方法を紹介してます.

関連性に応じて本書は既存の参考資料を紹介します. それとは別に, R Markdown の公式ウェブサイトにも役立つ情報が多く含まれています: https://rmarkdown.rstudio.com

本書は最初から順に読む必要はありません. 以降の各章はそれより前の章よりも難解になることは

[*6] https://yihui.org/en/2018/11/hard-markdown/

[*7] 本書は https://r4ds.had.co.nz/ で無料公開されています. また, 日本語訳『R ではじめるデータサイエンス』というタイトルでオライリー・ジャパンより出版されています.

[*8] https://kbroman.org/knitr_knutshell/

[*9] https://yihui.org/knitr/

[*10] https://gedevan-aleksizde.github.io/knitr-doc-ja/

ありません. 各章と各セクションのうち, 他よりも発展的と思われるものに対しては, タイトルにアスタリスク (*) を付けています. R Markdown でやりたい具体的なタスクがあるとき, あるいは目次に目を通していたら興味のある箇所が見つかった, という使い方が最も効率的な読み方でしょう. いくつかの箇所で相互参照を免れないところがありますが, 用例集を理解するのに必要な予備知識への参照のつもりです.

自分で用例集に挑戦したいならば, 本書の完全なソースコードと用例集は Github の https://github.com/yihui/rmarkdown-cookbook で自由に見ることができます[*11]. 本書の電子書籍版をお読みの場合, 掲載されているコードをお好きなテキストエディタにコピー&ペーストして実行することになるでしょう.

本書の構成

本書はそれぞれ単独のコンセプトを実演するため, 小規模な「レシピ」に細分化されています. 1 章では必要なソフトウェアツールのインストール方法を紹介しています. 2 章では R Markdown のコンセプトを概観します. 3 章では R Markdown の基本的な構成要素を紹介し, R Markdown 文書と R スクリプトの変換方法を紹介します. 4 章では, 改ページ, 参考文献リストの掲載, 番号付きの図, アニメーション, ダイアグラムといった文書の要素を作成する方法の話をします. 5 章では図の大きさやアラインメントといった文書の整形方法を紹介します. 6 章では LaTeX/PDF のみ出力したい場合に使える豆知識と小ワザを紹介します. 同様に 7 章では HTML ユーザーに対して, 8 章では Word ユーザーに対して豆知識や小ワザを紹介します. 同時に複数の出力フォーマットで生成したい場合 (これはしょっちゅう小ワザを駆使します), 9 章の記述が役に立つでしょう. 10 章は, 正直に言えば私が最も気に入らなかった箇所ですが, 私は多くのユーザーが表の作成方法を本当に欲していることを理解しています. 私はゴテゴテした装飾過多な表の専門家ではありませんが, その役に立つパッケージのリストを知ることはできるでしょう. 11 章では, あなたがまだ知らないであろう **knitr** のチャンクオプションのいくつかの応用をお教えします. 12, 13 章は **knitr** の出力とカスタムフック関数の挙動をうまく扱えるようになることのすばらしさをお教えしますので, 少し発展的ですがこれまたとても役に立つはずです. 14 章ではいろいろな **knitr** の小ワザを紹介します. 15 章では R Markdown で他のプログラミング言語を扱う例をお見せします. そう, R Markdown は R のためだけのものではありません. また, **knitr** がまだサポートしていない新しい言語でも動作させる方法も紹介します. 16 章は R Markdown とプロジェクトを関連付けて管理するための豆知識を紹介します. 17 はあなたのワークフローを改善する豆知識をいくつか提示します.

本書のレシピはそれぞれ独立した項目になっているので, あなたに決まった目的がなくてもこれら

[*11] 訳注: この日本語版のソースコードは https://github.com/Gedevan-Aleksizde/rmarkdown-cookbook で見られます. 用例集はさらに JP/examples ディレクトリを辿ることで見つかります.

の中から適当に取り上げて読むことができます.

ソフトウェア情報と表記のルール

本書をコンパイルした時点での基本的な R セッション情報は以下のとおりです[*12].

```
xfun::session_info(c(
  'bookdown', 'knitr', 'rmarkdown', 'rmdja', 'xfun'
), dependencies = FALSE)
```

```
## R version 4.1.1 (2021-08-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.3 LTS
##
## Locale:
##   LC_CTYPE=ja_JP.UTF-8
##   LC_NUMERIC=C
##   LC_TIME=ja_JP.UTF-8
##   LC_COLLATE=ja_JP.UTF-8
##   LC_MONETARY=ja_JP.UTF-8
##   LC_MESSAGES=ja_JP.UTF-8
##   LC_PAPER=ja_JP.UTF-8
##   LC_NAME=C
##   LC_ADDRESS=C
##   LC_TELEPHONE=C
##   LC_MEASUREMENT=ja_JP.UTF-8
##   LC_IDENTIFICATION=C
##
## Package version:
##   bookdown_0.24  knitr_1.34     rmarkdown_2.11
##   rmdja_0.4.6.9  xfun_0.26
##
## Pandoc version: 2.14.2
```

[*12] 訳注: 日本語版作成にあたって, **rmdja** パッケージ[*13]の開発版を使用しているため, 完全に同一のファイルを作成できる保証はないことをご容赦ください.

上記のセッション情報を見て分かるように, 本書では R ソースコードにプロンプト記号 (> や +) を付けたりしません. またテキスト出力は 2 連続ハッシュ ## でコメントアウトしています. これはコードをコピーして実行する際の利便性のためです (テキスト出力はコメントアウトされているので無視されます). パッケージ名は太字 (例: **rmarkdown**) で表記し, 本文中のコードやファイル名はタイプライタフォントで表記します (例: `knitr::knit('foo.Rmd')`). 関数名の末尾には括弧を付けます (例: `blogdown::serve_site()`). 二重コロン演算子 `::` はパッケージのオブジェクトへのアクセスを意味します.

"Rmd" は R Markdown のファイル拡張子名であり, 本書では R markdown の略称としても使用します.

謝辞

いつものことですが, まず本書の執筆作業の自由を与えていただいた雇用主である RStudio 社に感謝の意を表します. 執筆作業が始まってから, 上司である Tareef Kawaf との毎週のミーティング当初 15 分から 5 分に削減され, それから完全になくなりました. 私は複数の友人から所属先で耐えられないほど多くのミーティングがあり, 時間の浪費になっていると聞いていました. 集中力の維持の観点から, 最近ある友人は「5 分間 Slack をミュートすることができるかもしれないが, **1 日中**ミュートできないのか?」と嘆きました「もちろんできる!」と私は答えました. 私は 1 ヶ月でも好きなだけ Slack をミュートできるようになったようです. 誤解しないでください — Tareef や同僚が邪魔だという意味ではありません. 皆の提供してくれた自由がどれだけ価値あることかを伝えたいだけです.

R Markdown Definitive Guide を刊行したのち, このクックブックを執筆することを思いつきましたが, アイディアはまだ貧弱でした. 困難で高く付く作業でした. 最初に Michael Harper[*14] の後押しがなければ, この作業にまじめに取り組むことはなかったでしょう. Christophe Dervieux は助けが必要なときにいつも近くにいました. 彼の R と R Markdown のスキルにより作成されたダッシュボード (**flexdashboard** パッケージによるもの) は人々が興味を持つであろうもの, 役に立つであろうトピックを本書に記載する助けになりました. 同時に多数の Github issues を手伝ってくれたため, 最小限の再現例も添付してないバグ報告と格闘する時間を執筆作業に割くことができました. 同様に, Martin Schmelzer, Marcel Schilling, Ralf Stubner をはじめ数名がスタック・オーバーフロー上の R Markdown の質問に答えるのを手伝ってくれました. おそらく意図してのことではないと思いますが, 彼らの努力は私の多くの持ち時間を節約してくれました. 最近のスタック・オーバーフローでは Johannes Friedrich の活動が注意を引きます. これまでに何度か, スタック・オーバーフローの質問を開いたら彼がもう回答していた, ということがありました.

10.3 節では David Keyes が私を救ってくれました. 私は彼のことをあまり知りませんでしたが, 表

[*14] http://mikeyharper.uk

を作成するためのパッケージをいくつか紹介する すばらしいブログ記事[*15]を彼が書いていたおかげで助かりました. Holtz Yan の R Markdown の豆知識に関する投稿[*16], Nicholas Tierney の本 *R Markdown for Scientists*[*17] Maëlle Salmon の R Markdown の講座[*18], Jennifer Thompson の R Markdown の講座[*19], Emi Tanaka の R Markdown のワークショップ[*20], Alison Hill の R Markdown ワークショップ[*21] (私も講師の 1 人です), Alison Hill と Emi Tanaka's R Markdown のワークショップ[*22] といったそれ以外のオンライン上の資料もまた, たいへん助けになりました.

Maria Bekker-Nielsen Dunbar, Nathan Eastwood, Johannes Friedrich, Krishnakumar Gopalakrishnan, Xiangyun Huang, Florian Kohrt, Romain Lesur, Jiaxiang Li, Song Li, Ulrik Lyngs, Matt Small, Jake Stephen, Atsushi Yasumoto, Hao Zhu, John Zobolas といった方々がプルリクエストを送ったり, issues を埋めたりして多くの方が本書の Github リポジトリに貢献してくれました. 本書の素晴らしい表紙絵は Allison Horst によってデザインされ[*23], 全体のデザインは Kevin Craig によって完成されました.

本書の当初のアイディアの一部は 2018 年の RaukR Summer School で **knitr** のあまり知られていない機能について のリモート講演で生まれたものでした. 視聴者は **knitr** の機能についてレシピ形式のような手短な入門を好んでいるようでした. 私を招待していただいた, Marcin Kierczak と Sebastian Dilorenzo をはじめとするサマースクールのオーガナイザたちに感謝したいです. Genentech と DahShu.[*24] でものちに同様の講演を行いました. 招待していただいた Michael Lawrence と Yuqing Zhang, そしてフィードバックをくれた視聴者のみなさんにも感謝したいです. Paul Johnson からは 2020 年刊の *The American Statistician* に掲載された *R Markdown: The Definitive Guide* に対するとても有意義な批評をいただきました. 彼がこの本には詳細な例が欠けていると批判してくれたため, この「決定版ガイド」は十分に決定的とはいえないことになりました. 彼の論評には心から称賛と賛意を送ります. この新しいクックブックがこの溝を埋めてくれることを願います.

これは編集者の John Kimmel との仕事で 5 番目になる本です. 彼と Chapman & Hall/CRC のチームとの共同作業は常に喜びに満ちていました. 他の著者たちに **bookdown** が広く利用されるのは **bookdown** の成功だと John が言ってくれるたびに私は興奮しました. 私の以前の著作の プロダクションエディターであった Suzanne Lassandro が, 他にも多くの責任ある立場にあり著者と直接の接点がほとんどなくなった今も, 本書の手助けになるよう熱心に取り組んでいる

[*15] https://rfortherestofus.com/2019/11/how-to-make-beautiful-tables-in-r/
[*16] https://holtzy.github.io/Pimp-my-rmd/
[*17] https://rmd4sci.njtierney.com
[*18] https://github.com/maelle/rmd_course_isglobal
[*19] https://github.com/jenniferthompson/RepResearchRMarkdown
[*20] https://github.com/emitanaka/combine2019
[*21] https://arm.rbind.io
[*22] https://ysc-rmarkdown.netlify.app
[*23] https://github.com/yihui/rmarkdown-cookbook/issues/180
[*24] http://dahshu.org

と John から聞いて私は誇りに思いました. Suzanne と校正担当 (Rebecca Condit) は初稿から「たったの」377 箇所の問題を見つけ出してくれました. 実は次の本のミスは 30 箇所くらいだろうという以前の私の予想[*25]は楽観的すぎました. LaTeX の専門家 Shashi Kumar は PDF を印刷する直前の最後の障害となった, 厄介な LaTeX の問題を解決する手助けをしてくれました.

John は原稿へのフィードバックのために数名の査読を用意してくれました. 実質的に 9 人の偉大な査読を得ることになりました. 彼らの 1 人は共同著者として迎えられれば良かったのにと思うほど偉大でした! 9 人の査読との作業は膨大でしたが, 間違いなく苦労に見合った価値がありました. Carl Boettiger, John Blischak, Sharla Gelfand, Johannes Friedrich, Atsushi Yasumoto, そして残りの匿名の査読たちの有意義なフィードバックに感謝の意を送りたいと思います.

本書の最後のパートの作業は私の昔なじみの友人, Dong Guo と Qian Jia が引っ越した後の空き家 (ネット回線なし!) で行いました. 私が疲労困憊しとにかく静かな環境を必要としていた時, 家を一時的なオフィスとして使わせてくれた彼らに感謝します. 彼らに別れを告げるのは悲しいです. 私にとって, この本を彼らの家で仕上げられたことは, 両親とかわいらしい娘のいる彼ら家族とともに良き思い出となるでしょう.

最後に絶対に逃せないユニークなこととしては, COVID-19 のパンデミックの下で自宅にいた 2 人の小さな「超役に立つ同僚」(5 歳と 3 歳) に感謝することです. もし 2 人がいなければ, 5 ヶ月は早く刊行できたでしょう. 今となっては託児所 (Small Miracle) の先生が懐かしいですし, 料金もきっと高くはないと感じています...

Yihui Xie ネブラスカ州エルクホーンにて

[*25] https://bookdown.org/yihui/rmarkdown/acknowledgments.html

著者について

Yihui が本書のほとんどの文を書きました．これが第一著者であることを正当化する唯一の根拠です．Christophe は全ての Github issues をまとめ, そしていくつかのセクションを書いたというはっきりした貢献があります．Emily は本来は本書の査読者でした．Yihui が彼女を共同著者として招いたのは, Yihui が彼女と長いコメントでやり取りできるほどがまん強くないので, 自分でとてもうまく書けたと思っていたものに大量の追加注文を付けられることの苦しみを分からせるため (つまり仕返しのため) でした．いいえ, これは冗談です．彼女のコメントはとても有意義でしたが, Yihui には提案された全ての改善案に対処する時間がなかったため, 全幅の評価で彼女を招待したのです．

本書で「私」という表現があれば, それは Yihui のことを指します．「我々」ではなく「私」を使うのは, 共著者のことを忘れてしまったからではなく, 完全に Yihui の独自の意見を表明したいことを意味しています．彼は賢く見られたいと思っていますが, 実は愚かであるというのなら, それは自分だけであってほしいと思っています．

Yihui Xie

Yihui Xie (https://yihui.org) は RStudio (https://www.rstudio.com) のソフトウェアエンジニアです．アイオワ州立大学の統計学部で PhD を取得しました．インタラクティブな統計的グラフィックと統計的コンピューティングに関心があります．活動的な R ユーザーとして, **knitr**, **bookdown**, **blogdown**, **xaringan**, **tinytex**, **rolldown**, **animation**, **DT**, **tufte**, **formatR**, **fun**, **xfun**, **testit**, **mime**, **highr**, **servr**, **Rd2roxygen** といった R パッケージを開発しています．その中でも **animation** パッケージは 2009 年の John M. Chambers Statistical Software Award (ASA) を受賞しています．また, **shiny**, **rmarkdown**, **pagedown**, **leaflet** といったパッケージの開発メンバーにも加わっています．

彼は 2 つの本を書いています, *Dynamic Documents with knitr* (Xie, 2015), と *bookdown: Authoring Books and Technical Documents with R Markdown* (Xie, 2016), です．そして 2 つの本の共著者です, *blogdown: Creating Websites with R Markdown* (Xie Hill, and Thomas, 2017), と *R Markdown: The Definitive Guide* (Xie J.J. Allaire, and Grolemund, 2018) です．

2006 年, 彼は Capital of Statistics (https://cosx.org) を設立しました. これは中国国内の大きな統計学のオンラインコミュニティに成長しました. 彼はまた 2008 年に Chinese R conference を開始し, 以降, 中国での R カンファレンスの企画に関わってきました. アイオワ州立大学での PhD のトレーニングの間に, 彼は学部で Vince Sposito Statistical Computing Award (2011 年) と Snedecor Award (2012)[*26]を受賞しました.

彼はたいていの場合, twitter のメッセージを週に 1 度確認します (https://twitter.com/xieyihui). ほとんどの時間, 彼は Github (https://github.com/yihui) で見かけることができます.

彼には 4 つの趣味があります. 読むこと, 書くこと (ほとんどはブログ), 料理, そしてバドミントンをすることです. 彼は実際食べるより料理する方に関心があります. 食べるのを我慢できないほど好きな料理はそう多くありませんが, その一例は激辛料理でしょう. 料理がさらに楽しくなったので, レストランに行くことがまれになっています. レストランに行って「料理はどれくらい辛くしたらよろしいでしょうか」と訊かれれば, 彼はたいてい「シェフができる限界まで辛くして」と答えます.

Christophe Dervieux

Christophe Dervieux は R コミュニティの活動的なメンバーであり, 現在はフランス在住です. エネルギーと経済に関する修士号を取得しており, アナリストとして R を使った最初の仕事はマーケットデザインに関する経済研究です. これは developer advocate および R 管理者となって R の布教と職場での R ユーザーへのサポートをするようになる前のことです.

彼は人々が R を 最大限活用できるように手助けすることに関心があり, 彼が RStudio Community で sustainer として, あるいはいくつかの R パッケージの Github issues で動き回る姿を目にすることができるでしょう. どちらの場でも, “cderv” という短縮ハンドルネームで彼だと認識できるでしょう.

R 開発者としての彼は, **bookdown**, **rmarkdown**, **knitr** といったいくつかの R パッケージのコントリビューターです. **crrri** パッケージの開発メンバーの一人でもあります. 彼自身のプロジェクトは GitHub (https://github.com/cderv) で確認できますし, ときどき Twitter でアイディアを共有することもあります.

彼は辛い料理は苦手ですが, 毎週バドミントンを楽しんでいます.

[*26] 訳注: Committee of Presidents of Statistical Societies の授与するものではなく, 大学内で学生に授与されるもの

Emily Riederer

Emily Riederer は消費者金融業界でデータサイエンスの仕事をしており, R を使った分析ツールを構築するチームを率い, この業界でオープンサイエンスの文化を育てています. それ以前は, 彼女はチャペルヒルのノースカロライナ大学で数学と統計学を専攻していました.

Emily は頻繁に Twitter (https://twitter.com/emilyriederer) や自分のブログ (https://emily.rbind.io) で R について議論し, プロジェクトを共有します. その中には GitHub (https://github.com/emilyriederer) にある **projmgr** パッケージも含まれます. rOpenSci のパッケージレビュアーとしても活動し, さらに satRday Chicago R conference の発起メンバーの一人でもあります.

Emily の他の関心は読書とウェイトリフティングです. 彼女は自分で辛い料理が好きだと考えていますが, 合衆国内にしか住んだことがないため, その言葉が実際に意味するところをよく分かっていないのだと言われています.

翻訳者情報 (About Translators)

この日本語版ページ, および PDF ファイルを作成した人間の情報です.

片桐智志 (Katagiri, Satoshi)

山田工業所の中華鍋 (両手鍋) を使用しています. 私も四川料理のような辛いものは好きです.

I am the main translator, which means the most of this text is translated by me. Thus I am the mainly responsible person for this translation. I use a southern-style wok. I also like spicy dishes like Sichuan cuisine.

翻訳協力者への謝辞

加えて, 日本語版の修正提案に協力していただいた方を以下にクレジットします. R Markdown クックブックなので R を使って機械的に掲載してみます. 以下に Github での PR がマージされた方のアカウント名が表示されます.

- nnawata[*27]

これは以下のようなプログラムで出力しています.

```
contributors <- rbind(read.csv(textConnection(system("git shortlog -s master JP",
  intern = T)), header = F, sep = "\t"), read.csv(textConnection(system("git shortlog -s
  ↪ work JP",
  intern = T)), header = F, sep = "\t"))
contributors <- aggregate(contributors[, 1], by = list(contributors$V2),
```

[*27] https://github.com/nnawata

```
  sum)
contributors <- subset(contributors, !charmatch(contributors$Group.1,
  c("Katagiri, Satoshi", "S-Katagiri"), F))
cat(paste0("* [", contributors[order(contributors$x), ]$Group.1,
  "](", "https://github.com/", contributors[order(contributors$x),
    ]$Group.1, ")"), sep = ", ")
```

第1章

インストール方法

R Markdown を使うには R (R Core Team, 2021) と R パッケージである **rmarkdown** (JJ Allaire Xie McPherson, et al., 2021) のインストールが必要です.

```
# CRAN から rmarkdown パッケージを R にインストール
install.packages("rmarkdown")

# または，開発版をインストールしたければ GitHub
# からインストール
if (!requireNamespace("remotes")) install.packages("remotes")
remotes::install_github("rstudio/rmarkdown")
```

こだわりのあるテキストエディタや IDE (統合開発環境) がなければ, RStudio IDE (https://www.rstudio.com) のインストールも推奨します. RStudio は必須ではないですが, エディタに強力な R Markdown 支援機能があるので平均的なユーザーにとっては作業がより簡単になります. RStudio IDE を使わない選択をしたなら, Markdown を他の形式の文書に変換するために **rmarkdown** が使用する Pandoc(1.1 節参照) をインストールする必要があります.

PDF として作成する必要があるなら, LaTeX (1.2 節) およびいくつかのパッケージ (1.3) のインストールも必要になるかもしれません.

1.1 RStudio IDE にバンドルされていないバージョンの Pandoc を使う

RStudio IDE は特定のバージョンの Pandoc を同梱しているため, RStudio IDE を使用する場合は自分で Pandoc をインストールする必要はありません. しかし同梱されたバージョンが最新でないことはよくありますし, 必要なバージョンでないかもしれません. 別の Pandoc を自分でインストールすることができます. ほとんどの RStudio ユーザーは同梱されたバージョンを使用しているでしょうから, このバージョンの Pandoc は R Markdown での徹底的なテストを乗り越えていることを覚えておいてください. 異なるバージョン (特に新しいバージョン) を使う場合, 他の R Markdown ユーザーや開発者が解決できない問題にぶつかるかもしれません.

Pandoc のサイトに, プラットフォームごとの Pandoc のインストール方法の詳細なインストラクション https://pandoc.org/installing.html があります. 特定のバージョンを使うために Pandoc を自分でインストールしたのなら, 例えば以下のように `rmarkdown::find_pandoc()` 関数を呼び出して **rmarkdown** パッケージにそのことを知らせることになるでしょう.

```
# 特定のバージョンを検索
rmarkdown::find_pandoc(version = "2.9.1")

# 特定のディレクトリから検索
rmarkdown::find_pandoc(dir = "~/Downloads/Pandoc")

# 以前発見した Pandoc を無視して再検索する
rmarkdown::find_pandoc(cache = FALSE)
```

上記のコードチャンクのように, Pandoc のバージョンを特定する方法はいくつかあります. デフォルトでは `rmarkdown::find_pandoc()` はお使いのシステムの最新の Pandoc を発見しようとします. 発見できたなら, バージョン情報はキャッシュされ `cache = FALSE` でキャッシュは無効化されます. `pandoc` 実行ファイルの発見されるであろうディレクトリがどこにある可能性があるかは, ヘルプページの `?rmarkdown::find_pandoc` を見てください.

この関数は Rmd 文書内でも外部でも呼び出される可能性があります. あなたのコンピュータにインストールした特定のバージョンの Pandoc で Rmd 文書をコンパイルしたい場合, この関数を文書内のチャンクのどれかで呼び出すことになるでしょう. 例えばセットアップ用のチャンクで以下のように.

```
```{r, setup, include=FALSE}
rmarkdown::find_pandoc(version = '2.9.1')
```
```

1.2 PDF レポートの作成に LaTeX (TinyTeX) をインストールする

R Markdown から PDF 文書を作りたいなら, LaTeX がインストール済みである必要があります. 伝統的な選択肢として MiKTeX, MacTeX, そして TeX Live がありますが, R Markdown ユーザーに対しては TinyTeX[*1] のインストールを推奨します.

TinyTeX は TeX Live をもとにカスタムされた LaTeX ディストリビューションで, 比較的サイズが小さく, それでいて, 特に R ユーザーが使うようなほとんどの機能を備えています. TinyTeX のインストールや起動にはシステム管理者権限は不要です[*2]. TinyTeX は R パッケージの **tinytex** (Xie, 2021e) でインストールできます.

```
tinytex::install_tinytex()
# TinyTeX をアンインストールするなら,
# tinytex::uninstall_tinytex() を実行してください
```

“**tinytex**” は R パッケージのことを指し, “TinyTeX” は LaTeX ディストリビューションを指すことに注意してください. TinyTeX を使う利点は 2 つあります.

1. TinyTeX は (他の LaTeX ディストリビューションと比べて) 軽量であり, クロスプラットフォームでありポータブルです. 例えば USB ドライブや他のポータブルデバイスに TinyTeX のコピーを保存し, 同じオペレーティングシステムの別のコンピュータで使用することができます.
2. R Markdown を PDF へ変換する時, Pandoc はまず Markdown を中間ファイルとして LaTeX 文書に変換します. **tinytex** パッケージは LaTeX 文書を PDF にコンパイルするヘルパー関数を提供します (主な関数は `tinytex::latexmk()` です). TinyTeX を使っていて, インストールされていない LaTeX パッケージが必要ならば, **tinytex** は自動でインス

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

[*2] というより, あなたがシステムの唯一のユーザーなら Linux や macOS では TinyTeX を root 権限で (つまり sudo で) インストール**しない**ことをお薦めします.

トールしようとします. LaTeX ファイルに対するコンパイルも, 全ての相互参照を確実に解決するために十分な回数だけ行おうとします.

技術的に詳しい話に興味があるなら, Xie (2019) の論文と https://yihui.org/tinytex/faq/ の FAQ を確認するとよいかもしれません.

訳注

上記の FAQ を含む **tinytex** パッケージのドキュメントの日本語版も翻訳者により作成されています

https://gedevan-aleksizde.github.io/tinytex-doc-ja/index.html

1.3 足りない LaTeX パッケージをインストールする

文書を LaTeX を通して PDF にコンパイルしたい時, これらのようなエラーに遭遇するかもしれません.

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

!pdfTeX error: pdflatex (file 8r.enc):
  cannot open encoding file for reading

!pdfTeX error: /usr/local/bin/pdflatex (file tcrm0700):
  Font tcrm0700 at 600 not found
```

1.2 節で紹介した TinyTeX を使用しているなら, だいたいの場合このようなエラーに対処する必要はありません. **tinytex** (Xie, 2021e) が自動で対処してくれるからですが, 何らかの理由でこのようなエラーに遭遇した場合でもやはり, `tinytex::parse_install()` で足りない LaTex パッケージをインストールするのは簡単です. この関数は LaTeX ログファイルのパスを引数として, 足りないパッケージの問題を自動的に解決し, CTAN (the Comprehensive TEX Archive Network, https://ctan.org) で見つけられたものをインストールしようとします. LaTeX ログファイルは典型的には入力文書ファイルとおなじ基底名と, `.log` という拡張子名を持ちます. このログファイルを見つけられない場合, エラーメッセージをこの関数の `text` 引数に与えることができます. どちらの方法でも動作するはずです.

```
# ログファイルが filename.log だとする
tinytex::parse_install("filename.log")

# または `text` 引数を使う
tinytex::parse_install(
  text = "! LaTeX Error: File `ocgbase.sty' not found."
)
# "ocgx2" パッケージがインストールされる
```

TinyTeX を使わない場合, **tinytex** パッケージはやはりエラーログから LaTeX パッケージ名を解決しようとします. `tinytex::parse_packages()` を例えばこのように使用してください.

```
# ログファイル名が filename.log だったとする
tinytex::parse_packages("filename.log")

# または `text` 引数を使う
tinytex::parse_packages(
  text = "! LaTeX Error: File `ocgbase.sty' not found."
)
# "ocgx2" と返ってくるはず
```

パッケージ名が判明したら, あなたの LaTeX ディストリビューションのパッケージマネージャでインストールすることができます.

代わりに MiKTeX を使っているなら, これも自動で足りないパッケージをインストールできます. MikTeX のインストール中に “Always install missing packages on-the-fly” の設定に必ずチェックしてください. この設定をせずにインストールしていても, まだ MiKTeX Console で変更できます[*3].

[*3] https://github.com/rstudio/rmarkdown/issues/1285#issuecomment-374340175

訳注

日本語文書を作成する場合, いくらか追加の作業が必要かもしれません. 例えばコンパイル時, 毎回翻訳ファイルがないという旨の警告が出るかもしれません. これは出力に影響しませんが, 煩わしく感じるなら以下のようにして対応する LaTeX パッケージ (R のパッケージではないことに注意してください) をインストールすることで解決できます.

```
tinytex::tlmgr_install("texlive-msg-translations")
```

一方で以前から TeX Live を使用しているがここ数年は更新していない, という方にとっては, 手動でパッケージをインストールする必要があるかもしれません. 2021 年現在は, `haranoaji`, `bxjscls` `luatex-ja` といった LaTeX パッケージが日本語文書の作成に広く使われます. 既に書かれているように, **tinytex** はかなりの精度で必要なパッケージを自動でインストールしてくれますが, インストール済みの TeX を使用する場合は **tinytex** を使わず手動でインストールする必要があるかもしれません. 上記の `tinytex::tlmgr_install()` 関数は `tlmgr` のコマンドを実行するための関数なので, **tinytex** を使用していない環境では `tlmgr install ...` を代わりに実行することになります.

PDF 出力のためのセットアップは翻訳者が独自に書いた補足資料 https://rpubs.com/ktgrstsh/755893 も参考になるかもしれません.

第2章

コンセプトについての概論

このテキストの目標は R Markdown を最大限活用するために多くの豆知識と小ワザを見せることです. 以降の各章ではより効率的で簡潔なコードを書き, 出力をカスタマイズする技術を実演します. これを始める前に, これらを理解し, 覚え, 応用し,「リミックス」できる助けになるよう, R Markdown の動作がどうなっているかを少しだけ学んでおくと役に立つでしょう. この節では文書を knit する処理と出力を変更する「重要な切り替えレバー」について簡潔に概観します. この資料は後に続く章の内容理解に必要ではありません. 読み飛ばすのは自由です. しかし全てのピースをどう当てはめるかについて, より豊かなメンタルモデルを構築する助けになるかもしれません.

2.1 レンダリング時に何が起こっているのか

R Markdown はいくつかの異なるプロセスを合わせて文書を作成しています. これが R Markdown の全てのパーツがどう連動してるか理解するのに混乱する主な元凶です.[*1] 幸運にも, ユーザーが文書を作成できるようになるためにはこれらの処理の内部の挙動を全て理解することは必須ではありません. しかし, 文書の挙動を変えようとするだろうユーザーにとっては, どのパーツがどの挙動を担当しているかを理解することは重要です. あなたが検索する適切な範囲を絞れるようになれば, ヘルプを探すのがより簡単になります.

R Markdown 文書に対する基本的なワークフローの構造を図 2.1 に示します. ステップ (矢印) と, 出力ファイルが生成される前に作成される中間ファイルを強調しています. 全ての処理は `rmarkdown::render()` 関数内に実装されています. 以降は各ステップを詳細に説明します.

`.Rmd` 文書は, 文書の本来の形式です. YAML (メタデータ), テキスト (ナラティブ), コードチャンク を含んでいます.

[*1] Allison Horst が R Markdown の処理を魔法になぞらえたすばらしいイラストに描き出してくれました (https://github.com/allisonhorst/stats-illustrations/raw/master/rstats-artwork/rmarkdown_wizards.png). そして, この絵はまさに本書の扉絵に使われました.

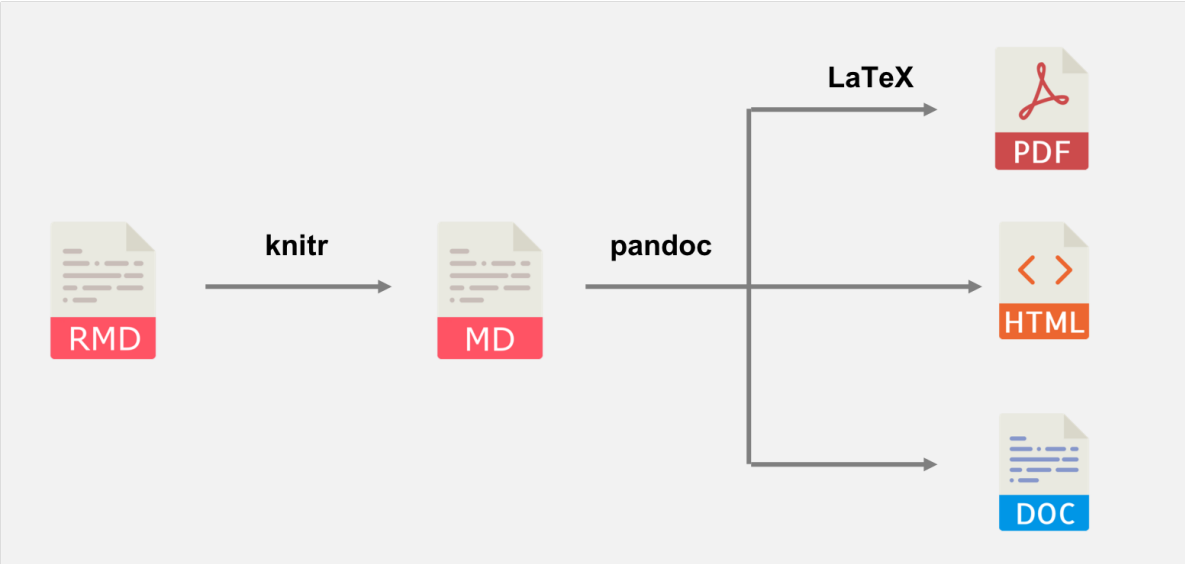


図 2.1: R Markdown 文書がどのように最終的な出力文書に変換されるかを表すダイアグラム

最初に **knitr** (Xie, 2021c) の `knit()` 関数が `.Rmd` ファイルに埋め込まれた全てのコードを実行し, 出力文書に出力コードを表示します. 全ての結果は, 一時的に作られた `.md` ファイルに含まれるよう, 適正なマークアップ言語へと変換されます.

その後 `.md` ファイルは, あるマークアップ言語のファイルから別のものへと変換するための多用途なツールである Pandoc によって処理されます. Pandoc は文書の YAML フロントマターで指定された何らかのパラメータ (例: `title`, `author`, `date`) を受け取り, 文書を `output` パラメータで指定された出力フォーマット (HTML へ出力する `html_document` のような) へ変換します.

出力フォーマットが PDF ならば, さらに処理が 1 層追加され, Pandoc が中間ファイルの `.md` をもう一つの中間ファイル `.tex` に変換します. このファイルはその後, 最終的な PDF 文書を形成するため LaTeX によって処理されます. 1.2 節で話したように, **rmarkdown** パッケージは **tinytex** パッケージ (Xie, 2021e) の `latexmk()` 関数を呼び出し, これが次々に LaTeX を呼び出して `.tex` をコンパイルし `.pdf` にします.

簡潔にまとめると, `rmarkdown::render()` = `knitr::knit()` + Pandoc (PDF の場合のみ + LaTeX) ということです.

Robin Linacre が `https://stackoverflow.com/q/40563479/559676` で R Markdown と **knitr** と Pandoc の関係について良い要約を書いてくれました. この投稿には上記の概観よりも技術的に細かい話も含まれています.

全ての R Markdown 文書が常に Pandoc を通してコンパイルされるわけではないことに注意してください. 中間ファイル `.md` は他の Markdown レンダラによってもコンパイルできます. 例えば 2 つ例を挙げます.

- **xaringan** パッケージ (Xie, 2021f) は出力された `.md` をウェブブラウザ上で Markdown コンテンツを表示するための JavaScript ライブラリに渡します.[*2]
- **blogdown** パッケージは (Xie Dervieux, and Presmanes Hill, 2021) `.Rmarkdown` 文書形式をサポートし, `.Rmarkdown` を knit して `.markdown` にします. 通常であれば, Markdown 文書は外部のサイトジェネレータによって HTML にレンダリングされます.

2.2 R Markdown の解剖学

R Markdown にあるいくつかの部品を考慮することで, 我々はさらに 1 レベル深く掘り下げることができます. では, この部品がレンダリング作業中にいつどのように処理を変化させるかに注目してみましょう.

2.2.1 YAML メタデータ

YAML metadata (YAML ヘッダとも呼びます) はレンダリング作業中の多くのステージで処理され, 様々な形で最終的な文書に作用することができます. YAML メタデータは Pandoc, **rmarkdown**, そして **knitr** のそれぞれで読み込まれます. その過程で, YAML メタデータに含まれる情報は, コード, コンテンツ, そしてレンダリング処理に影響をあたえます.

典型的な YAML ヘッダは以下のような形をしており, 文書とレンダリング操作指示の基礎となるメタデータを含んでいます.

```
---
title: My R Markdown Report
author: Yihui Xie
output: html_document
---
```

上記の場合, `title`, `author` フィールドは Pandoc によって処理され, テンプレートの変数の値が設定されます. デフォルトのテンプレートでは, `title` と `author` の情報は得られた文書の冒頭に現れます. Pandoc が YAML ヘッダの情報をどう扱うかのより詳細な話は, Pandoc マニュアルの

[*2] **訳注**: **xaringan** について日本語で言及している例は少ないですが, 次のページが用法・技術的な説明の両面で優れています. https://qiita.com/nozma/items/21c56c7319e4fefceb79

YAML metadata block.[*3] に関するセクションで見られます.[*4]

対照的に output フィールドは **rmarkdown** によるレンダリング処理中に出力フォーマット関数 rmarkdown::html_document() に適用されます. output に指定した出力フォーマットに引数を与えることで, 以降のレンダリング処理に影響させることができます. 例えばこのように書きます.

```
output:
  html_document:
    toc: true
    toc_float: true
```

これは rmarkdown::render() に, rmarkdown::html_document(toc = TRUE, toc_float = TRUE) と指示することと同じです. これらのオプションが何をするのか知るために, あるいは使える他のオプションを知るためには, R コンソールで ?rmarkdown::html_document を実行してヘルプページを読むとよいでしょう. output: html_document は output: rmarkdown::html_document と等価であることに注意してください. 出力フォーマットが rmarkdown:: のような修飾子を持たない場合, R Makrodown はこれを **rmarkdown** パッケージ由来のものと想定します. そうでないなら, R パッケージ名のプレフィックスが必要です. 例えば bookdown::html_document2 のような.

17.4 節に書いたように, YAML ヘッダ内でパラメータを選択したのなら, コンテンツとコードにも影響することができます. 簡潔に言うと, R Markdown ドキュメント全体で参照可能な変数や R 評価式をヘッダに含めることができるということです. 例えば以下のヘッダでは start_date と end_date パラメータを定義することで, 以降の R Markdown 文書内で params というリスト内に反映されます. つまり, R コード内でこれらの値を使いたければ, params$start_date と params$end_date でアクセスできるということです.

```
---
title: My RMarkdown
author: Yihui Xie
output: html_document
params:
  start_date: '2020-01-01'
```

[*3] https://pandoc.org/MANUAL.html#extension-yaml_metadata_block

[*4] **訳注**: 日本語訳での対応箇所はこちら: https://pandoc-doc-ja.readthedocs.io/ja/latest/users-guide.html#metadata-blocks

```
  end_date: '2020-06-01'
---
```

2.2.2 ナラティブ

ナラティブ (物語術) としてのテキスト要素は YAML メタデータやコードチャンクよりは理解が簡単でしょう．典型的には，これはテキストエディタで書いているような感覚でしょう．しかし Markdown コンテンツは，どのようにコンテンツが作られるか，そこからどうやって文書の構造が作られるか，の両方に関して，単純なテキストよりも強力で面白いものに違いありません．

世のナラティブの多くは人の手で書かれていますが，多くの R Markdown 文書では，用いられているコードと分析を参照しようとします．この理由として，4 章において，コードがテキストの一部を生成するのを助ける様々な方法が実演されています．例えば単語を結合してリストにしたり (4.11 節)，参考文献リストを書いたり (4.5 節) といった方法です．この変換は `.Rmd` から `.md` への変換と同様に **knitr** で制御されます．

Markdown のテキストは文書に構造を与えることもできます．Markdown の構文をこの場で復習するには紙面が足りませんが，[*5] 特に関連深い概念の 1 つとしてセクション見出しがあります．これは 1 つ以上の，対応したレベルの数のハッシュ (#) で表現されます．例えば，以下のように．

```
# 第 1 水準の見出し

## 第 2 水準の見出し

### 第 3 水準の見出し
```

これらの見出しは **rmarkdown** が `.md` を最終的な出力フォーマットに変換する際に文書全体に構造を与えます．この構造は，ある属性を付与することで章や節を参照し整形するのに役立ちます．例えば以下のように，Pandoc 構文は見出しの記述に `{#id}` の表記にしたがってユニークな識別子をつけることで参照が作成できますし，`{.クラス名}` のように書くことで，セクションに一つないし複数のクラスを付与できます．

[*5] Markdown の復習には，代わりに，https://bookdown.org/yihui/bookdown/markdown-syntax.html をご覧になってください．

```
## 第 2 水準の見出し {#introduction .important .large}
```

これから学ぶいくつもの方法で, 例えば ID やクラスを参照することで, このセクションにアクセスすることができるようになります. 具体例として, 4.7 節ではセクション ID を使って文書内のどこでも相互参照する方法を実演していますし, 7.6 節では小節を再構成させる `.tabset` クラスを紹介しています.

R Markdown のテキスト部分で見られる最後の興味深いコンテンツのタイプとして, 特定の出力したいフォーマットに対して「生のコンテンツ」をそのまま書き出す方法, 例えば LaTeX 出力に対して LaTeX コードを直接書く (6.11 節), HTML 出力に対して HTML コードを直接書く (9.5 節), 等を挙げます. 生のコンテンツは基本的な Markdown ではできないことが達成できますが, 出力フォーマットが異なるとたいていは無視されてしまうことに留意してください. 例えば生の LaTeX コマンドは出力が HTML の場合, 無視されます.

2.2.3 コードチャンク

コードチャンクは R Markdown にとっての心臓の鼓動です. チャンク内のコードは **knitr** によって実行され, 出力は Markdown に翻訳され, レポートは現在のスクリプトとデータに動的に同期します. 各コードチャンクは言語エンジン (15 章), 任意に指定できるラベル, チャンクオプション (11 章), そしてコードで構成されます.

コードチャンクに対してできるいくつかの修正について理解するためには, **knitr** の処理を少しだけ詳しく知ることが有意義です. 各チャンクでは, **knitr** 言語エンジンは 3 つの入力の部品を得ます. knit 環境 (`knitr::knit_global()`), 入力されたコード, そしてチャンクオプションのリストです. コードチャンクはコードの出力とともに見た目も整形して返します. 副作用として, knit 環境も修正されます. 例えばコードチャンク内のソースコードを介してこの環境内で新しい変数がつくられます. この処理は図 2.2 のように表せます.

この処理は以下の方法で修正できます.

- 言語エンジンを変更する
- チャンクオプションを, グローバル, ローカル, あるいは言語エンジンに特定のものに修正する
- フックやチャプターを使用して, 入出力にさらなる処理を追加する

例えば 12.1 節で, 後処理をするフックを加えてソースコードの特定行を編集する方法を学べるでしょう.

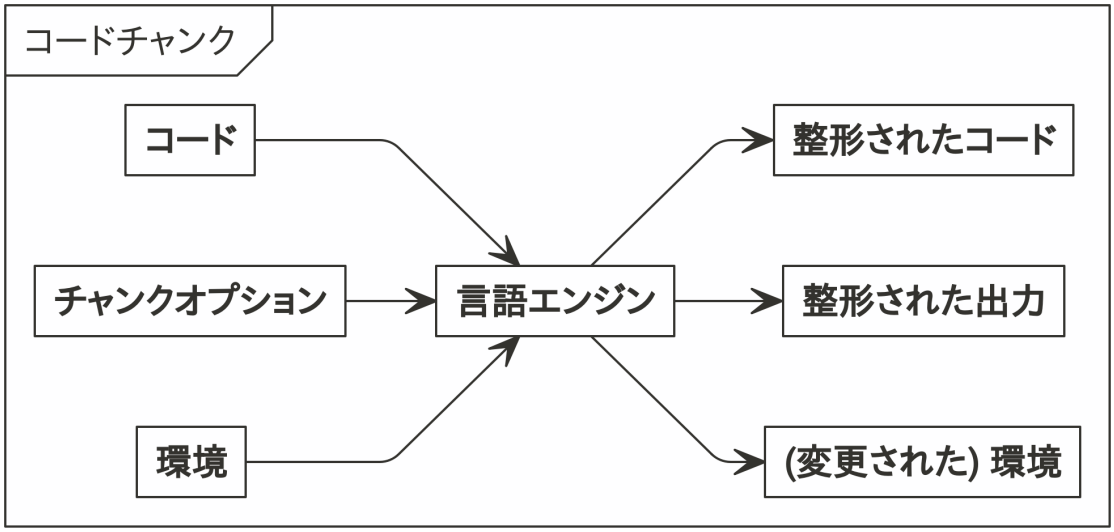


図 2.2: 言語エンジンへの入出力フローチャート

コードチャンクには 2.2.2 節でつぶさに見てきたナラティブのクラスと識別子に類似するコンセプトがあります. コードチャンクは識別子 (よく「チャンクラベル」と呼ばれます) を言語エンジンの直後に任意で指定することができます. チャンクオプション `class.source` と `class.output` でそれぞれコードブロックとテキスト出力ブロックに対するクラスを設定することもできます (7.3 節参照). 例えばチャンクヘッダ ```` ```{r summary-stats, class.output = 'large-text'} ```` はチャンクラベルに `summary-stats` を与え, テキスト出力ブロックに `large-text` というクラスを与えています. チャンクのラベルは 1 つだけですが, クラスは複数持つことができます.

2.2.4 文書の本文

文書を執筆し編集する際に理解すべき重要なことは, コードとナラティブの小片が文書内のいくつもの節やコンテナを作る方法です. 例えばこのような文書があったとします.

```
# タイトル

## X 節

ここから導入文
```

````
```{r chunk-x}
x <- 1
print(x)
```

### 第 1 小節

ここに詳細な話

### 第 2 小節

ここにさらに詳しい話

## Y 節

ここから新しい節

```{r chunk-y}
y <- 2
print(y)
```
````

この文書を書いていると, それぞれの小片は, テキストとコードを含む独立した節を一直線上に並べたものだとみなせるでしょう. しかし我々が実際にしているのは, 概念としては図 2.3 でより細かく描いているように, 入れ子 (ネスト) になったコンテナの作成です[*6]

この図に関する 2 つの重要な特徴は (1) テキストやコードのどのセクションも個別のコンテナであり, (2) コンテナは他の別のコンテナを入れ子にできる, ということです. この入れ子は R Markdown 文書を RStudio IDE で執筆し, 文書のアウトラインを展開[*7]しているとはっきりと分かります.

図 2.3 では同じレベルのヘッダは同じレベルの入れ子を表していることに注意してください. より低いレベルのヘッダはより高レベルなヘッダのコンテナ内部にあります. この場合, 通常は高レベ

[*6] 実際にはここで見えているよりも多くのコンテナがあります. 例えば knit されたコードチャンクや, コードと出力がそれぞれ別のコンテナとして存在し, そしてこれらは親要素を共有しています.

[*7] **訳注**: エディタ右上にあるボタンで表示を切り替えることができます.

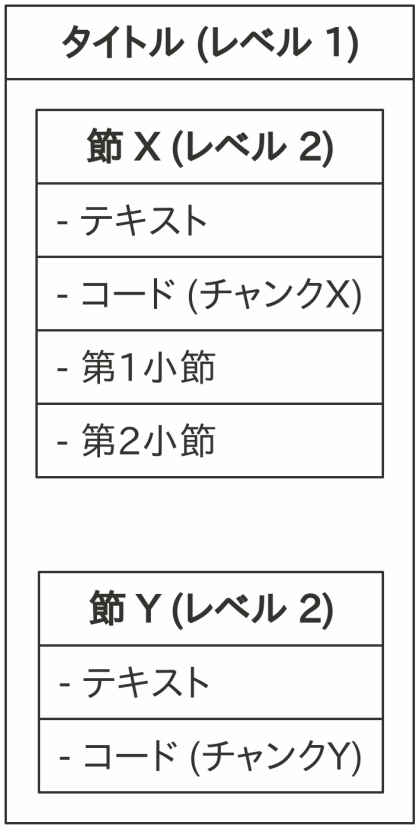


図 2.3: 入れ子状のコンテナとして表現された単純な R Markdown 文書の例

ルの節を「親」といい, 低レベルの節を「子」といいます. 例えば「小節」は「節」の子です. 5.8 節で実演するように, ヘッダだけでなく ::: を使ってまとまりの単位を作ることができます.

このテキストで説明されているフォーマットやスタイルのオプションを適用するとき, この構造は重要な意味を持ちます. 例えば, Pandoc が抽象構文木 (AST) でどのように文書を表現するかを我々が学ぶ時 (4.20 節) や, HTML 出力のスタイルを決めるために CSS セレクタを使用する時 (7.1 節ほか), 入れ子構造の概念が現れます.

フォーマットとスタイルは類似するいずれのタイプのコンテナ (例えばコードブロック) や, あるコンテナの内部にある全てのコンテナ (例:「Y 節」以下にある全てのコンテナ) に対して適用できます. 加えて 2.2.2 節で説明したように, 同一のクラスを特定の節に対して適用し同様のものとして扱うよう明示できますが, この場合の共通するクラス名は, 節と節に共通のプロパティや共通の意図を示すようになります.

本書を読みながら, 特定の「レシピ」がどんな種類のコンテナに対して作用しているのかを自問し思いを馳せることは役に立つでしょう.

2.3 結果を変えるために変更できるのはなにか?

では, ここまでで概観してきたものを要約し, これから何をすべきかを下見していきましょう.

rmarkdown で R Markdown 文書をレンダリングする処理は **knitr** で `.Rmd` を `.md` で変換する処理, それから (典型的には) Pandoc で `.md` を望む出力に変換する処理で構成されます.

`.Rmd` から `.md` への変換のステップではレポート内の全てのコードの実行と「翻訳」を取り扱うことから, ここでの「コンテンツ」への変更はほぼ, `.Rmd` のうち **knitr** が翻訳するコードを編集する作業に絡んできます. これらのステップ全体を操作するツールには **knitr** チャンクオプションおよびフックがあります.

`.md` はフォーマットされていないプレーンテキストです. ここで Pandoc が登場し, HTML や PDF, Word といった最終的な出力フォーマットへ変換されます. この流れに沿って構造とスタイルを付与します. この処理では スタイルシート (CSS), 生 LaTeX または HTML コード, Pandoc テンプレート, Lua フィルタといった様々なツールが助けになります. R Markdown 文書の入れ子構造を理解し, よく考えて識別子とクラスを使うことで, これらのツールを取捨選択して出力の目標となる箇所に応用することができます.

最後に, YAML メタデータはこれらのステップの切り替えに役に立つことでしょう. パラメータを変更することでコードがどう動作するかを変更でき, メタデータを変更すればテキストの内容を変化させ, 出力オプションの変更は異なる命令のセットを備える `render()` 関数を与えます.

もちろんこれらは全て大まかな経験則であり, 絶対的な事実として扱うべきでありません. 究極的には, 機能を完璧にきれいに分類することはできません. 本書全体を通じて, 説明されている結果の多くは, しばしば実現までの道筋が複数あり, さらにそのパイプラインの様々なステージの説明に立ち入ることになることが分かるでしょう. 例えば文書内に画像を挿入する作業では, `.Rmd` から `.md` への変換の段階で R コード `knitr::include_graphics()` を使うこともあれば, Markdown 構文 (`![]()`) を直接使うこともあるでしょう. ややこしく思えるかもしれませんし, アプローチごとに異なる利点を持つこともあります. しかし悩まないでください. なんにせよ, あなたの問題を解決する多くの有効な方法が存在し, あなたはその中から自分にとって最も理にかなうアプローチに従うことができます.

さあこの辺にしておきましょう. 本書の残りの部分で, これまで議論してきた R Markdown を最大限活用するための, あらゆるコンポネントを変更する方法のより具体的な例を使って, あなたの絵の下書きに色をつけることができます.

第3章

基本

この章では, R Markdown の重要な概念をいくつか提示します. まず「テキスト」「コード」という R Markdown の基本的なコンポネントを紹介します. 次に, R Markdown 文書をどうやって R スクリプトへ変換するか, あるいは逆の変換はどうやるかを提示します.

もっと基本的な話を求める方は, *R Markdown Definitive Guide* (Xie J.J. Allaire, and Grolemund, 2018) の 2 章を見てください.

3.1 コードチャンクとインライン R コード

R Markdown 文書はテキスト (ナラティブ) とコードが混合してできています. Rmd 文書には 2 種類のコード, コードチャンクとインライン (行内) R コードです. 以下は簡単な例です.

````
```{r}
x <- 5  # 円の半径
```

半径 `r x` の円に対し,
その面積は `r pi * x^2` である.
````

通常コードチャンクは ```{} で始まり, ``` で終わります. コードチャンク内ではコードを何行でも書いてかまいません. インライン R コードは `r ` という構文を使って文書のナラティブの中に埋め込まれます. 上記の例ではコードチャンク内で円の半径として変数 x を定義し, 次のパラグラフでこの円の面積を計算しています.

チャンクオプションを通してコードチャンクの挙動と出力をカスタマイズできます (オプションはカーリー・ブレイス {} 内に与えます). 例のいくつかは 11 章で見つかるでしょう. コードチャンクに別のプログラミング言語のコードを書くこともできます (15 章を見てください).

3.2 RStudio のビジュアルエディタで R Markdown を書く

あなたがまだ Markdown の書き方に慣れていないか, Markdown コードを書きたくなければ, RStudio ver. 1.4 には実験的ですが Markdown 文書用のビジュアルエディタがあります. これは図 3.1 で示すように Word のような伝統的な WYSIWYG なエディタと似ていると感じるでしょう. この完全なドキュメントは https://rstudio.github.io/visual-markdown-editing/ で見ることができます.

ビジュアルエディタによって, ヘッダ, 図, 表, 脚注などといった Pandoc でサポートされているほとんどあらゆる Markdown 要素を視覚的に編集できます. あなたは全ての構文を覚えなくてもよいのです. ある要素の構文を忘れた場合, RStudio ツールバー (図 3.1 参照) を使うかキーボードショートカットを使って, 要素を追加したり編集したりできます.

既に Markdown に熟練しているなら, ツールバーの一番右端のボタンを右クリックしてソースモードとビジュアルモードを切り替えられるので, 文書をソースモードのままで書くこともできます.

3.3 R スクリプトをレポートにレンダリングする

長らく RMarkdown を使っていても, 別の選択肢があることを見落としていることがあります. Dean Attali はこれを "**knitr** の秘宝[*1]と読んでいます. 純粋な R スクリプトを直接レンダリングできるということです. RStudio IDE をお使いなら, R スクリプトをレンダリングするキーボードショートカットは Rmd 文書を knit するときと同じ (`Ctrl / Cmd + Shift + K`) です.

R スクリプトをレポートにレンダリングすると, まず `knitr::spin()` 関数が呼ばれスクリプトが Rmd ファイルに変換されます. この関数こそ Dean Attali が「**knitr** の秘宝」と呼んでいるものです. レポートには全てのテキストとグラフィックの出力が掲載されます.

レポートの要素を細かく管理したいなら, 以下のような構文が役に立ちます.

- Roxygen コメントはテキストとして扱われます. roxygen コメントは `#'` で始まる R のコメントで, レポートにナラティブを書くのに役立ちます. コメント内ではあらゆる Markdown 構文を使うことができます.

[*1] https://deanattali.com/2015/03/24/knitrs-best-hidden-gem-spin/

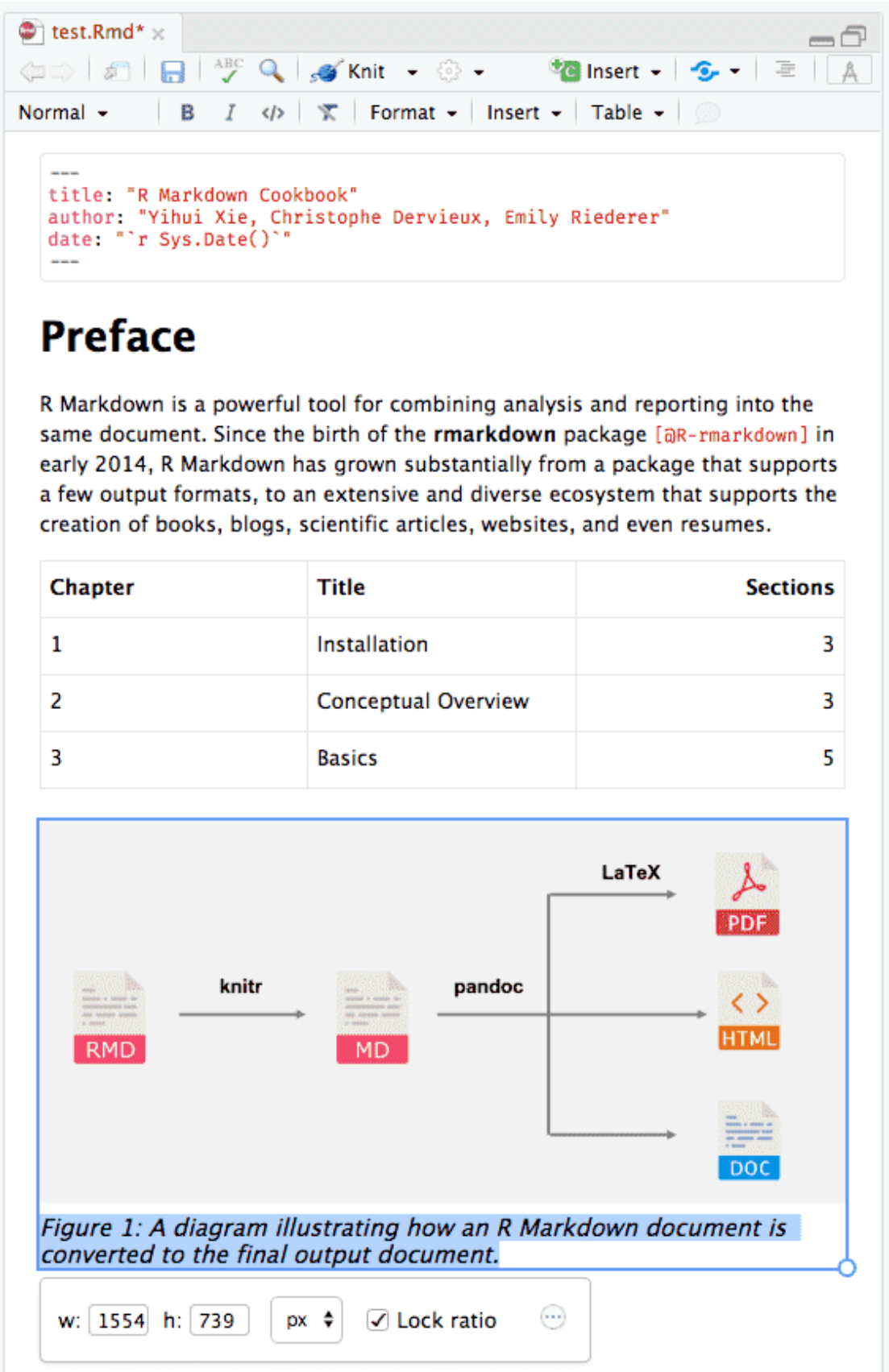


図 3.1: RStudio のビジュアル Markdown エディタ

- `#+` で始まるコメントは `knitr` のチャンクヘッダとして扱われます．例えば，`knitr::spin()` は `#+ label, fig.width=5` というコメントを，R Markdown の ```` ```{r label, fig.width=5} ```` というチャンクヘッダへ翻訳します．
- `{{ code }}` で囲まれた R コードは R Markdown のインライン R コードへ翻訳されます．`{{ code }}` は 1 行で書かなければならないことに注意してください．
- YAML フロントマターも，R スクリプトの冒頭の roxigen コメント内に書くことができます．YAML フィールドのインデントには，特に気をつけてください．これはとても大事なところです．YAML のインデントを省くと YAML に記述したデータ構造は別の正しくないものになります．例えば `keep_tex: true` というフィールドは，後の例のように `pdf_document` 以下に 2 つ以上のスペースでインデントするべきです．
- `/*` と `*/` の間の任意のテキストは無視されます (つまり，完全にコメントとして扱われます)

上記のルールの全ての例を表現したのが以下です．

```
#' ---
#' title: "純粋な R script から生成したレポート"
#' output:
#'   pdf_document:
#'     keep_tex: true
#' ---
#'
#' これは `knitr::spin()` によって生成されたレポートです．
#'
#' **knitr** オプションをいくつか試してみましょう:

#+ echo=FALSE, fig.width=7
#  これは通常の R コメント文です．
plot(cars)

#' ここにインラインの値を書きましょう．$\pi$ の値は
{{ pi }}
#' であると知られています．
#'
#' 最後に，全ての roxygen コメントは任意だということを書いておきます．
#' プロットの大きさなど出力要素をコントロールしようと思わない限り
#' チャンクオプションも必要でありません
```

```
# /* C 言語のコメントのように /* と */ の間にコメントを書きましょう:
Sys.sleep(60)
# */
```

このスクリプトがレポートにレンダリングされた時, knitr::spin() はこれを R Markdown へと変換します.

```
---
title: "純粋な R script から生成したレポート"
output:
  pdf_document:
    keep_tex: true
---

これは `knitr::spin()` によって生成されたレポートです.

**knitr** オプションをいくつか試してみましょう:

```{r echo=FALSE, fig.width=7}
#  これは通常の R コメント文です.
plot(cars)
```

ここにインラインの値を書きましょう. π の値は
``r  pi  ``
であると知られています.

最後に, 全ての roxygen コメントは任意だということを書いておきます.
プロットの大きさなど出力要素をコントロールしようと思わない限り
チャンクオプションも必要でありません
```

このレポート生成方法は, 主に R スクリプトを使って作業していて多くのナラティブを必要としないときに, 特に役立つでしょう. レポートの中でテキストの割合が高いなら, 全てのテキストを

roxygen コメントに入れなくてもいい R Markdown がより良い選択でしょう.

3.4 Markdown から R script への変換

R Markdown から全ての R コードを取り出したい時は, `knitr::purl()` 関数を呼ぶことができます. 以下は `purl.Rmd` というファイル名の簡単な Rmd の例です.

````
---
title: R コードを取りだすために `purl()` を使いましょう
---

`knitr::purl()` 関数は **knitr** 文書から R コードチャンクを取り出しコードを R スクリプ
↪ トに保存します.

以下は簡単なチャンクです.

```{r, simple, echo=TRUE}
1 + 1
```

`r 2 * pi` のようなインライン R 式はデフォルトでは無視されます.

特定のコードチャンクを取り出してほしくない場合は，以下の例のようにチャンクオプション
↪ `purl = FALSE` を設定できます.

```{r, ignored, purl=FALSE}
x = rnorm(1000)
```
````

`knitr::purl("purl.Rmd")` を呼び出したら, 以下の R スクリプト (デフォルトのファイル名は `purl.R`) が生成されます.

```
## ---- simple, echo=TRUE-----------------------------
1 + 1
```

上記の R スクリプトでは, チャンクオプションがコメントとして書かれています. 純粋な R コードが欲しい場合, knitr::purl() を documentation = 0 という引数を与えて呼べば, 以下のような R スクリプトが生成されます.

```
1 + 1
```

テキストを全て残したいときは documentation = 2 引数を使えば, 以下のような R スクリプトを生成します.

```
#' ---
#' title: R コードを取りだすために `purl()` を使いましょう
#' ---
#'
#' `knitr::purl()` 関数は **knitr** 文書から  R コードチャンクを取り出しコードを R スクリプトに保存します.
#'
#' 以下は簡単なチャンクです.
#'
## ---- simple, echo=TRUE-----------------------------
1 + 1

#'
#' `r 2 * pi` のようなインライン R 式はデフォルトでは無視されます.
#'
#' 特定のコードチャンクを取り出してほしくない場合は, 以下の例のようにチャンクオプション `purl = FALSE` を設定できます.
#'
```

purl = FALSE というオプションのあるコードチャンクは R スクリプトから除外されることに注意してください.

インライン R コードはデフォルトでは無視されます. R スクリプトにインライン表現も含めたい

なら, `knitr::purl()` を呼ぶ前に R のグローバルオプション `options(knitr.purl.inline = TRUE)` を設定する必要があります.

3.5 R Markdown Notebook

R Markdown Definitive Guide (Xie J.J. Allaire, and Grolemund, 2018) の Section 2.2[*2] で言及したように, Rmd 文書をコンパイルする方法はいくつかあります. その 1 つは次の例のように `html_notebook` という出力フォーマットで R Markdown Notebook を使うことです.

```
---
title: An R Markdown Notebook
output: html_notebook
---
```

RStudio でこの出力フォーマットを使うと, ツールバー上の `Knit` ボタンが `Preview` ボタンになります.

notebook を使う主な利点は Rmd 文書を**同じ R セッションで繰り返し**作業できることです. コードチャンクにある緑色の矢印ボタンを押せばチャンクを個別に随時実行でき, エディタ上でテキストやグラフの出力を見られます. ツールバー上の `Preview` ボタンを押すと, Rmd 文書を既に実行したコードチャンクの出力を含む HTML 文書へレンダリングするだけです. `Preview` ボタンは一切のコードチャンクを実行しません. これと比較して, 他の出力フォーマットを使い `knit` ボタンを押したときには, RStudio は文書全体をコンパイルする (つまり全てのコードチャンクが一気に実行されます) ために R セッションを新規で立ち上げますので, たいていはもっと時間がかかります.

コードチャンクを個別に実行した時に出力がインライン表示されるという RStudio のデフォルトの挙動が気に入らないなら, `Tools -> Global Options -> R Markdown` から “Show output inline for all R Markdown documents” というオプションのチェックを外すことができます. 以降, コードチャンクを実行すると出力はソースエディタ内ではなく R コンソールに表示されます. このオプションは以下のように YAML メタデータで個別の Rmd 文書ごとに設定することもできます.

```
editor_options:
  chunk_output_type: console
```

[*2] https://bookdown.org/yihui/rmarkdown/compile.html

第4章

文書の要素

本章では, 改ページ, YAML メタデータ, セクションヘッダ, 引用, 相互参照, 数式, アニメーション, 対話的プロット, ダイアグラム, コメントといった R Markdown 文書の要素をカスタマイズしたり生成したりするのに使える豆知識と小ワザを紹介します.

4.1 改ページ (改段) を挿入する

改ページしたい場合, \newpage を文書に挿入できます.[*1] これは LaTeX コマンドですが, **rmarkdown** パッケージは LaTeX 出力フォーマットでも, 以下のような HTML, Word, ODT などのいくつかの非 LaTeX 出力フォーマットでも認識することができます.[*2]

```
---
title: Breaking pages
output:
  pdf_document: default
  word_document: default
  html_document: default
  odt_document: default
---

# 第一節
```

[*1] **訳注**: 正確には \newpage コマンドは改「段」です. 二段組の場合, 次の段に改めるため, 必ずページを改めるわけではありません.

[*2] HTML 出力では, 改ページは HTML ページの印刷時のみ意味をなし, それ以外では HTML は単一の連続したページになるため, 改ページを見ることはありません.

```
\newpage

# 第二節
```

これは Pandoc の Lua フィルタ に基づく機能です (4.20 節参照). 技術的なことに興味のある方はこのパッケージのビネットを見てください.

```
vignette("lua-filters", package = "rmarkdown")
```

4.2 文書タイトルを動的に設定する

インライン R コード (3.1 節) は, Rmd 文書内のどこでも, YAML メタデータの部分であっても, 使うことができます. つまり次の例のように, インライン R コードによって文書のタイトルなどの YAML メタデータを動的に生成できるということです.

```
---
title: "自動車 `r nrow(mtcars)` 台の分析"
---
```

文書タイトルが後の文書内で作成される R の変数に依存する場合, 以下の例のように変数の後にくる YAML セクションに `title` フィールドを書き加えることができます.

```
---
author: "利口なアナリスト"
output: pdf_document
---

我々の市場シェアを頑張って計算してみました.
```

```
```{r}
share <- runif(1)
```


---
title: "我々の市場シェアは今や `r round(100 * share, 2)`% です!"

これはとても `r if(share > 0.8) "喜ばしい" else "悲しい"` ことです.
```

上記の例では, 変数 share を生成してから文書のタイトルを追加しています. このような場合であってもうまくいくのは, Pandoc は文書内に YAML セクションをいくつ書いても読み込み, そして全てをマージすることができるためです.

タイトルだけでなくどの YAML フィールドも, パラメータ化されたレポートから動的に生成することができます (17.4 節参照). 例えばこのように.

```
---
title: "`r params$doc_title`"
author: "利口なアナリスト"
params:
  doc_title: "デフォルトのタイトル"
---
```

タイトルを動的なパラメータにしておくと, タイトルだけ異なるレポートを簡単に一括で生成できます.

この節ではタイトルを例にしましたが, このアイディアは YAML セクションのどのメタデータのフィールドにも適用可能です.

4.3 R コード内で文書メタデータにアクセスする

Rmd 文書をコンパイルする際には, YAML セクションの全てのメタデータはリストオブジェクト rmarkdown::metadata に格納されます. 例えば rmarkdown::metadata$title には文書のタイトルが与えられます. この metadata オブジェクトは R コード内で使うことができるので, YAML メ

タデータに与えられた情報をハードコードしなくてすみます. 例えば以下のように **blastula** パッケージ (Richard Iannone and Cheng, 2020) で E メールを送る時, 文書のタイトルをメールの件名に, 著者フィールドを送信者情報に使うことができます.

```
---
title: 重要なレポート
author: John Doe
email: john@example.com
---

重要な分析ができましたので結果をメールで送りたいと思います.

```{r}
library(rmarkdown)
library(blastula)
smtp_send(
  ...,
  from = setNames(metadata$email, metadata$author),
  subject = metadata$title
)
```
```

4.4 番号のない節

ほとんどの出力フォーマットは `number_sections` オプションをサポートしています. これを `true` に設定すれば, 以下の例のように節への番号付けを有効にできます.

```
output:
  html_document:
    number_sections: true
  pdf_document:
    number_sections: true
```

特定の節に番号を付けたくないならば, number_sections オプションは true のままにして, その節のヘッダの直後に {-} を加えます. 例えばこのように.

```
# この節には番号がつきません {-}
```

全く同じことを, {.unnumbered} を使ってもできます. 例えば {.unnumbered #section-id} のように, 他の属性を追加することもできます. 詳細は https://pandoc.org/MANUAL.html#extension-header_attributes を確認してください.

付番されていない節は記述に特記情報を追加するのに使われます. 例えば本書では,「はじめに」と「著者について」の章は本文ではないため付番されていません. 図 4.1 を見ればわかるように, 実際の本文は番号の付いていない 2 つの章の後から始まり, 本文の章は付番されています.


目次

図 4.1: 付番された章とされていない章を示すための目次のスクリーンショット

節番号は 1 つづつ増えます. もし付番した節の後に付番されていない節を挿入し, その後さらに付番した節が始まると, 節番号は再び増加していきます.

4.5 参考文献と引用

参考文献目録を出力文書に含める方法の概要は, Xie (2016) の Section 2.8[*3] を見るとよいでしょう. 基本的な使用法として, YAML メタデータの `bibliography` フィールドに文献目録ファイルを指定する必要があります. 例えばこのようにします.

```
---
output: html_document
bibliography: references.bib
---
```

この BibTeX データベースは `*.bib` という拡張子の付いたプレーンテキストとして与えられ, ファイルに文献アイテムがこのようなエントリで含まれています.

```
@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 = {2019},
  url = {https://www.R-project.org},
}
```

文書内では `@key` という構文で文献アイテムを直接引用することができます. `key` 部分はエントリの最初の行にある引用キーのことです. 上記の例なら `@R-base` です. 括弧で囲んで引用したいなら, `[@key]` を使います. 複数のエントリを同時に引用するなら, `[@key-1; @key-2; @key-3]` のようにセミコロンでキーを区切ります. 著者名を表示しないのなら, `[-@R-base]` のように `@` の前にマイナス記号を付けます.

[*3] https://bookdown.org/yihui/bookdown/citations.html

4.5.1 引用スタイルの変更

Pandoc は Chicago 式の著者名-出版年形式の引用スタイルと参考文献スタイルをデフォルトで使います．他のスタイルを使うには，例えば例のように，メタデータフィールドの `csl` で CSL (Citation Style Language) ファイルを指定します．

```
---
output: html_document
bibliography: references.bib
csl: biomed-central.csl
---
```

必要としているフォーマットを見つけるには，Zotero Style Repository,[*4] を使うことをおすすめします．これは必要なスタイルの検索とダウンロードが簡単にできます．

CSL ファイルは個別のフォーマット要件に合うようにを修正できます．例えば “et al.” の前に表示する著者の人数を変更して短縮できます．これは https://editor.citationstyles.org にあるようなビジュアルエディタを使って簡単にできます．

4.5.2 引用していない文献を参考文献に追加する

デフォルトでは参考文献には文書で直接参照されたアイテムのみ表示されます．本文中に実際に引用されていない文献を含めたい場合，`notice` というダミーのメタデータフィールドを定義し，そこで引用します．

```
---
nocite: |
  @item1, @item2
---
```

[*4] https://www.zotero.org/styles

4.5.3 全てのアイテムを参考文献に掲載する

文献目録のすべてのアイテムを明示的に言及したくないが, 参考文献には掲載したいというなら, 以下のような構文が使えます.

```
---
nocite: '@*'
---
```

これは全てのアイテムを参考文献として強制的に掲載させます.

4.5.4 参考文献の後に補遺を掲載する (*)

デフォルトでは参考文献は文書全体の最後に掲載されます. しかし参考文献一覧の後に追加のテキストを置きたいこともあるでしょう. 一番よくあるのは文書に補遺 (appendix) を含めたいときです. 以下に示すように, `<div id="refs"></div>` を使うことで参考文献一覧の位置を強制変更できます.

```
# 参考文献

<div id="refs"></div>

# 補遺
```

`<div>` は HTML タグですが, この方法は PDF など他の出力フォーマットでも機能します.

さらによい方法としては以下の例のように **bookdown** パッケージ (Xie, 2021a) を使い, 補遺の開始前に special header[*5] `# (APPENDIX) Appendix {-}` が挿入できます.

```
# 参考文献
```

[*5] https://bookdown.org/yihui/bookdown/markdown-extensions-by-bookdown.html#special-headers

```
<div id="refs"></div>

# (APPENDIX) 補遺 {-}

# 追加情報

これは「補遺 A」になる.

# さらにもう 1 つ

これは「補遺 B」になる.
```

LaTeX/PDF および HTML フォーマットでは補遺の付番スタイルは自動的に変更されます (たいていは A, A.1, A.2, B, B.1, … という形式です).

4.6 R パッケージの引用を生成する

R パッケージを引用するには, base R の `citation()` を使うことができます. BibTeX 用の引用エントリを生成したいなら, `citation()` の返り値を `toBibtex()` を与えることができます. 例えばこうです.

```
01 toBibtex(citation("xaringan"))
```

```
@Manual{,
  title = {xaringan: Presentation Ninja},
  author = {Yihui Xie},
  year = {2021},
  note = {R package version 0.22},
  url = {https://CRAN.R-project.org/package=xaringan},
}
```

`toBibtex()` で生成されたエントリを使うには, 出力を `.bib` ファイルにコピーし, 引用キーを追加

しなければなりません (例えば `@Manual{`, の部分を `@Manual{R-xaringan`, と書き換える). これは `knitr::write_bib()` 関数によって自動化できます. この関数は引用エントリを生成し, 自動的にキーを加えてファイルに書き込みます. 例えばこのようにします.

```
knitr::write_bib(c(.packages(), "bookdown"), "packages.bib")
```

第 1 引数はパッケージ名の文字列ベクトルで, 第 2 引数は `.bib` ファイルのパスであるべきです. 上記の例では, `.packages()` は現在の R セッションが読み込んでいる全てのパッケージ名を返します. これらのパッケージのいずれかが更新された (例えば著者, タイトル, 年, あるいはバージョンが変更された) とき, `write_bib()` は自動的に `.bib` を更新できます.

引用エントリには 2 つのタイプが選択肢としてあります. 1 つはパッケージの `DESCRIPTION` ファイルをもとに生成したもので, もう 1 つは, もしパッケージに `CITATION` ファイルが存在するなら, そこから生成したものです. 前者のタイプは引用キーが `R-(`**パッケージ名**`)` という形式 (例えば `R-knitr`) になり, 後者のタイプはパッケージ名と公開年を結合したもの (例: `knitr2015`) がキーとなります . 同じ年に複数のエントリがあるときは, 接尾文字が追加されます. 例えば `knitr2015a` と `knitr2015b` のように. 前者のタイプはしばしばパッケージ自体を引用 (つまり, ソフトウェアとして) するのに使われますが, 後者のタイプは論文や書籍のようなパッケージと関連する出版物といったものが多いです.

```
knitr::write_bib(c("knitr", "rmarkdown"), width = 60)
```

```
@Manual{R-knitr,
  title = {knitr: A General-Purpose Package for Dynamic
    Report Generation in R},
  author = {Yihui Xie},
  year = {2021},
  note = {R package version 1.34},
  url = {https://yihui.org/knitr/},
}

@Manual{R-rmarkdown,
  title = {rmarkdown: Dynamic Documents for R},
```

```
  author = {JJ Allaire and Yihui Xie and Jonathan McPherson
    and Javier Luraschi and Kevin Ushey and Aron Atkins
    and Hadley Wickham and Joe Cheng and Winston Chang and
    Richard Iannone},
  year = {2021},
  note = {R package version 2.11},
  url = {https://CRAN.R-project.org/package=rmarkdown},
}

@Book{knitr2015,
  title = {Đynamic Đocuments 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
    Đ. 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},
}
```

```
@Book{rmarkdown2018,
  title = {R Markdown: The Definitive Guide},
  author = {Yihui Xie and J.J. Allaire and Garrett
    Grolemund},
  publisher = {Chapman and Hall/CRC},
  address = {Boca Raton, Florida},
  year = {2018},
  note = {ISBN 9781138359338},
  url = {https://bookdown.org/yihui/rmarkdown},
}

@Book{rmarkdown2020,
  title = {R Markdown Cookbook},
  author = {Yihui Xie and Christophe Dervieux and Emily
    Riederer},
  publisher = {Chapman and Hall/CRC},
  address = {Boca Raton, Florida},
  year = {2020},
  note = {ISBN 9780367563837},
  url = {https://bookdown.org/yihui/rmarkdown-cookbook},
}
```

ファイルパスの引数がないと, knitr::write_bib() は上記の例のように引用エントリをコンソールに出力します.

write_bib() は既存の文献目録ファイルを上書きするように設計されていることに注意してください. 文献目録に手動で他のエントリを追加したい場合, 2 つ目の .bib ファイルを作成して, この例のように bibliography フィールドで参照してください.

```
---
bibliography: [packages.bib, references.bib]
---

```{r, include=FALSE}
knitr::write_bib(file = 'packages.bib')
```
```

```
```

上記の例では packages.bib は自動で生成されたものなので, 手動で変更すべきではありません. それ以外の全ての引用エントリは references.bib に手動で書き込むことができます.

ここまでは R パッケージの引用を生成する方法を 1 つだけ紹介しています. それ以外のタイプの文献で動的に引用を生成するには, **knitcitations** パッケージ (Boettiger, 2021) を見てください.

4.7 文書内の相互参照

相互参照 はあなたの文書を通して読者を誘導するのに役に立つ方法であり, R Markdown ではこれを自動的に行なえます. これは **bookdown** 本の Chapter 2[*6] で既に説明されていますが, 以下で簡潔な説明をします.

相互参照を使用するにあたって, 以下が必要になります.

- **bookdown 出力フォーマット**: 相互参照は基本となる **rmarkdown** パッケージでは直接提供されず, **bookdown** (Xie, 2021a) による拡張機能として提供されています. よって YAML の output フィールドで **bookdown** のフォーマット (例: html_document2, pdf_document2, word_document2 など) を使用しなければなりません.
- **図 (または表) に対するキャプション**: キャプションのない図は単なる画像として直接埋め込まれるためめ, 付番された図 (figure) にはなりません.
- **ラベルの設定されたコードチャンク**: チャンクが生成した図を参照する識別子を提供してくれます.

これらの条件が整って初めて, テキスト内で \@ref(type:label) という構文を使って相互参照を作成できます. label はチャンクラベルであり, type は参照される環境 (例:, tab, fig, eqn) です. 以下に例を示します.

```
---
title: 図，表，数式を相互参照する
author: bookdown による生成
output:
  bookdown::pdf_document2:
```

[*6] https://bookdown.org/yihui/bookdown/components.html

```
    latex_engine: lualatex
  bookdown::html_document2: default
documentclass: ltjsarticle
---

図 \@ref(fig:cars-plot) を見よ.

```{r cars-plot, fig.cap="自動車のデータ", echo=FALSE}
par(mar = c(4, 4, .2, .1))
plot(cars)  # a scatterplot
```

次に数式 \@ref(eq:mean) を見よ.

\begin{equation}
\bar{X} = \frac{\sum_{i=1}^n X_i}{n} (\#eq:mean)
\end{equation}

さらに表 \@ref(tab:mtcars) を見よ.

```{r mtcars, echo=FALSE}
knitr::kable(mtcars[1:5, 1:5], caption = "mtcars データ")
```
```

この文書の出力を図 4.2 に示します.

数式, 定理, 節の見出しにも相互参照することができます. これらのタイプの参照の方法は **bookdown** 本の 2.2, 2.6 節でより詳しく説明されています.

4.8 日付を自動的に更新する

出力されたレポートに Rmd 文書がコンパイルされた日付を表示したいなら, YAML メタデータの `date` フィールドにインライン R コードを追加し, 現在の日付を得るために `Sys.Date()` or `Sys.time()` 関数を使用できます.

図, 表, 数式を相互参照する

bookdown による生成

図 1 を見よ.

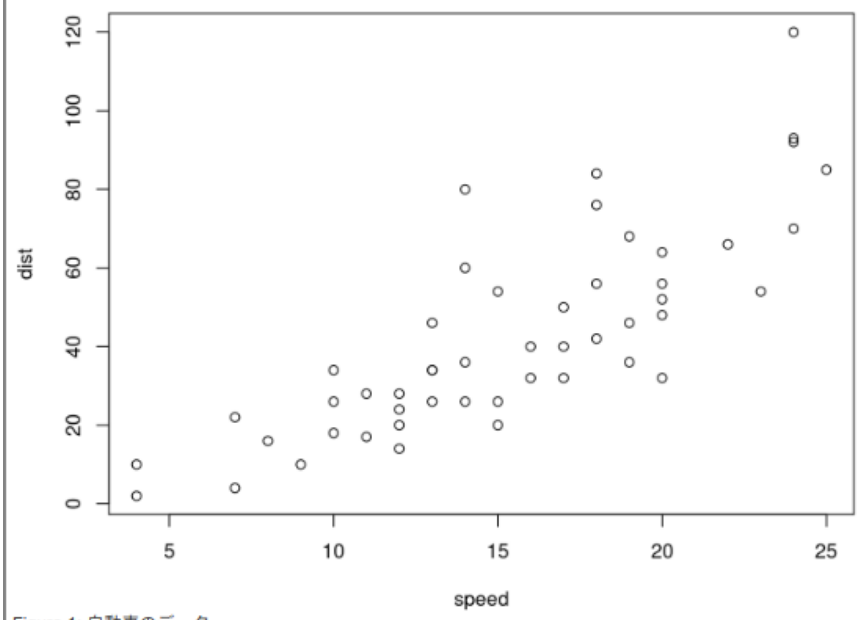


Figure 1: 自動車のデータ

次に数式(1) を見よ.

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n} \tag{1}$$

さらに表 1 を見よ.

Table 1: mtcars データ

	mpg	cyl	disp	hp	drat
Mazda RX4	21.0	6	160	110	3.90
Mazda RX4 Wag	21.0	6	160	110	3.90
Datsun 710	22.8	4	108	93	3.85
Hornet 4 Drive	21.4	6	258	110	3.08
Hornet Sportabout	18.7	8	360	175	3.15

図 4.2: R Markdown 文書内の相互参照の例

表 4.1: R における日付と時刻のフォーマット

コード	意味	コード	意味
%a	曜日の略称	%A	曜日の名称
%b	月の略称	%B	月の名称
%c	ロケール依存の時刻フォーマット	%d	数値表記の日
%H	数値表記の時間 (24 時間)	%I	数値表記の時間 (12 時間)
%j	1 年の何日目か	%m	数値表記の月
%M	数値表記の分	%p	ロケール依存の午前 / 午後フォーマット
%S	数値表記の秒	%U	年の何週目か (日曜日始まり)
%w	数値表記の曜日 (0=日曜日)	%W	年の何週目か (月曜日始まり)
%x	ロケール依存の日付フォーマット	%X	ロケール依存の時刻フォーマット
%y	下 2 桁表記の年	%Y	4 桁表記の年
%z	GMT との時差	%Z	タイムゾーン (文字表記)

```
date: "`r Sys.Date()`"
```

もっと人間にとって読みやすい, 特定の日次フォーマットを指定したいかもしれません. 例えば以下のようにします.

```
date: "`r format(Sys.time(), '%x')`"
```

例えば 2021 年 9 月 15 日 といったコードはあなたが文書を knit するごとに, 日付を動的に生成します. 日付のフォーマットをカスタマイズしたいならば, ご自分でフォーマット文字列を与えて変更できます. いくつか例をお見せしましょう.

- `%Y %B`: 2021 9 月
- `%y/%m/%d`: 21/09/15
- `%b%d (%a)`: 9 月 15 (水)

表 4.1 は POSIXct フォーマットの一覧です.

最後に, 説明文を日付に含めたいときのことを書いておきます. このように R コードの前に「最終コンパイル日」のような何らかの文を追加することができます.

```
date: "最終コンパイル日 `r format(Sys.time(), '%Y/%m/%d')`"
```

4.9　文書に複数の著者を表記する

R Markdown 文書の YAML フロントマターに複数の著者を加える方法は複数あります．単純に，全員を同列に並べたい場合，1 つの文字列を与えることでできます．例えばこのように．

```
---
title: "無題"
author: "John Doe, Jane Doe"
---
```

別の方法として，各エントリごとに行を分けたいならば，YAML フィールドにエントリのリストを与えることができます．これは著者ごとに E メールアドレスや所属情報を加えたいときに役に立ちます．例えばこのように．

```
---
author:
  - John Doe, 組織 1
  - Jane Doe, 組織 2
---
```

追加情報を文書の脚注として追記したい時，Markdown 構文の `^[]` を利用できます．これは著者ごとに連絡先 E メールや住所といった多くの情報を含めたい場合により便利です．厳密な動作は出力フォーマットに依存します．

```
---
author:
  - John Doe^[組織 1, john@example.org]
  - Jane Doe^[組織 2, jane@example.org]
```

```
---
```

特定の R Markdown テンプレートを使うと YAML に追加パラメータを直接指定できます. 例えば Distill[*7] 出力フォーマットは `url`, `affiliation`, `affiliation_url` を指定することが可能です. まずは **distill** パッケージ (JJ Allaire Rich Iannone, et al., 2021) をインストールします.

```
01 install.packages("distill")
```

Distill フォーマットは詳細な著者情報を与えて使うことができます. 例えばこのように.

```
---
title: "R Markdown のための Ðistill"
author:
  - name: "JJ Allaire"
    url: https://github.com/jjallaire
    affiliation: RStudio
    affiliation_url: https://www.rstudio.com
output: distill::distill_article
---
```

4.10 図のキャプションへの付番

以下の例のように, **bookdown** (Xie, 2021a) 出力フォーマット を, 図のキャプションに図番号を追加するのに使うことができます.

```
---
output: bookdown::html_document2
---
```

[*7] https://rstudio.github.io/distill/

```
```{r cars, fig.cap = "すごいプロット"}
plot(cars)
```

```{r mtcars, fig.cap = "これもすごいプロット"}
plot(mpg ~ hp, mtcars)
```
```

4.7 節では表や数式といった他の要素でどのように動くか, そして付番された要素をテキスト内で相互参照する方法を実演しています. `html_document2` の他にも, `pdf_document2`, `word_document2` といった他の出力に対する同様のフォーマット関数もあります.

bookdown 以外の R Markdown 出力フォーマットにもこの機能を追加できます. 鍵となるのはこれらが **bookdown** 出力フォーマットの「基本フォーマット」であることです. 例えば, `rticles::jss_article` フォーマットで図に付番と相互参照をするために以下が使えます.

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

bookdown 出力フォーマット関数のヘルプページを読んで, `base_format` 引数があるかどうか確認してみてください (例: `?bookdown::html_document2`).

4.11 単語をコンマ区切りで結合する

文字列ベクトルを人間の読みやすい形で出力したいとします. 例えば `x <- c("apple", "banana", "cherry")` について, きっとあなたは `[1] "apple" "banana" "cherry"` のような R が通常ベクトルを出力する形式は好まず, 代わりに "`apple, banana, and cherry`" という文字列がほしいのではないでしょうか. R 基本関数には文字列ベクトルを連結して 1 つにまとめる `paste()` があります. 例えば `paste(x, collapse = ', ')` とすれば, 出力は `"apple, banana, cherry"` となるでしょう. この方法の困ったところは (1) 接続詞 "and" が欠けており, (2) ベクトルの要素が 2 つだけの場合はコンマを使うべきでない (`"apple, banana"` ではなく `"apple and banana"` という出力になるべき) ということです.

`knitr::combine_words()` 関数は文字列ベクトルの長さにかかわらず, 要素を連結して文にできます.

基本的に, 単語 1 つに対してはそのまま同じものを返し, “A and B” という 2 つの単語に対しては "A and B" と返し, 3 つ以上なら "A, B, C, ..., Y, and Z" というふうに返します. この関数はさらに出力をカスタマイズするいくつかの引数を持っています. 例えば出力される単語をバッククオートで囲みたいなら, knitr::combine_words(x, before = '`') を使うこともできます. 以下に他の引数についてもさらなる例を示します. これらの出力例から引数の意味がよくわからないのであれば, ヘルプページ ?knitr::combine_words もご覧ください.

```
v <- c("apple", "banana", "cherry")
knitr::combine_words(v)
## apple, banana, and cherry
knitr::combine_words(v, before = "`", after = "'")
## `apple', `banana', and `cherry'
knitr::combine_words(v, sep = "、", and = "そして")
## apple、banana、そして cherry
knitr::combine_words(v, sep = " / ", and = "")
## apple / banana / cherry
knitr::combine_words(v[1])  # 単語 1 つ
## apple
knitr::combine_words(v[1:2])  # 単語 2 つ
## apple and banana
knitr::combine_words(LETTERS[1:5])
## A, B, C, D, and E
```

この関数はインライン R コードを使うときに特に使いやすいでしょう. 例えばこのように.

```
今朝は `r knitr::combine_words(v, sep = '、', and='')`を食べた.
```

4.12 複数の改行コードを維持する

Markdown ユーザは, verbatim 環境 (コードブロック) 以外の場所では空白 (改行コード含む) は大抵の場合意味を持たないことに気づき, 驚くでしょう. 2 つ以上のスペースはスペース 1 つと同じであり, 改行 1 つはスペース 1 つと同じです. LaTeX や HTML を使ったことがあるなら, これらの言語と同じルールであるため驚くことはないかもしれません.

Markdown では, 空白行はしばしば段落などの要素の分離に使われます. 新しい段落に入らずに改行をするには, 末尾にスペース 2 つを追加しなければなりません. 特に詩や歌詞を引用したいときなど, 複数回改行したいときもあるかもしれません. 各行の末尾にスペース 2 つを手動で書き加えるのはうんざりする作業です. `blogdown:::quote_poem()` はこの作業を自動でやってくれます. 例えばこのように.

```
blogdown:::quote_poem(c("かたつむり", "そろそろ登れ",
  "富士の山"))
## [1] "> かたつむり  \n そろそろ登れ  \n 富士の山"
```

RStudio IDE と **blogdown** パッケージ (Xie Dervieux, and Presmanes Hill, 2021) をインストールして使っているなら, 改行を維持したいテキストを選択し, ツールバーの “Addins” から RStudio アドインの “Quote Poem” をクリックすることができます. 例えば以下のテキスト (fenced code block 記法内) は末尾にスペースが付いていません.

```
田子の浦ゆ
うち出でてみれば
真白にそ
富士の高嶺に
雪は降りける

--- 山部赤人
```

上記の詩句を選択肢, RStudio アドインの “Quote Poem” をクリックすれば, こう出力されます.

> 田子の浦ゆ
> うち出でてみれば
> 真白にそ
> 富士の高嶺に
> 雪は降りける
>
> — 山部赤人

たまに「fenced code block は空白を維持するのに, 詩句をコードブロックに書くのはなぜですか」

と質問があります．コードは詩的でありますが，詩はコードではありません．コーディング中毒にならないようにしましょう．

:::{.infobox .caution data-latex”{caution}“}

訳注

上記の例では，最終行の出典の右寄せが再現できません．右寄せには Pandoc の fenced `Div` blocks の機能が使用されています (9.6 節)．詳細はこの文書のソース (Rmd と CSS ファイル) を確認してください．HTML 版をご覧ならば上部ツールバーの “Edit” ボタンからソースの URL を辿ることができます．:::

4.13 モデルを数式に変換する

Daniel Anderson らによって開発された **equatiomatic** パッケージ (Anderson Heiss, and Sumners, 2021) (https://github.com/datalorax/equatiomatic) は R で当てはめたモデルに対応する数式を表示するための便利な自動化された手段です．簡単な例を以下に示します．

```
fit <- lm(mpg ~ cyl + disp, mtcars)
# 理論モデルを表示
equatiomatic::extract_eq(fit)
```

$$\text{mpg} = \alpha + \beta_1(\text{cyl}) + \beta_2(\text{disp}) + \epsilon$$

```
# 実際の係数を表示
equatiomatic::extract_eq(fit, use_coefs = TRUE)
```

$$\widehat{\text{mpg}} = 34.66 - 1.59(\text{cyl}) - 0.02(\text{disp})$$

実際の数式を表示するには，チャンクオプション `results = "asis"` (オプションの意味は 11.11 節参照) が必要です．そうしないと，テキスト出力がそのまま表示されてしまいます．

このパッケージについてより詳しく知りたいならば，ドキュメントを読み，Github 上での開発状況を追ってください．

4.14　複数の R プロットからアニメーションを作成する

1 つのコードチャンクで連続したプロットを生成したとき, これらを結合して 1 つのアニメーションを生成できます. 出力フォーマットが HTML なら, これは簡単です. **gifski** パッケージ (Ooms, 2021a) をインストールし, チャンクオプション `animation.hook = "gifski"` 設定するだけです. 図 4.3 はシンプルな「パックマン」のアニメーションで, これは以下のコードで作成しました.

````
```{r, animation.hook="gifski"}
for (i in 1:2) {
  pie(c(i %% 2, 6), col = c('red', 'yellow'), labels = NA)
}
```
````

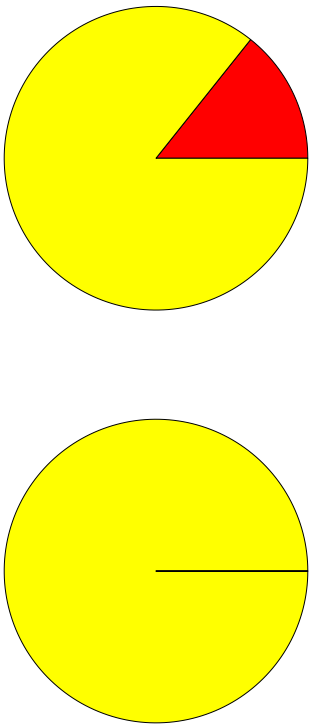


図 4.3: パックマンのアニメーション

アニメーションのフォーマットは GIF で, HTML 出力ではうまく動作しますが, LaTeX は GIF

を直接サポートしていません．あなたが本書の PDF または印刷版を読んでいるなら，図 4.3 が 2 つの動かない画像になっているのはこれが理由です．本書のオンライン版を読めば，実際のアニメーションが見られるでしょう．

PDF でもアニメーションを動作させることはできますが，事前準備が 2 つ必要です．第 1 に，LaTeX パッケージの **animate**[*8] を読み込む必要があります (方法は 6.4 節参照)．第 2 に，Acrobat Reader でのみアニメーションを見ることができます．第 2 位に，Acrobat Reader でのみアニメーションの動作を見ることができます．その上で以下の例のように，チャンクオプション `fig.show = "animate"` で **animate** パッケージ を使いアニメーションを作成できるようにします．

````
---
title: PDF でのアニメーション
output:
  pdf_document:
    extra_dependencies: animate
---

以下のアニメーションは Acrobat Reader でのみ見ることができます．

```{r, fig.show='animate'}
for (i in 1:2) {
  pie(c(i %% 2, 6), col = c('red', 'yellow'), labels = NA)
}
```
````

アニメーションのイメージフレーム間の表示間隔はチャンクオプション `interval` で設定できます．デフォルトでは `interval = 1` (つまり 1 秒) です．

R パッケージ **animation** (Xie, 2018) には，統計的計算の方法やアイディアを表現するアニメーションの例がいくつか入っています．**gganimate** パッケージ (Pedersen and Robinson, 2020) は **ggplot2** (Wickham Chang, et al., 2021) に基づいた滑らかなアニメーションの作成を可能にします．どちらも R Markdown で動作します．

[*8] https://ctan.org/pkg/animate

4.15 ダイアグラムを作成する

ダイアグラムやフローチャートを生成する, R とは独立したプログラム (例: Graphviz) は多くありますが, これらは Rmd 文書内のコードチャンク内で直接取り扱うほうが簡単です.

R ではいくつかのパッケージが使用可能ですが, その中で **DiagrammeR** (Richard Iannone, 2020) とその他いくつかを最後に簡単に解説します. 完全なデモは https://rich-iannone.github.io/DiagrammeR/ で見ることができます. この節では基本的な使用法とダイアグラム内で R コードを使う方法を紹介します.

4.15.1 基本的なダイアグラム

DiagrammeR はいくつかの異なるグラフ言語を使ってグラフを作成する方法を提供します. この節では Graphviz の例を提示しますが,[*9] **DiagrammeR** は純粋に R コードだけでグラフを作ることもできます.

RStudio IDE は Graphviz (`.gv`) および mermaid (`.mmd`) ファイルをネイティブにサポートしています. これらのタイプのファイルを RStudio で編集すると, シンタックスハイライトされるという利点があります. RStudio のツールバーの “Preview” ボタンをクリックすると, ダイアグラムをプレビューすることができます. 図 4.4 は, 4 つのステップを表す 4 つの矩形で構成された, フローチャートの単純な例です. これは以下のコードで生成されています.

```
DiagrammeR::grViz("digraph {
  graph [layout = dot, rankdir = TB]

  node [shape = rectangle]
  rec1 [label = 'ステップ 1. 起床する']
  rec2 [label = 'ステップ 2. コードを書く']
  rec3 [label = 'ステップ 3. ???']
  rec4 [label = 'ステップ 4. 収入を得る']

  # ノード ID でエッジを定義
  rec1 -> rec2 -> rec3 -> rec4
```

[*9] あなたのバックグラウンド次第では, この節は **DiagrammeR** に対する偏った解説になるかもしれません. このパッケージに興味があるなら, パッケージの公式ドキュメントをご覧ください.

```
  }",
  height = 500)
```

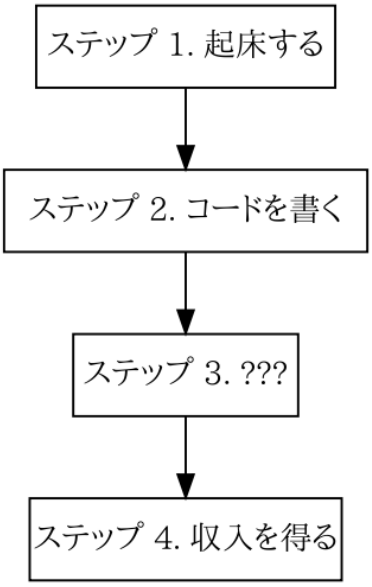


図 4.4: プログラマの絵空事を表したダイアグラム

ノードの形状, 色, 線のタイプを定義したり, パラメータを追加したりできる拡張的な操作も用意されています.

4.15.2 図にパラメータを追加する

Graphviz の置換機能は可読性を損なうことなく, R コードを Graphviz のグラフ設定に混ぜ込むことができます. @@ を伴う置換を指定するには, そこに置換されるのは有効な R 評価式であることを確実にせねばなりません. 評価式は脚注として置かれ, そして R ベクトルオブジェクトを返すものでなくてはなりません. @@ という記法のすぐ後には数字が続き, これは R 評価式の脚注番号に対応します. 図 4.5 はダイアグラムへの R コードの埋め込みと評価の例です.

```
DiagrammeR::grViz("
  digraph graph2 {
```

```
03
04    graph [layout = dot, rankdir = LR]
05
06    # node definitions with substituted label text
07    node [shape = oval]
08    a [label = '@@1']
09    b [label = '@@2']
10    c [label = '@@3']
11    d [label = '@@4']
12
13    a -> b -> c -> d
14    }
15
16    [1]: names(iris)[1]
17    [2]: names(iris)[2]
18    [3]: names(iris)[3]
19    [4]: names(iris)[4]
20    ",
21    height = 100)
```

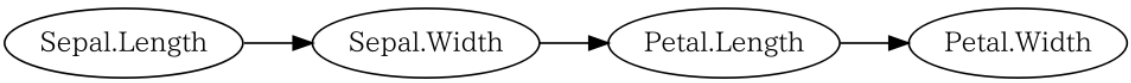


図 4.5: R から入力されたパラメータを使用したダイアグラム

4.15.3 その他のダイアグラム作成パッケージ

ダイアグラム作成に使えるパッケージとして, **nomnoml** (de Vries and Luraschi, 2020), **diagram** (Soetaert, 2020), **dagitty** (Textor van der Zander, and Ankan, 2021), **ggdag** (Barrett,

2021), **plantuml** (https://github.com/rkrug/plantuml) といったものも見ておくとよいでしょう.

4.16 特殊文字をエスケープする

Markdown 構文で特殊な意味を持つ文字がいくつかあります. これらの文字を直接使いたい場合, エスケープしなければなりません. 例えばテキストを囲むアンダースコアの組はたいていの場合テキストをイタリック体にします. イタリック体ではなく, アンダースコアをそのまま表示させたいなら, アンダースコアをエスケープする必要があります. 特殊な文字をエスケープする方法は, その直前にバックスラッシュを付けることです. 例えば「**ここは _イタリックに _したくない**.」というふうに. 同様に, # をセクションヘッダを表してほしくないなら, \# **これは見出しではない** などと書くこともできます.

4.12 節で言及したように, 連続した空白文字は 1 つの正規スペースとして表示されます. 書いたとおりに連続した空白文字を表示させたいならば, 1 つ 1 つにエスケープが必要です. 例えば **ソーシャル \ \ \ ディスタンス維持** というふうに. 空白がエスケープされた時, 空白は「改行しない空白」に変換されます. これは, そのスペースの位置では行が折り返されないということです. 例えば `Mr.\ Dervieux` と言うふうに.

4.17 テキストのコメントアウト

ソース文書内のテキストを最終的な出力文書に表示させないようコメントアウトするのはとても便利です. この用途のため, HTML の構文である `<!-- **ここにコメント** -->` を使えます. コメントはどの出力フォーマットにも表示されません.

コメントは 1 行でも, 複数行にも広げられます. これは草稿を書くのに便利でしょう.

RStudio を使っているなら, 1 行丸ごとコメントアウトするのにキーボードショートカット `Ctrl + Shift + C` (MacOS なら `Command + Shift + C`) を使えます.

4.18 目次から見出しを省略する

目次に特定のセクションの見出しを表示させたくないなら, 見出しに 2 つのクラスを追加できます. `unlisted` と `unnumbered` です. 例えばこのように

```
# 見出し {.unlisted .unnumbered}
```

この機能は Pandoc 2.10 以降のバージョンが必要です．`rmarkdown::pandoc_version()` で Pandoc のバージョンを確認しましょう．バージョンが 2.10 より古いなら，新しいバージョンをインストールすることになるでしょう (1.1 節参照)．

4.19 全てのコードを補遺に置く (*)

対象読者がレポートを読む時，計算の詳細に強く関心があるのでない限り，あなたはレポートにソースコードブロックを表示させたくないかもしれません．この用途で，チャンクオプション `echo = FALSE` を設定してソースコードを隠し，読者がプログラムコードで気が散らないようにすることができます．しかしそれでも，ソースコードは再現可能性のある研究のために重要です．読者はレポートを読み終わった後に計算の正しさを検証したいと思うかも知れません．この場合，本文中の全てのコードブロックをまとめ，文書の末尾 (例えば補遺として) に表示するというのは良い考えでしょう．

チャンクオプションの `ref.label` と `knitr::all_labels()` 関数を使い，文書内の全てのコードチャンクを取り出して 1 つのコードチャンクにまとめる簡単な方法があります．例えばこのように．

```
# 補遺：本稿で使ったコード全文

```{r ref.label=knitr::all_labels(), echo=TRUE, eval=FALSE}
```
```

チャンクオプション `ref.label` について詳しく知らないならば，14.1.3 節を読んでください．

`knitr::all_labels()` 関数は文書内の全てのチャンクラベルを返すため，`ref.label = knitr::all_labels()` は全てのソースコードチャンクを回収しこのチャンクに持ってくることを意味します．チャンクオプション `echo = TRUE` (コードを表示させる) と `eval = FALSE` (全てのコードはすでに実行されているため，このコードチャンクは実行してはいけません) を付与すれば，1 つのコードチャンクに全てのソースコードのコピーを表示させられます．

`ref.label` は任意のチャンクラベルの文字列ベクトルであるため，補遺に表示するコードチャンクを一部だけにするようにラベルをフィルタリングできます．以下はその例 (Ariel Muldoon[*10] によるものです) として `setup` と `get-labels` というラベルを排除しています．

[*10] https://yihui.org/en/2018/09/code-appendix/

````
```{r get-labels, echo = FALSE}
labs = knitr::all_labels()
labs = setdiff(labs, c("setup", "get-labels"))
```

```{r all-code, ref.label=labs, eval=FALSE}
```
````

`knitr::all_labels()` の引数を使ってコードチャンクをフィルタリングできます. 例えば `Rcpp` エンジン (`engine == "Rcpp"`) を使用した全てのコードチャンクを得て, かつ文書に表示しない (`echo = FALSE`) ようにするには `knitr::all_labels(engine == "Rcpp", echo == FALSE)` を使えます. どのコードチャンクを補遺に表示したいのか, 正確にコントロールしたいならば, 指定したいコードチャンクに特殊なチャンクオプション `appendix = TRUE` を使い, それらのチャンクのラベルを得るのに `ref.label = knitr::all_labels(appendix == TRUE)` を使えます.

4.20 Pandoc の Lua フィルタから操作する (*)

技術的にはこの節は少し発展的ですが, Markdown の内容が Pandoc 抽象構文木 (AST) にどのように翻訳されるかを一度学べば, Lua というプログラミング言語を使ってどのような Markdown の要素も操作する力を得ることになります.

基本として, Pandoc は Markdown ファイルを読み取り, その内容が AST にパースされます. Pandoc はこの AST を Lua スクリプトを使って修正することを可能にします. AST の意味するものを示すため, 以下のような簡単な Markdown ファイル (`ast.md`) を使います.

```
01 ## 第 1 節
02
03 Hello world!
```

このファイルは見出し 1 つとパラグラフ 1 つを持っています. Pandoc がこの内容をパースした後にファイルを JSON 形式に変換すれば, R ユーザーにとっては 結果として現れる AST を理解するよりも簡単でしょう.

```
pandoc -f markdown -t json -o ast.json ast.md
```

そして JSON ファイルを R に読み込み, データ構造を書き出します.

この操作をしたら, Markdown の内容は再帰的なリストで表現されていることが分かるでしょう. その構造を以下に表します. ラベル `t` は “type”, `c` は “content” を表します. 例として見出しを取り上げてみましょう. タイプは “Header” で, その中身は 3 つの要素が含まれています. 見出しのレベル (2), 属性 (例えば ID が `section-one` であること), そしてテキストの内容です.

```
xfun:::tree(
  jsonlite::fromJSON('ast.json', simplifyVector = FALSE)
)
```

```
List of 3
 ¦-pandoc-api-version:List of 2
 ¦  ¦-: int 1
 ¦  ¦-: int 22
 ¦-meta              : Named list()
 ¦-blocks            :List of 2
    ¦-:List of 2
    ¦  ¦-t: chr "Header"
    ¦  ¦-c:List of 3
    ¦     ¦-: int 2
    ¦     ¦-:List of 3
    ¦     ¦  ¦-: chr "第 1 節"
    ¦     ¦  ¦-: list()
    ¦     ¦  ¦-: list()
    ¦     ¦-:List of 1
    ¦        ¦-:List of 2
    ¦           ¦-t: chr "Str"
    ¦           ¦-c: chr "第 1 節"
    ¦-:List of 2
       ¦-t: chr "Para"
       ¦-c:List of 3
          ¦-:List of 2
```

```
¦  ¦-t: chr "Str"
¦  ¦-c: chr "Hello"
¦-:List of 1
¦  ¦-t: chr "Space"
¦-:List of 2
   ¦-t: chr "Str"
   ¦-c: chr "world!"
```

あなたが AST に気づけば, Lua によって修正することができます. Pandoc は組み込みの Lua インタプリタを持っているので, 追加でツールをインストールする必要はありません. Lua スクリプトは Pandoc では「Lua フィルタ」と呼ばれます. 次に見出しのレベルを 1 上げる, 例えばレベル 3 の見出しを 2 に変換する簡単な例を見せます. これは文書のトップレベルの見出しがレベル 2 で, 代わりにレベル 1 から始めたい場合に便利です.

最初に `raise-header.lua` という名前の Lua スクリプトファイルを作ります. これには `Header` という名前の関数が含まれており, “Header” タイプの要素を修正したいということを意味しています (一般に, あるタイプの要素を処理するためにタイプ名を関数名として使うことができます).

```lua
function Header(el)
  -- 見出しのレベルは要素の持つ 'level' 属性でアクセスできます.
  -- 後述の Pandoc ドキュメントを見てください.
  if (el.level <= 1) then
    error("h1 のレベルを上げる方法がわかりません")
  end
  el.level = el.level - 1
  return el
end
```

そしてこのスクリプト Pandoc の `--lua-filter` 引数に与えることができます. 例えばこうです.

```
pandoc -t markdown --atx-headers \
  --lua-filter=raise-header.lua ast.md
```

```
[WARNING] Deprecated: --atx-headers. Use --markdown-headings=atx instead.
# 第 1 節
```

```
Hello world!
```

`## Section One` を `# Section One` へ変換することに成功したのがお分かりかと思います. この例は些細なものだと思うかも知れませんし, どうして次のように単に正規表現を使って `##` を `#` に置き換えないのかと思うことでしょう.

```
01 gsub("^##", "#", readLines("ast.md"))
```

たいていの場合, 構造化された文書を操作するのに正規表現はロバストな手段ではありません. 例えば `##` が R コード内でコメントに使われているというように, ほぼいつも例外があるためです. AST は構造化されたデータを与えてくれるので, 確実に意図した要素を修正していることが分かります.

Pandoc の Lua フィルタに関する追加ドキュメントが https://pandoc.org/lua-filters.html にあり, ここで多くの例を見つけることができます. GitHub リポジトリ https://github.com/pandoc/lua-filters のコミュニティで書かれたフィルタを見つけることもできます.

R Markdown の世界では Lua フィルタを活用しているパッケージの例の一部が以下になります (たいていは `inst/` ディレクトリにあります).

- **rmarkdown** パッケージ (https://github.com/rstudio/rmarkdown) は改行 (4.1 節参照) を挿入するフィルタとカスタムブロック (9.6 節参照) を生成するフィルタを含んでいます.
- **pagedown** パッケージ (Xie Lesur, et al., 2021) には脚注を実装するのを助けるフィルタと HTML ページに図のリストを表示するフィルタがあります.
- **govdown** パッケージ (Garmonsway, 2021) には Pandoc の `Div` による囲みを適切な HTML タグに変換するフィルタがあります.

本書の 5.1.2 節でも Lua フィルタでテキストの色を変更する方法を紹介する例を見ることができます.

Lua フィルタを (上記のパッケージのように) 導入するために R パッケージ を作りたくない R Markdown ユーザーは, これらの Lua スクリプトをコンピュータのどこかに保存し, R Markdown 出力フォーマットの `pandoc_args` オプションを次の例のように適用することもできます.

```
---
output:
```

```
  html_document:
    pandoc_args:
      - --lua-filter=raise-header.lua
---
```

第5章

書式

Markdown 言語の最大の強みは, その簡潔さが初心者にとっても読み書きを非常に簡単にさせていることです. これはオリジナルの Markdown 言語の考案者も次のようにまとめている設計原理の鍵です.

> Markdown 形式の文書は見たままに, タグや整形の指示文でマークアップされず, プレーンテキストとして出力されるべきである.
>
> — John Gruber[*1]

しかし, これはカスタマイズのコストとして跳ね返ります. 典型的なワードプロセッサの多くの機能は Markdown でそのまま使うことができません. 例えば以下のような機能です.

- テキストの一部のフォントサイズを変更する
- ある単語のフォント色を変更する
- テキストアラインメントを指定する

こういった機能があなたの努力に見合うかどうかはあなたの判断に委ねます. Markdown は「自然界」はプレーンテキストからなり, (見た目上の) 面白さを欲求して**作為**すべきではない, という禁欲主義者たちの哲学をいくらか反映しています. いずれにせよ, この章では R Markdown 文書の見た目や要素のスタイルをカスタマイズをどうやればできるかの豆知識をいくつか提示します.

Markdown 言語の基礎の復習が必要ならば, https://www.rstudio.com/resources/cheatsheets/ にある R Markdown チートシート[*2]には基本構文の概観がうまく盛り込まれています.

[*2] **訳注**: 同ページで日本語版も公開されています.

5.1 フォント色

Markdown 構文にはテキストの色を変更する方法は組み込まれていません. HTML と LaTeX の構文で単語の書式を変更できます.

- HTML では, テキストを `<span>` タグで囲み CSS で色を設定します. 例えば `<span style="color: red;">text</span>` というふうに.
- PDF では, LaTeX コマンドの `\textcolor{}{}` が使えます. これには LaTeX パッケージの **xcolor** が必要で, Pandoc のデフォルトの LaTex テンプレートに含まれています.

PDF でテキストの色を変更する例として, 以下のようなものを挙げます.

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

薔薇は \textcolor{red}{赤い}, 菫は \textcolor{blue}{青い}.
```

上記の例では, カーリー・ブレイス（{}）のペアの 1 番目には指定するテキスト色が含まれ, 2 番めには色を適用したいテキストが含まれています.

複数の出力フォーマットに対応する R Markdown の文書をデザインしたいときは, 生の HTML または LaTeX コードを文書の中に埋め込むべきではありません. それは, 出力フォーマットがかわると無視される (例: LaTeX コードは HTML では無視され, HTML タグは LaTeX 出力時には失われます.) ためです. 次に, この問題に対処する方法を 2 つ提示します.

5.1.1 生の HTML/LaTeX コードを書く関数を使う

以下のようなカスタム R 関数を書くことで **knitr** パッケージの `is_latex_output()` および `is_html_output()` 関数を使って, 出力フォーマットに依存した適切な構文を挿入することができます.

```
01 colorize <- function(x, color) {
02   if (knitr::is_latex_output()) {
```

```
    sprintf("\\textcolor{%s}{%s}", color, x)
  } else if (knitr::is_html_output()) {
    sprintf("<span style='color: %s;'>%s</span>", color,
      x)
  } else x
}
```

そうするとインライン R コード内で `` `r colorize("文の一部を赤色にする", "red")` `` ように使うことができます．これは 文の一部を赤色にする でしょう (モノクロで印刷されたものを読んでいるなら, 赤色には見えないはずです).

5.1.2 Lua フィルタを使う (*)

Lua という他のプログラミング言語が関わるこの方法は, R ユーザにとっては少し発展的ですが, きわめて強力です．Pandoc の Lua フィルタ (4.20 節参照) を使って Markdown 要素をプログラムで修正することができます．以下は使用例の全容です．

````
---
title: "Color text with a Lua filter"
output:
  html_document:
    pandoc_args: ["--lua-filter=color-text.lua"]
  pdf_document:
    pandoc_args: ["--lua-filter=color-text.lua"]
    keep_tex: true
---

First, we define a Lua filter and write it to
the file `color-text.lua`.

```{cat, engine.opts = list(file = "color-text.lua")}
Span = function(el)
  color = el.attributes['color']
  -- if no color attribute, return unchange
````
````

```
  if color == nil then return el end

  -- transform to <span style="color: red;"></span>
  if FORMAT:match 'html' then
    -- remove color attributes
    el.attributes['color'] = nil
    -- use style attribute instead
    el.attributes['style'] = 'color: ' .. color .. ';'
    -- return full span element
    return el
  elseif FORMAT:match 'latex' then
    -- remove color attributes
    el.attributes['color'] = nil
    -- encapsulate in latex code
    table.insert(
      el.content, 1,
      pandoc.RawInline('latex', '\\textcolor{'..color..'}{')
    )
    table.insert(
      el.content,
      pandoc.RawInline('latex', '}')
    )
    -- returns only span content
    return el.content
  else
    -- for other format return unchanged
    return el
  end
end
```

Now we can test the filter with some text in brackets with the `color` attribute, e.g.,

```
> Roses are [red and **bold**]{color="red"} and
```

```
> violets are [blue]{color="blue"}.
```

この例では, bracketed_spans という名称の Pandoc Markdown 拡張機能をこっそり使っています. これはテキストに属性を付けて書くことを可能にします. 例えば [text]{.class attribute="value"} のように. cat コードチャンク[*3]内で定義された Lua フィルタは, 出力が HTML ならば という形でテキストを配置し, LaTeX なら \textcolor{...}{} として配置します. color-text.lua というファイル名で書き出しコマンドラインオプション --lua-filter で有効になった Lua フィルタは出力フォーマットの pandoc_args オプションを経由して Pandoc に与えられます.

従来の方法と比較して, Lua フィルタを使う利点はパーレン (()) の中でも Markdown 構文が使えることですが, 以前の節で紹介した R の colorize() 関数は Markdown 構文を使うことができません (例えば colorize('**太字**') と書いても太字にはなりません).

5.2 テキストをインデントする

4.12 節で話したように, Markdown では空白文字はしばしば意味をなさなくなります. さらに Markdown は, デフォルトでインデントの空白を無視します. しかしインデントを維持したいことがあります. 例えば詩や演説文などです. このような場合は垂直線 (¦) で始まる罫線ブロックを使うことができます. 改行と行頭のスペースは出力でも維持されます. 例えばこのように[*4]

```
¦ When dollars appear it's a sign
¦   that your code does not quite align
¦ Ensure that your math
¦   in xaringan hath
¦   been placed on a single long line
```

出力はこうなります.

[*3] cat コードチャンクを詳しく知らないのなら, 15.6 節を見てください. ここでは, チャンクを .lua ファイルに書き出す便利な方法としてこのエンジンを使っています. そのため Lua スクリプトを color-text.lua という別のファイルとして管理しなくてもよいわけです. cat エンジンを使いたくなというなら, コードチャンクに Lua コードを埋め込む代わりに Lua コードを正しくコピーして別のファイルに保存することができます.

[*4] Claus Ekstrøm: https://yihui.org/en/2018/06/xaringan-math-limerick/ 作のリメリックです.

> When dollars appear it's a sign
> that your code does not quite align
> Ensure that your math
> in xaringan hath
> been placed on a single long line

各行は Markdown のソースでは改行コードが使われています (ハードラップ). 次に続く行をスペースで始めれば, 1 つ前の行の改行と行頭のスペースは通常は無視されます. 例えばこのように入力します.

```
| 採用責任者
| ニンジャの学校,
  ハッカーの大学
| 404 Not Found Road,
  Undefined 0x1234, NA
```

出力はこうなります.

> 採用責任者
> ニンジャの学校, ハッカーの大学
> 404 Not Found Road, Undefined 0x1234, NA

「ニンジャの学校」の直後の改行が無視されているのがわかると思います.

5.3 テキスト出力の幅を制御する

R コードから表示されたテキスト出力の幅が広すぎることがたまにあります. 出力文書のページ幅が固定 (例えば PDF 文書) ならばテキスト出力がページ余白をはみ出すことがあります. その例が図 5.1 です.

R グローバルオプションの `width` は R 関数からのテキスト出力の印字幅を制御するのに使うことができます. デフォルトが大きすぎるなら, 値を小さくしてみてください. このオプションは典型的には, おおまかに 1 行ごとの文字数を表しています (東アジア言語は例外です). 例えばこのように.

このチャンクの出力は幅広すぎる

```
```{r}
options(width = 300)
matrix(runif(100), ncol = 20)
```
```

このチャンクの出力のほうが良い

```
```{r}
options(width = 60)
matrix(runif(100), ncol = 20)
```
```

全ての R 関数が `width` オプションを尊重しているわけではありません．このオプションが動作しないなら, 唯一の選択は長いテキスト行を折り返しすることです．実際これは `html_document` 出力フォーマットのデフォルトの挙動です．あなたの使っている HTML 出力フォーマットが長い行の折返しをしないのなら, 以下の CSS コード を適用してみてください (解説は 7.1 節を参照).

```
pre code {
  white-space: pre-wrap;
}
```

PDF 出力では, 行の折返しはよりトリッキーになります． 解決策の 1 つは, Pandoc 引数の `--listing` を使うことで有効になる LaTeX パッケージの **listings** を使うことです．そうしたなら, このパッケージに対するオプションを設定しなければならず, またその設定コードは外部 LaTeX ファイルに含めることができます (方法は 6.1 節参照) 例えばこのように.

```
---
output:
  pdf_document:
    pandoc_args: --listings
    includes:
```

```
      in_header: preamble.tex
---
```

preamble.tex 内では, **listings** パッケージのオプションを設定しています.

```
\lstset{
  breaklines=true
}
```

listings によるコードブロックの見た目が気に入らないなら, \lstset{} で他の **listings** オプションを設定することができます. 例えば basicstyle=\ttfamily でフォントファミリを変更できます. このパッケージのより詳細な情報はドキュメント https://ctan.org/pkg/listings で見つけることができます.

図 5.1 は長い行のあるデフォルトの pdf_document 出力で, ページ余白をはみ出しています. 図 5.2 は **listings** パッケージでテキストを折り返したときの PDF 出力です.

訳注

listings には多くのオプションがありますが, それだけでデフォルトのシンタックスハイライトを再現するのは難しいです. コードブロックの折返しは **knitr** の styler オプションである程度制御できます. Pandoc は出力ブロックをほとんど表示オプションのない verbatim 環境として出力し, これが問題の主な原因です. フィルタや LaTeX マクロを使うなどしてこの環境を置き換えればデフォルトのシンタックスハイライトと折返しを両立することができます.

5.4 グラフ・画像のサイズを制御する

R が作成するグラフのサイズはチャンクオプション fig.width と fig.height でインチ単位で制御できます. 同様に fig.dim オプション に長さ 2 のベクトルで幅と高さを指定できます. 例えば fig.dim = c(8, 6) は fig.width = 8 と fig.height = 6 を指定したのと同じです. これらのオプションはグラフの物理的なサイズを設定し, さらに out.width と out.height を使い出力時に異なるサイズで, 例えば out.width = "50%" のように表示することが出来ます.

R コードチャンクで生成されないグラフや画像は, 2 通りの方法で掲載できます.

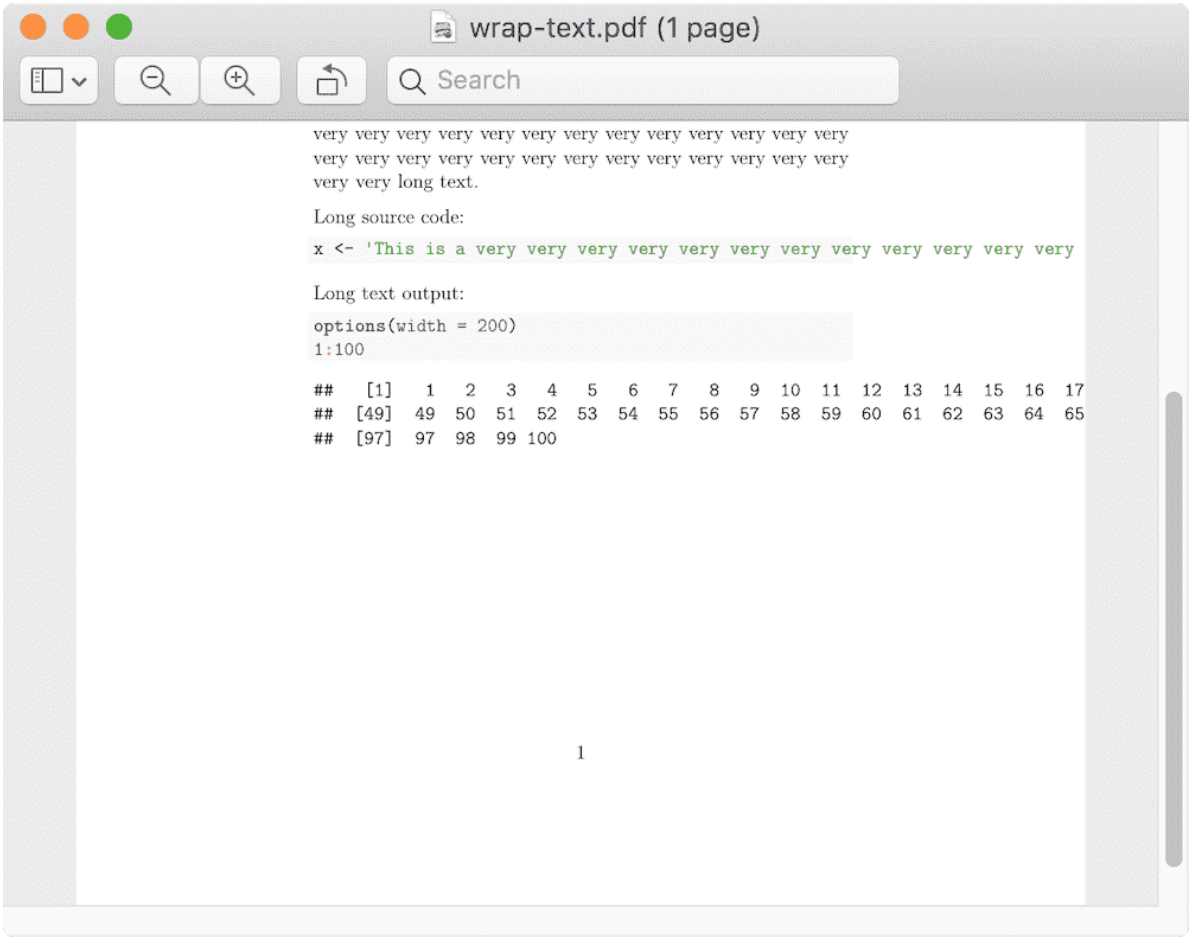


図 5.1: 幅が広すぎる通常のテキスト出力

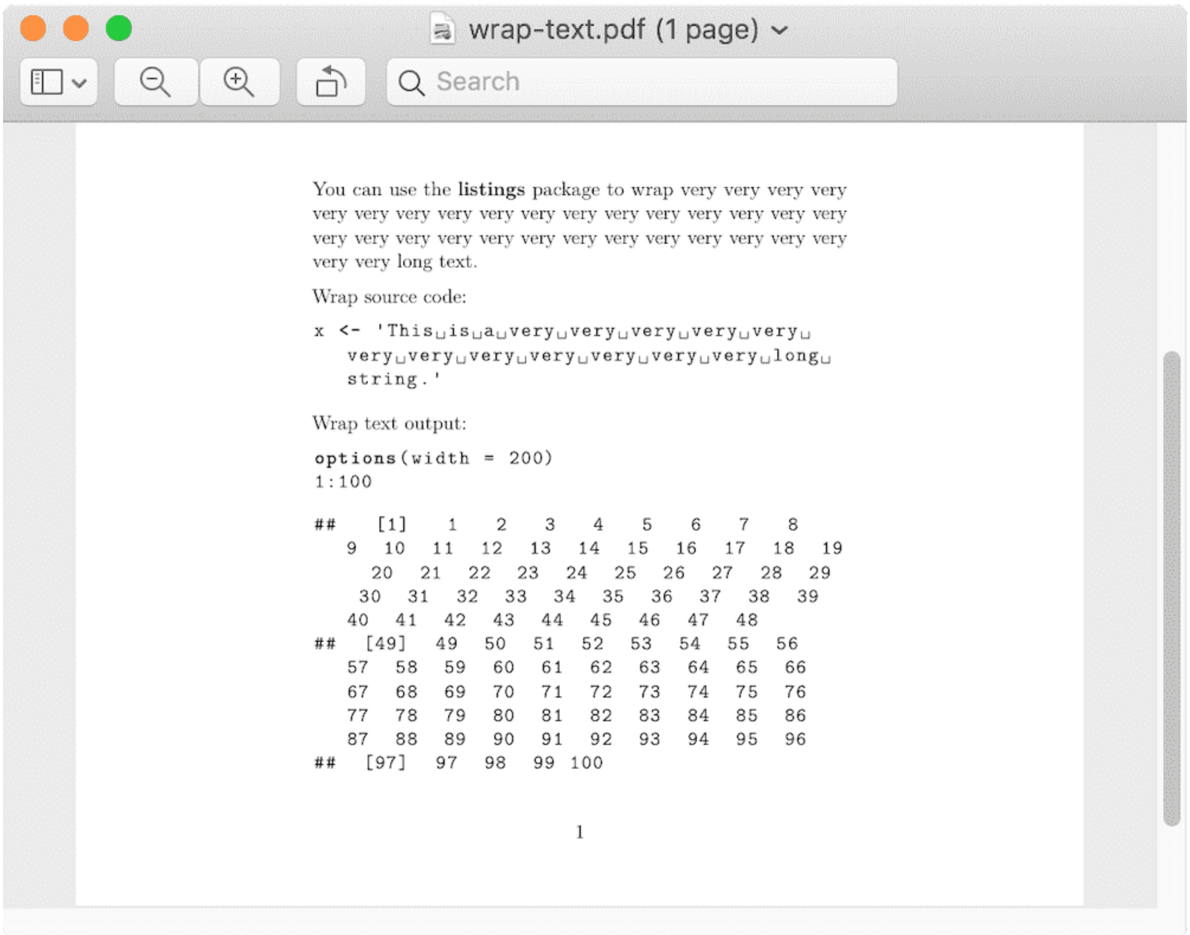
- Markdown 構文 `![キャプション](画像ファイルパス)` を使う．この場合は `width`, `height` 属性でサイズを設定できます 例えばこのように．

```
次のパラグラフに画像を掲載する．

![すてきな画像](なんとか.png){width=50%}
```

- コードチャンクで **knitr** 関数 `knitr::include_graphics()` を使う． そのチャンクで `out.width` と `out.height` というオプションを設定することもできます．例えばこのように．

```
R function を使って外部画像ファイルを掲載します
```



wrap-text.pdf (1 page)

You can use the **listings** package to wrap very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very long text.

Wrap source code:

```
x <- 'This␣is␣a␣very␣very␣very␣very␣very␣
   very␣very␣very␣very␣very␣very␣very␣long␣
   string.'
```

Wrap text output:

```
options(width = 200)
1:100
```

```
##    [1]    1    2    3    4    5    6    7    8
   9   10   11   12   13   14   15   16   17   18   19
     20   21   22   23   24   25   26   27   28   29
    30   31   32   33   34   35   36   37   38   39
   40   41   42   43   44   45   46   47   48
##   [49]   49   50   51   52   53   54   55   56
   57   58   59   60   61   62   63   64   65   66
   67   68   69   70   71   72   73   74   75   76
   77   78   79   80   81   82   83   84   85   86
   87   88   89   90   91   92   93   94   95   96
##   [97]   97   98   99  100
```

1

図 5.2: listings パッケージで折り返されたテキスト出力

````
```{r, echo=FALSE, out.width="50%", fig.cap="良い画像"}
knitr::include_graphics("なんとか.png")
```
````

上記の例では幅 `50%` が使われており, 画像コンテナの半分の幅にすることを意味します (もし画像がページの子要素ではなく, ページに直接含まれていると仮定すると, これはページ幅の半分を意味します). 特定の出力フォーマットに対してのみ画像を生成することが分かっているのなら, 単位を特定することもできます. たとえば出力フォーマットが HTML なら `300px` と書けるでしょう.

5.5 図のアラインメント

チャンクオプション `fig.align` は図のアラインメントを指定します. 例えば `fig.align = 'center'` で中央揃え, あるいは `fig.align = 'right'` で右揃えができます. このオプションは HTML と LaTeX 出力の両方で機能しますが, 他の出力フォーマット (残念ですが Word といったものは) では機能しないかもしれません. R コードチャンクで描画されたグラフも, `knitr::include_graphics()` で取り込まれた外部イメージに対しても機能します.

5.6 コードチャンクをそのまま (verbatim) 表示

典型的には, 私達がコードチャンクとインラインコードを書くときには **knitr** によってパースされ評価してほしいと思って書きます. しかし **knitr** を使ったチュートリアルを書きたいなら, **knitr** にパース**されない**コードチャンクやインラインコードを生成する必要があり, そしてチャンクヘッダの中身も掲載したいということもあるでしょう.

残念なことにコードチャンクをさらに別のバッククオートのレイヤで囲むことは出来ませんので, 代わりにチャンクヘッダに `` `r ''` `` を挿入して, ソースコード内でコードチャンクを無効化しなければなりません. これは **knitr** によって, **空の文字列**のインラインコードであるものと評価されます. 次のソース文書の中にある「コードチャンク」

````
```{r, eval=TRUE}`r ''`
1 + 1
```
````

は出力時にはこのようにレンダリングされます.

````
```{r, eval=TRUE}
1 + 1
```
````

空の文字列で置き換えられるため, インラインコードは消え去ります. しかしこれは第 1 歩にすぎません. 出力時になんらかの無加工のコードを表示するには, Markdown の構文はコードブロックで包まれているべきです (スペース 4 つ分のインデントかバッククオートによる囲みで). 上記の出力を見たいとき, 実際のソースは以下のようになります.

````
````
````
````

````
```{r, eval=TRUE}`r ''`
1 + 1
```
````

なぜバッククオートが 4 つなのでしょうか. これは N 個のバッククオートを包むには, 少なくとも N+1 個のバッククオートを使わなければならないからです.

5.6.1 インライン R コードをそのまま表示

行内のコードをそのまま表示する方法はいくつかあります. 最初の方法は `` `r `` の直後でインラインコードを改行することです. 例えばこのように.

````
これは出力時にインライン `R` コードをそのまま表示します `` `r
1+1` ``.
````

これが出力文書ではこうなっているはずです.

> これは出力時にインライン R コードをそのまま表示します `r 1+1`.

この小ワザは 2 つの理由で動作します. (1) Markdown パーサはしばしば単独の改行文字を単なるスペース 1 つとして扱う (2 連続の改行は新しい段落を始めることと比べてみてください) ということと, (2) **knitr** は `` `r `` をパースするのに直後にスペース 1 つを要求する, つまりここにスペースがないとインラインコードとして扱われないということです.

インライン R コードをそのまま表示する別の方法は, R コードを `knitr::inline_expr()` で包むことです. 例えば.

````
これで出力時にインライン `R` コードがそのまま表示されます
`` `r knitr::inline_expr("1+1")` ``.
````

私 (Yihui) は 2 つ目の方法をお薦めします. 1 つ目の方法は多かれ少なかれ Markdown 構文と **knitr** パーサに対するハック的なものだからです.

5.7 コードブロックに行番号を表示する (*)

attr.source = ".numberLines" でソースコードブロックにも行番号を付けることも, attr.output = ".numberLines" でテキスト出力ブロックに行番号を付けることもできます (これらのオプションの詳細は 11.13 節参照). 例えば.

````
```{r, attr.source='.numberLines'}
if (TRUE) {
  x <- 1:10
  x + 1
}
```
````

出力はこうなります.

```
01 if (TRUE) {
02   x <- 1:10
03   x + 1
04 }
```

HTML 出力では, Pandoc が提供するシンタックスハイライト のテーマ を選ぶ必要があることに注意してください. これは出力フォーマットの highlight オプションを default や textmate にすべきではないということを意味します. ヘルプページ ?rmarkdown::html_document でこのオプションの他の値の一覧を見ることができます. 例えばこう設定できます.

```
output:
  html_document:
    highlight: tango
```

bookdown の gitbook 出力フォーマットでは, コードの左側の適切な位置に行番号を表示するために CSS を多少調整する必要があるかもしれません. 以下は本書で使用しているものです (行番

号がページ左余白に近すぎると思ったら, `left` の値を `-0.2em` などに増やして調整してください).[*5]

```
pre.numberSource code > span > a:first-child::before {
  left: -0.3em;
}
```

revealjs の `revealjs_presentation` 出力フォーマット (El Hattab and JJ Allaire, 2017) に対しても CSS の調整が必要かもしれません.

```
.reveal pre code {
  overflow: visible;
}
```

カスタム CSS スタイルを HTML 出力に適用する方法がわからないなら, 7.1 節を見てください.

`startFrom` 属性で開始する数字を指定することもできます. 例えば.

````
```{r, attr.source='.numberLines startFrom="5"'}
if (TRUE) {
  1:10
}
```
````

現時点では Word 出力での行番号はサポートしていません.

5.8 多段組み (*)

Pandoc の Markdown はスライド文書に対する多段レイアウトをサポートしていますが, 他のタイプの文書ではサポートしていません. このレシピでは通常の HTML 文書や LaTeX 文書での多段レイアウトを使う方法を紹介します[*6]. これは **knitr** の issue https://github.com/yihui/knitr/

[*5] **訳注**: 日本語版は **rmdja** パッケージの出力フォーマットを使用しており, これはデフォルトで行番号を表示し, かつ gitbook に対応した調整を予め搭載しています.

[*6] **訳注**: 二段組にしたいのが PDF 限定であれば, YAML フロントマターのみで簡単に制御できるかもしれません (6.2 節参照).

issues/1743 での Atsushi Yasumoto[*7] の解決策に着想を得ました.

考慮する必要があるのが HTML 出力のみなら話はかなり単純です. 任意の HTML 要素を横に並べて表示するのは CSS を使えば比較的簡単にできるからです. コードチャンクのテキスト出力を横に並べるだけならば, もっと簡単になります. 以下は 1 つ目の例です.

```
---
output: html_document
---

```{r attr.source="style='display:inline-block;'", collapse=TRUE}
1:10  # 1 から 10 の数列
10:1  # その逆順
```
```

CSS 属性 `display: inline-block;` は, コードブロックの出力 (つまり HTML タグの `<pre>` です) をインライン要素として表示しなさいという意味です. デフォルトではこれらのブロックはブロックレベル要素 (つまり `display: block;`) として表示され, 行を丸ごと占有します. チャンクオプション `collapse = TRUE` はテキスト出力を R ソースコードブロックと結合することを意味するので, ソースとテキスト出力が同じ `<pre>` ブロックに配置されます.

HTML 出力時に任意の順で横に並べたい場合, Pandoc の fenced `Div`.[*8] を使うことができます. “Div” は HTML タグの `<div>` に由来しますが, 任意のブロックやコンテナと解釈できます. `Div` の開始と終了は は 3 つ以上のコロン (例: `:::`) です. より多くのコロンの `Div` は, よりコロンの少ない `Div` を含むことができます. fenced `Div` の重要で有用な機能は, これに属性を付与できるということです. 例えば CSS 属性 `display: flex;` を外側のコンテナに適用できるので, 内側のコンテナは横並びに配置されます.

```
---
output: html_document
---

:::: {style="display: flex;"}
```

[*7] **訳注**: @atusy のこと

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

```
::: {}
ここは **最初の** Div です.

```{r}
str(iris)
```
:::

::: {}
こっちは右側に配置されるブロックです.

```{r}
plot(iris[, -5])
```
:::

::::
```

上記の例では外側の Div (::::) は 2 つの Div (:::) を含んでいます. この中にさらに Div を追加することもできます. とても強力な CSS 属性 display: flex; (CSS Flexbox) についてもっと知るためには https://css-tricks.com/snippets/css/a-guide-to-flexbox/ というガイドを読めばよいでしょう. CSS グリッド (display: grid;) もまた強力で, 上記の例にも使えます. もし試してみたいなら, display: flex; を display: grid; grid-template-columns: 1fr 1fr; grid-column-gap: 10px; に置き換えてみてください. グリッドレイアウトについてもっと知りたければ, https://css-tricks.com/snippets/css/complete-guide-grid/ のガイドを見てください.

HTML でも LaTeX でも同じように使えるレイアウトにしたいのなら, よりトリッキーになります. 以下に HTML, LaTeX そして Beamer で使える用例の全容を示します.

```
---
output:
  pdf_document:
    latex_engine: lualatex
    keep_tex: true
```

```
    includes:
      in_header: columns.tex
  html_document:
    css: columns.css
  beamer_presentation:
    keep_tex: true
    latex_engine: lualatex
    includes:
      in_header: columns.tex
documentclass: "`r if(knitr::opts_knit$get('rmarkdown.pandoc.to') == 'beamer') 'beamer'
↪  else 'ltjsarticle'`"
mainfont: 'Noto Sans CJK JP'
---

# 二段組み

以下は 3 つの子要素の Div を横並びに持つ Div コンテナです．中央の Div は空で，左右の Div
↪  の間に空白を作るためだけに存在します．

::::::: {.cols data-latex=""}

::: {.col data-latex="{0.55\textwidth}"}
```{r, echo=FALSE, fig.width=5, fig.height=4}
par(mar = c(4, 4, .2, .1))
plot(cars, pch = 19)
```
:::

::: {.col data-latex="{0.05\textwidth}"}
\
<!-- 段どうしのセパレータとして機能するだけの空の Div（空白入り） -->
:::

::: {.col data-latex="{0.4\textwidth}"}
左側の図は `cars` データを表しています．
```

```
> いろはにほへと ちりぬるを
わかよたれそ つねならむ
うゐのおくやま けふこえて
あさきゆめみし ゑひもせす

:::
::::::
```

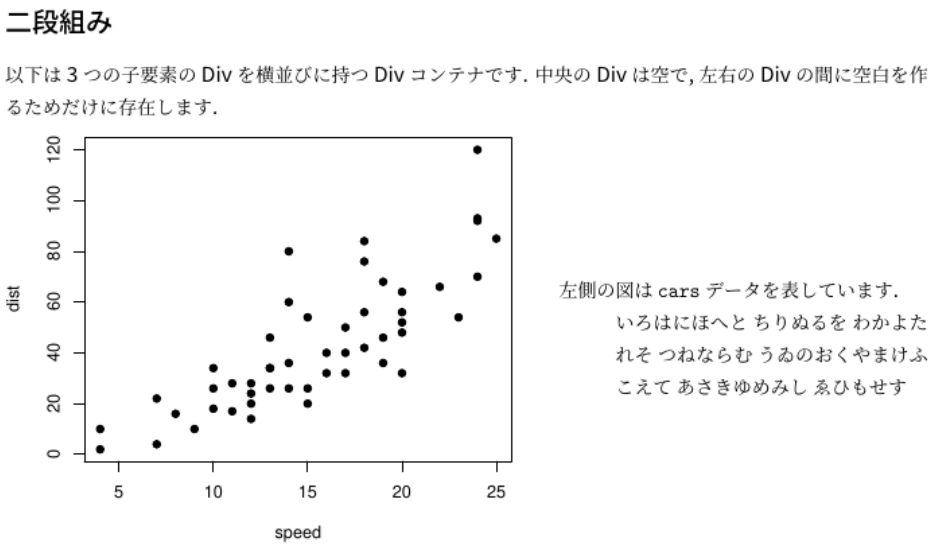


図 5.3: HTML, LaTeX, Beamer で動作する二段組み

図 5.3 がその出力です. この例では外側の `.cols` クラスを持つ `Div` と, 内側に `.col` クラスを持つ 3 つの `Div` を使っています. HTML 出力では, 外部 CSS ファイル `columns.css` を導入し, その中で Flexbox レイアウトを外側の `Div` に適用しているので, 内側の `Div` が横並びになります.

```
.cols {display: flex; }
```

LaTeX 出力 (`pdf_document`) では, `columns.tex` に含まれている「あまり行儀の良くない裏ワザ」を

LaTeX プリアンブルに適用し, LaTeX 環境 `cols` と `col` を定義しなければなりません.

```
\newenvironment{cols}[1][]{}{}

\newenvironment{col}[1]{\begin{minipage}{#1}\ignorespaces}{%
\end{minipage}
\ifhmode\unskip\fi
\aftergroup\useignorespacesandallpars}

\def\useignorespacesandallpars#1\ignorespaces\fi{%
#1\fi\ignorespacesandallpars}

\makeatletter
\def\ignorespacesandallpars{%
  \@ifnextchar\par
    {\expandafter\ignorespacesandallpars\@gobble}%
    {}%
}
\makeatother
```

`col` 環境が特に複雑な主な理由としては, LaTeX 出力で Pandoc は各 `Div` でいつも段落を改めるので, この改段を除去しなければならないからです. そうしないと `Div` を横並びに配置することはできません. このハックは https://tex.stackexchange.com/q/179016/9128 から借用しました.

Beamer 出力でも `columns.tex` で同じハックを適用しています. Pandoc は スライドショー[*9]用に `::: {.columns}`, `::: {.column}`, `::: {.incremental}` といった特別な `Div` を提供していることに注意してください. これらは特別な意味を持つため, この節のような方法で `Div` を LaTeX 環境に変換するときには, これらのタイプの `Div` を**使わない**ように注意しなければなりません. `columns` や `column` という名前の `Div` タイプを使わず, `cols`, `col` 使ったのは, これが理由です.

fenced `Div` についてより詳しく知りたいなら, 9.6 節を見てください.

[*9] https://pandoc.org/MANUAL.html#producing-slide-shows-with-pandoc

第6章

LaTeX 出力

多くの著作者にとって作品の主な出力は PDF レポートですが, この出力では強力な LaTeX のスタイル設定を活用できます. この章では, LaTeX コードやパッケージをプリアンブルに含めることや, カスタム LaTeX テンプレートの使用, ヘッダとフッタの追加, 図を分割して生成する方法, 生の LaTeX コードを文書の本文に書く方法, といった PDF レポートのカスタマイズに使えるアプローチについて議論します.

ただし, 始める前に注意しておきたいことがあります. R Markdown の恩恵の 1 つは単一のソース文書から複数のフォーマットの文書を生成できるということです. あなたの作品を単一の出力に対して仕立て上げるこにとよって, その出力フォーマット単体の見た目やパフォーマンスは向上するかもしれませんが, それはこの移植性を犠牲にすることでもあります. この問題は LaTeX に限ったことでなく. 他の出力フォーマットでも同様です.

6.1 プリアンブルに LaTeX コードを追加する

LaTeX 文書の一般的な構造はこのようになっています.

```
\documentclass{article}
% preamble
\begin{document}
% body
\end{document}
```

これは文書クラスを `\documentclass{}` で宣言し, 必要に応じて特定の LaTeX パッケージを読み込んだり特定のオプションをプリアンブルで設定し, そして `\begin{document}` に続いて文書の本文

を書き始めています．Markdown 文書はほとんどがこの文書の本文に対応します．

プリアンブルになにか追加したい時，pdf_document の includes オプションを使わねばなりません．このオプションは 3 つのサブオプションを持ちます．in_header，before_body，そして after_body です．いずれも 1 つ以上のファイルパスを指定できます．in_header に指定されたファイルはプリアンブルに追加されます．before_body と after_body に指定されたファイルはそれぞれ本文の前と後に追加されます．

例えば以下はテキスト内のハイパーリンクを脚注に変える小ワザです．この小ワザが役に立つのは，PDF 出力された文書が紙に印刷されたときに，読者は紙面上のリンク (\href{URL}{text} で生成されたもの) をクリックすることはできませんが，脚注で URL を見ることはできるからです．この小ワザはテキストと URL の両方を表示します．

```
% あなたはレンダリング前に \href のコピーを保存したいかもしれない
% \let\oldhref\href
\renewcommand{\href}[2]{#2\footnote{\url{#1}}}
```

上記のコードを任意のファイル名（例えば preamble.tex ）に保存してから，プリアンブルに読み込んでください．

```
output:
  pdf_document:
    includes:
      in_header: "preamble.tex"
```

この小ワザに限れば，実際に自分で実装しなくても，Pandoc のデフォルトの TaTeX テンプレート (6.2 節参照) に組み込まれた機能である YAML オプション links-as-notes を true にすることで簡単にできます．

コードをプリアンブルに追加する別の方法として，YAML フロントマター の header-includes フィールドに直接コードを与えることができます．6.3 節でその例を紹介しています．header-includes を使う利点は R Markdown 1 文書の内部に全てを含められることです．しかしレポートを複数の出力フォーマットで生成したいのなら，やはり includes を使う方法をお薦めします．header-includes は使われ方に制約がないため，非 LaTeX 出力の文書に対しても読み込まれてしまうからです．これと比較して，includes オプションは pdf_document フォーマットにのみ適用されます．

6.2 LaTeX 出力の Pandoc オプション

LaTeX 出力に対してデフォルトの Pandoc テンプレートを使うなら, PDF 出力の文書の見た目を調整するオプションが何種類もあります. そのうちいくつかの例を以下に挙げておきます. 完全なリストは https://pandoc.org/MANUAL.html#variables-for-latex で見ることができます.

```
documentclass: book
classoption:
  - twocolumn
  - landscape
papersize: a5
linestretch: 1.5
fontsize: 12pt
links-as-notes: true
```

あなたが LaTeX をある程度ご存知なら, これらのオプションの意味は明らかでしょう. documentclass オプション は, 例えば article, book, report などの文書クラスを設定します. classoption は文書クラスに与えたいオプションをリストにしたもので, 例えば二段組の文書を作りたいなら twocolumn オプション,[*1], 横置きレイアウトにするなら landscape オプション (デフォルトでは縦置き (portrait) レイアウト) があります. papersize オプションは a4, paper, a5 といった用紙サイズを設定します. linestretch オプションは行間を設定します. fontsize オプションはフォントサイズを 10pt, 11pt, 12pt というふうに設定します. links-as-notes オプションはテキスト内のリンクを脚注に置き換えます. 紙に印刷する際には読者は紙面上のリンクをクリックできませんが, 脚注の URL を見ることができるので便利です.

フォントの変更は少しトリッキーで, どの LaTeX エンジンを使っているかに依存します. LaTeX ベースの出力フォーマットで通常デフォルトの pdflatex を使っているのなら[*2], fontfamily オプションを使って読み込む LaTeX フォントパッケージを選択してください. 例えばこのように.

[*1] このオプションは文書全体を変更しますが, 特定の位置から再度一段組に戻したいのなら, そこに \onecolumn コマンドを挿入することになるでしょう. 二段組モードを続けたいなら \twocolumn を挿入します.

[*2] **訳注**: 日本語文書を **pdflatex** で出力することは全く不可能というわけではありませんが, 技術的制約が多いため LaTeX に慣れている方以外にはお薦めしません. **xelatex** または **lualatex** の使用をお薦めします.

```
fontfamily: accanthis
output:
  pdf_document:
    latex_engine: pdflatex
```

これで文書に Accanthis[*3] フォントが使われます. 他にも多数の LaTeX フォントパッケージのリストがあるので https://tug.org/FontCatalogue/ を見てください. LaTeX ディストリビューションに TinyTeX をお使いで, インストールされていないフォントパッケージが要求されるときは, 文書がコンパイルされる際に自動でインストールされるはずです (1.2 節参照).

LaTeX エンジンに `xelatex` または `lualatex` を使っているなら, ローカルのコンピュータで使用可能なフォントから選ぶことができ, LaTeX パッケージの追加インストールはしなくともよいです. YAML オプションで `mainfont`, `sansfont`, `monofont` を使えば, それぞれメインのフォント, サンセリフ体, そしてタイプライタ体のフォントを指定できます.[*4] 例えばこのように.

```
mainfont: Arial
output:
  pdf_document:
    latex_engine: xelatex
```

Beamer の文書は LaTeX 文書なので, Beamer でスライドを生成する時にもこれらのオプションを使用できます. 加えて, Pandoc は Beamer スライド用にオプションをいくつか追加提供しています. それらは https://pandoc.org/MANUAL.html#variables-for-beamer-slides で確認できます. 例えば `institute` オプションで著者の所属機関を指定することができます.

```
---
output: beamer_presentation
institute: "ハッカーの大学"
---
```

[*3] https://tug.org/FontCatalogue/accanthis/

[*4] **訳注**: **rmdja** パッケージでは YAML フロントマターで 3 種類のフォントをまとめて設定できたり, あるいは欧文用フォントと和文用フォントを個別に細かく指定できたりします. 詳細はパッケージのドキュメント等を参考にしてください.

6.3　表紙ページにロゴを置く

LaTeX パッケージの **titling** は表題ブロックを画像に変更するのに使えます．以下は R ロゴ (`logo.jpg`) を表紙に配置する方法の全容を示したものです．画像は LaTeX のサポートする形式 (例えば `jpg`, `png`, `pdf`) ならなんでも使えます．

```
---
title: LaTeX のタイトルにロゴを追加する
author: Michael Harper
date: 2018/12/7
output:
  pdf_document:
    latex_engine: lualatex
documentclass: ltjsarticle
header-includes:
  - \usepackage{titling}
  - \pretitle{\begin{center}
    \includegraphics[width=2in,height=2in]{logo.jpg}\LARGE\\}
  - \posttitle{\end{center}}
---

<!-- 改ページを含めることもできます．これで文書を強制的に 2 ページ目から始めさせます．-->

\newpage

ここからあなたのレポート

```{r, include=FALSE}
# R ロゴをカレントディレクトリにコピー
file.copy(file.path(R.home("doc"), "html", "logo.jpg"), '.')
```
```

図 6.1 がこの出力例です．

LaTeX パッケージ (**titling**) を特に要求しない代替方法として，Markdown 構文を使って `title`

図 6.1: LaTeX の表紙ページにロゴを追加する

フィールドに画像を挿入する方法があります. 例えばこのように.

```
title: |
  ![](logo.jpg){width=1in}
  LaTeX のタイトルにロゴを追加する
```

この場合, 最初の例にあった YAML フロントマターの `header-includes` フィールドは不要になります. 例からは見えませんが, `![](logo.jpg){width=1in}` の末尾にスペースが 2 つあることに注意してください. これは Markdown では改行を意味します (4.12 節参照). 改行がない場合画像とタイトルは同じ行に現れてしまい, あなたの意図するものではないはずです.

6.4 LaTeX パッケージを追加で読み込む

追加の LaTeX パッケージ を使うことで文書のスタイルに拡張的なカスタマイズが可能になります. 加えて **kableExtra** (Zhu, 2021) のようないくつかのパッケージでは R パッケージの関数が LaTeX に依存して機能するものもあります. R でもよくあるように, これらの関数を使えるように

なる前に R Markdown 文書内でパッケージを読み込む必要があります.

6.4.1 LaTeX パッケージを読み込む

pdf_document の YAML 設定で extra_dependencies オプション を使って追加の LaTeX パッケージを読み込めます. これにより中間出力の LaTeX 文書で読み込むべき LaTeX パッケージのリストを与えることができます. 例えばこのように.

```
---
title: "追加 LaTeX パッケージを使う"
output:
  pdf_document:
    extra_dependencies: ["bbm", "threeparttable"]
---
```

パッケージ読み込み時のオプションを指定する必要があるなら, 第 2 のレベルを加えてオプションをリストとして与えられます. 例えばこのように.

```
output:
  pdf_document:
    extra_dependencies:
      caption: ["labelfont={bf}"]
      hyperref: ["unicode=true", "breaklinks=true"]
      lmodern: null
```

これは LaTeX に慣れた人にとっては以下の LaTeX コードと同じです.

```
\usepackage[labelfont={bf}]{caption}
\usepackage[unicode=true, breaklinks=true]{hyperref}
\userpackage{lmodern}
```

6.1 節で紹介した includes 引数よりも extra_dependencies 引数を使う利点は, 外部ファイルを読み込む必要がないため, Rmd 文書が自己完結的になりうるということです.

6.4.2 パッケージの例

LaTeX には広範なコミュニティがあり Comprehensive TeX Archive Network[*5] (CTAN) 全体には 4,000 種類以上のパッケージがあります．ここにレポートづくりに使えるかもしれない LaTeX パッケージの例をいくつか挙げます．

- pdfpages[*6]: あなたの文書内に, 別の外部 PDF 文書からページを丸ごと持ってきて埋め込むことができます．
- caption[*7]: キャプションのサブタイトルを変更します．例えば図のタイトルをイタリックや太字にできます．
- fancyhdr[*8]: 全てのページのラニングタイトル (欄外見出し) を変更できます．

6.5 図の位置を制御する

LaTeX に共通の不満点の 1 つは図表の配置です．Microsoft Word のような図がユーザーの指定した場所にそのまま置かれるワードプロセッサと違い, LaTeX は特定の組版ルールに反しないように図を配置しようとします．そうなると図はテキストで参照した場所から浮動 (フロート) するかもしれません．この節では (図などの) フロート環境がどう機能するかについての背景情報と, その挙動をカスタマイズするためにどうオプションを与えればよいか解説します．

6.5.1 フロート環境

LaTeX ではデフォルトではキャプションのある図は `figure` 環境で生成されます．例えば Pandoc は以下の画像を含む Markdown コードを…,

```
![This is a figure.](images/cool.jpg)
```

こう変換します．

[*5] https://ctan.org
[*6] https://ctan.org/pkg/pdfpages
[*7] https://ctan.org/pkg/caption
[*8] https://ctan.org/pkg/fancyhdr

```
\begin{figure}
  \includegraphics{images/cool.jpg}
  \caption{This is a figure.}
\end{figure}
```

figure 環境はフロート環境です．フロートの詳細な説明は https://en.wikibooks.org/wiki/LaTeX/Floats,_Figures_and_Captions で読むことができます．要約するとフロートは，図や表のようにページで区切られないコンテナとして使われます．図表が現在のページの余白に収まらないときには，LaTeX は次のページの先頭に配置します．図が十分に縦長だと，テキストを数行分の余白が残っていたとしても，次のページ全てを占有します．この挙動は，\begin{figure}[b] のように，\begin{figure}の後の角カッコ内のいくつかの配置指定修飾子によって制御できます．以下は使用可能な記号のリストです．

- h: フロートを**ここ** (here) に配置します．つまりソーステキスト上に現れるところとほぼ同じ位置です．
- t: そのページの**先頭** (top) に配置します．
- b: そのページの**末尾** (bottom) に配置します．
- p: フロート専用の特別な**ページ** (page) に配置します．
- !: LaTex が「良い」フロートの位置を決定するための内部パラメータ上書きします．
- H: フロートを正確に LaTex コード上と同じ位置に配置します．**float** パッケージが必要です (\usepackage{float})．

これらの修飾子は併用できます．例えば !b は LaTeX が図をページ末尾に置くよう強制できます．デフォルトの挙動は tbp です．これは LaTeX が図をまずページ先頭に，ついで末尾に，そして独立したページに置こうとします．

6.5.2 図がフロートするのを防ぐ

多くのユーザは初めに，伝統的なワードプロセッサの挙動を再現できるよう，文書内を図が移動するのを防ぎたくなります．これを実現するには，まず LaTeX パッケージの **float** を読み込まなければなりません．YAML に以下の記述を含めることでできます．

```
output:
  pdf_document:
```

```
    extra_dependencies: ["float"]
```

チャンクオプション fig.pos をフロートの挙動を制御するのに使えます．オプションの値 !H は文書でのいかなる移動も防ぎます．以下の行を　R Markdown 文書の最初のコードチャンクに書くことで，全てのチャンクがこの設定になるように，文書のデフォルトの挙動を設定できます．

```
01 knitr::opts_chunk$set(fig.pos = "!H", out.extra = "")
```

一般論として，LaTeX の図のフロートを強制的にやめさせることをおすすめしません．よくある要望なので，本書にこの解決策を盛り込んだのですが，[*9] LaTeX が図をフロートできないときにはいくつかの深刻な副作用が発生することがあります．

6.5.3　フロートを後回しに強制する

全てのフロートを固定するよう強制する代わりに，テキストの後ろにフロートが回るよう強制する方法があります．これはよくある問題を排除できます．問題とは関連するテキストが現れるよりも前に図がページの先頭に現れてしまうということで，こうなるとレポートを読む流れが破壊されてしまいます．LaTeX パッケージの **flafter** を使って以下のようにすることで，常に図がテキストより後に現れるよう強制できます．

```
output:
  pdf_document:
    extra_dependencies: ["flafter"]
```

6.5.4　LaTeX 配置ルールを調整する (*)

LaTeX のフロート配置パラメータの初期値は，あなたにとっては「理にかなった」配置を全体的に邪魔しているかもしれません．堅実などころか悪質なまでに．これらのデフォルト設定を表 6.1 に示します．

[*9] 関連するスタック・オーバーフローの質問 https://stackoverflow.com/q/16626462/559676 は 45,000 回以上閲覧されました．

表 6.1: LaTeX デフォルトのフロート設定

コマンド	概要	デフォルト
topfraction	ページ先頭からフロートが占めるページ割合の最大値	0.7
bottomfraction	ページ末尾からフロートが占めるページ割合の最大値	0.3
textfraction	1 ページに占めるテキストの割合の最小値	0.2
floatpagefraction	1 ページに占めるフロートの割合の最小値	0.5
topnumber	ページ先頭のフロート最大数	2
bottomnumber	ページ末尾のフロート最大数	1
totalnumber	1 ページの最大フロート数	3

LaTeX に図を動かさないよう努力してもらうために, これらの設定を変えることができます. LaTeX プリアンブルファイルに, 1 ページのテキストの最小量を減らすような以下のコードを追加し, フロートが収まる余地を増やすことができます.

```
\renewcommand{\topfraction}{.85}
\renewcommand{\bottomfraction}{.7}
\renewcommand{\textfraction}{.15}
\renewcommand{\floatpagefraction}{.66}
\setcounter{topnumber}{3}
\setcounter{bottomnumber}{3}
\setcounter{totalnumber}{4}
```

これらの記述を `.tex` ファイルに追加したら, 6.1 節で紹介した方法で LaTeX 文書のプリアンブルで読み込ませることができます.

6.6 LaTeX で複数の図をまとめる

複数の画像を 1 つの画像環境に含めたいときがあるかもしれません. 複数の画像 (sub-figures, サブ図) を 1 つの環境に配置しそれぞれに副題を与えることで, サブ図をまとめることができます.

複数の図をまとめるには LaTeX パッケージの **subfig** が必要です. `pdf_document` 出力の YAML オプションの `extra_dependencies` で読み込ませることができます. 例は以下のようになります.

```
---
output:
  pdf_document:
    extra_dependencies: "subfig"
---
```

あるコードチャンクからの全てのプロットを並べるためには, チャンクオプション `fig.cap` (環境全体のキャプション) と `fig.subcap` (サブ図のためのキャプションの文字列ベクトル) を使わなければなりません. 最良の出力を得るためには, 以下のような選択肢も使用できます.

- `fig.ncol`: サブ図の列の数です. デフォルトでは全てのグラフが単一の行に並べられます. これを使って複数の行に分けられます.
- `out.width`: 個別のグラフの出力幅です. 通常はこれを `100%` を列の数で割ったものに設定します. 例えば 2 つグラフがあるなら, `out.width` は `50%` 以下にすべきです. そうしないとグラフはページの外枠をはみ出すかもしれません.

以下は具体例の 1 つです.

```
---
output:
  pdf_document:
    extra_dependencies: "subfig"
---
```

```
```{r, fig.cap='Figure 1', fig.subcap=c('(a)', '(b)', '(c)')}
plot(1:10)
plot(cars, pch = 19)
boxplot(Sepal.Width ~ Species, data = iris)
```
```

この出力を図 6.2 に示します. 簡潔にするために, 上記の例はチャンクヘッダの `fig.ncol = 2`, `out.width = "50%"`, `fig.align = "center"` や長くなるキャプションなどのチャンクオプションをい

くつか省略しています.

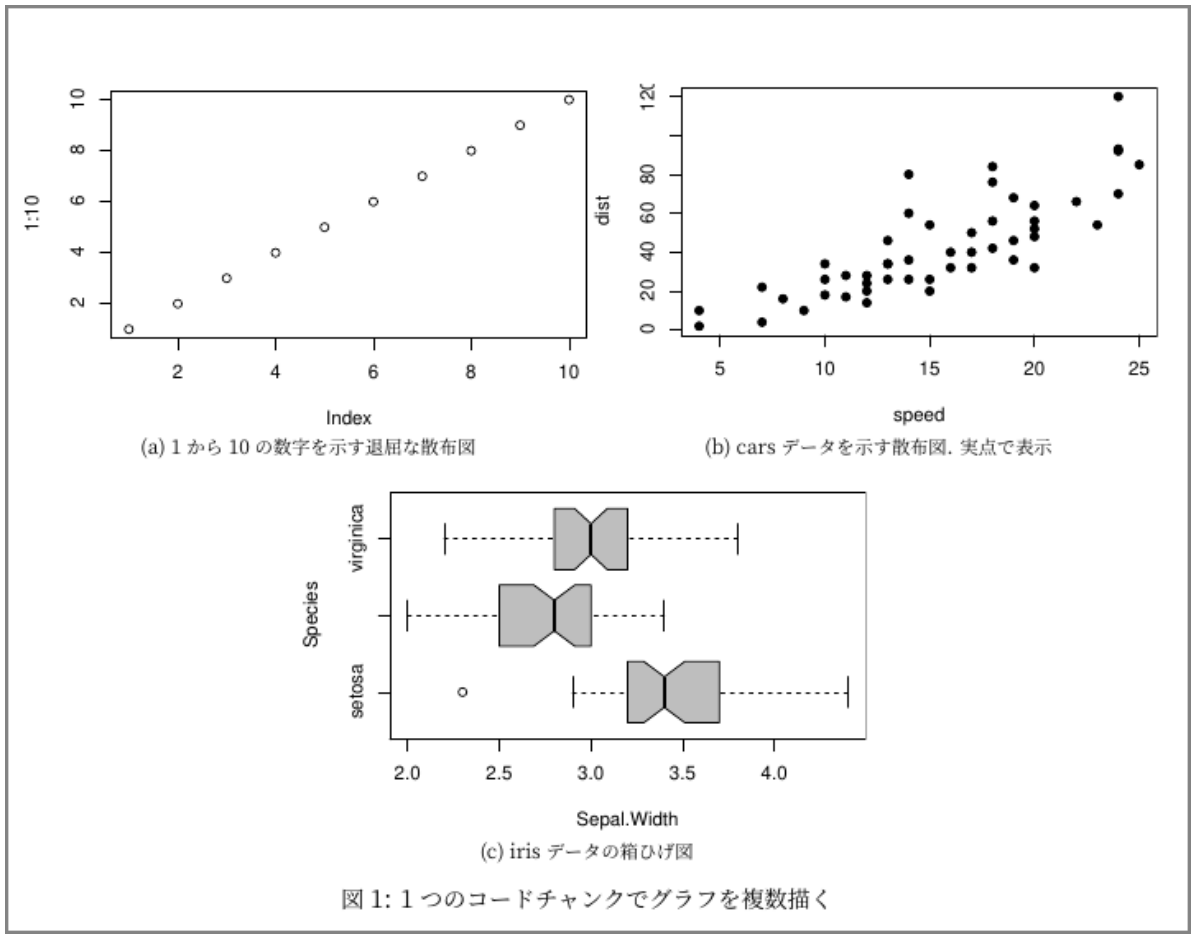


図 6.2: 複数の図を含む単一の figure 環境の例

6.7 Unicode 文字を含む文書をレンダリングする

```
! Package inputenc Error:
  Unicode char \u8: not set up for use with LaTeX.
```

もしこのようなエラーにでくわしたら, おそらくデフォルトの LaTeX エンジンである `pdflatex` を使って文書 (中間ファイルの `.tex`) を PDF へレンダリングしているのでしょう. `pdflatex` はそのファイルにある何らかの Unicode 文字を処理できません. このようなときは, `xelatex` か `lualatex` へ切り替えることになるでしょう. 例えばこのように.

```
output:
  pdf_document:
    latex_engine: xelatex
```

他の文書出力フォーマットの LaTeX エンジン, 特に pdf_document ベースの bookdown::pdf_document2 や tufte::tufte_handout といったもののエンジンも変更できます. 例えばこのように.

```
output:
  bookdown::pdf_document2:
    latex_engine: lualatex
  tufte::tufte_handout:
    latex_engine: xelatex
```

6.8 LaTeX のコードフラグメントを生成する

もしはじめから純粋な LaTeX 文書で作業していたとしても, R Markdown はやはり便利だとわかることもあるでしょう. R Markdown で書いて, 文書を他の LaTeX 文書に読み込める LaTeX のコード片 (フラグメント) に変換したほうが便利なこともあります.

Rmd 文書を LaTeX にレンダリングするとき, \documentclass{}, \begin{body}, \end{body} を含む完全な LaTeX 文書が生成されます. フラグメントはこの完全な文書の主に本文の部分です. LaTeX フラグメントをレンダリングするのに, latex_fragment 出力フォーマットが使えます. 例えばこのように.

```
---
output: latex_fragment
---
```

これは .tex ファイルをレンダリングします. 例えば foo.Rmd は foo.tex にレンダリングされ, 別の LaTeX 文書で \input{foo.tex} を使うことでフラグメントを読み込めます.

6.9 カスタムヘッダとフッタ (*)

LaTeX パッケージの **fancyhdr** は文書のヘッダとフッタをカスタマイズするいくつかのコマンドを提供します. より完全なガイドとして, https://ctan.org/pkg/fancyhdr の完全版ドキュメントを参照してください. 最初に, パッケージを読み込みます. それからヘッダのスタイルを変えます. 例えばこのように.

```
\usepackage{fancyhdr}
\pagestyle{fancy}
```

このパッケージは異なる 3 つのインターフェースを提示します. ここでは `\fancyhead` と `\fancyfoot` コマンドを使います. 形式を決める構文は `\fancyhead[selectors]{output text}` で, カスタマイズしたいヘッダの箇所をセレクタが宣言しています. ページの位置を指定する以下のようなセレクタが使えます.

- **E** 偶数ページ
- **O** 奇数ページ
- **L** ページ左側
- **C** ページ中央
- **R** ページ右側

例えば `\fancyhead[LE,RO]{`**あなたの名前**`}` は偶数ページの頭の左側と, 奇数ページの頭の右側に「あなたの名前」と印字します. さらに LaTeX コマンドを織り交ぜることで, 各ページの詳細情報を取りだすことができます.

- `\thepage`: 現在のページ番号
- `\thechapter`: 現在の章番号
- `\thesection`: 現在の節番号
- `\chaptername`: 英語の “Chapter” の単語, あるいは現在の言語でそれに対応するもの, または著者がこのコマンドを再定義してできたテキスト.
- `\leftmark`: 大文字で現在のトップレベル構造の名前と番号.
- `\rightmark`: 大文字で現在のトップレベル構造に次ぐレベルの名前と番号.

以下は LaTeX コードの例で, 6.1 節で紹介した方法でプリアンブルに書き加えることができます.

```
\usepackage{fancyhdr}
\pagestyle{fancy}
% ヘッダ中央
\fancyhead[CO,CE]{Your Ðocument Header}
% フッタ中央
\fancyfoot[CO,CE]{And this is a fancy footer}
% 偶数ページ左と奇数ページ右にページ番号
\fancyfoot[LE,RO]{\thepage}
```

デフォルトではヘッダとフッタは PDF 文書の最初のページには表示されません. 表示にもフッタを表示したいなら, もう 1 行 `\fancypagestyle{plain}{\pagestyle{fancy}}` を追加しなければなりません.

6.10 Pandoc の LaTeX テンプレートをカスタマイズする (*)

Pandoc はテンプレート を通じて Markdown を LaTeX に変換します. テンプレートは Pandoc 変数を含む LaTeX ファイルであり, Pandoc はこれらの変数を値に置き換えます. 以下は `$body$` という変数を 1 つだけ含んだ単純なテンプレートです.

```
\documentclass{article}
\begin{document}
$body$
\end{document}
```

`$body$` の値は Markdown ドキュメントの本文から生成された LaTeX コードです. 例えば Markdown で本文が `Hello **world**!` ならば, `$body$` の値は `Hello \textbf{world}!` となります.

6.1, 6.2, 6.4 節で紹介した LaTeX のカスタマイズ方法だけでは不十分なら, 代わりにカスタムテンプレートを使ってみてください. テンプレートはその内部に任意の LaTeX コードを使うことが可能なので, はるかに柔軟です. テンプレートを使うには, `pdf_document` の `template` オプション にテンプレートのパスを含めます.

```
output:
  pdf_document:
    template: my-template.tex
```

Pandoc のデフォルトの LaTeX テンプレートは https://github.com/jgm/pandoc/tree/master/data/templates で見ることができます (ファイル名は `default.latex`). 自分でテンプレートを作成したい場合, このテンプレートから作りたいと思うことでしょう.

Pandoc 変数 (`$body$` や `$title$` など) の完全なリストとその意味は Pandoc マニュアルの https://pandoc.org/MANUAL.html#templates で見ることができます. 任意のカスタム変数を使うこともでき, それは典型的には YAML メタデータからテンプレートへと与えられます. もし具体例で学びたいなら, **MonashEBSTemplates** パッケージ (https://github.com/robjhyndman/MonashEBSTemplates) を確認することもできます. これはいくつかのカスタム LaTeX テンプレートを提供しています. これらのテンプレートは `inst/rmarkdown/templates/*/resources/` ディレクトリ (`*` はテンプレート名を指します) 以下にあります. 例えば出力フォーマット `MonashEBSTemplates::memo` 用のテンプレートは YAML メタデータの変数 `branding` を使って, モナシュ大学のブランドロゴを含むかどうかを制御できます. 以下のようにテンプレート内で `if` 文を使うことで実現しています.

```
$if(branding)$%
\includegraphics[height=1.5cm]{monash2}
\vspace*{-0.6cm}
$else$
\vspace*{-1cm}
$endif$
```

6.11 生の LaTeX コードを書く

デフォルトでは Pandoc は LaTeX へ変換する時, 文書内の LaTeX コードを維持するので, Markdown 内で LaTeX コマンドや環境を使うことができます. しかし, LaTeX コードが Pandoc がパースするには複雑過ぎる場合には, Pandoc は通常の Markdown として扱います. 結果として特別な LaTeX の文字はエスケープされます. 例えばバックスラッシュ `\` は `\textbackslash{}` に変換されるかもしれません.

Pandoc が Markdown 文書内の生の LaTeX コードに確実に手を付けないようにするには, コードを fenced block で囲み, `=latex` の属性を付けることができます. 例えばこのように.

```
```{=latex}
\begin{tabular}{ll}
A & B \\
A & B \\
\end{tabular}
```
```

`latex` の前の等号を忘れないでください. つまり `latex` ではなく `=latex` です. この機能は Pandoc 2.0 以降のバージョンが必要です (`rmarkdown::pandoc_version()` で確認してください).

6.12 ハードコア LaTeX ユーザーのために (*)

R Markdown はきっと執筆と組版のための最善の文書フォーマットではないでしょう. シンプルさは長所であると同時に短所でもあります. LaTeX はタイプすべきコマンドの多さと引き換えに, 組版の観点で Markdown よりはるかに強力です. あなたにとって組版がはるかに優先すべき事項で, あらゆる LaTeX コマンドや環境を使うことに満足しているのなら, 文書全体で Markdown を使う代わりに純粋な LaTeX コードを使えばよいのです.

knitr パッケージは R Markdown に限定されない多様なソース文書フォーマットをサポートしています. 以下は R コードと純粋な LaTeX コードが混ざり合っている例です

```
\documentclass{article}
\usepackage[T1]{fontenc}

\begin{document}

これがコードチャンクです.

<<foo, fig.height=4>>=
1 + 1
par(mar = c(4, 4, .2, .2))
```

```
plot(rnorm(100))
@

インラインコードを書くこともできます．例えば $\pi=\Sexpr{pi}$ とか，
 \Sexpr{1.9910214e28} で大きな数値を表現できます．

\end{document}
```

例えば上記のファイルが `latex.Rnw` であるようにファイル名には通常 `.Rnw` という拡張子がつきます．考え方は同じですが R コードチャンク構文とインライン R コードを書く構文とは異なっています．R コードチャンクは `<<>>=` で始まり (チャンクオプションは括弧内に書きます), `@` で終わります．インライン R コードは `Sexpr{}` 内に書きます．

`knitr::knit()` 関数は `Rnw` 文書を出力ファイルである LaTeX (`.tex`) にコンパイルでき，それをさらに `pdflatex` といった LaTeX ツールを通して PDF にコンパイルできます．`.Rnw` から PDF を一足飛びでコンパイルするのに `knitr::knit2pdf()` を使うこともできます．RStudio を使っているならツールバーの `Compile PDF` を押すこともできます．注意してほしいのは，Rnw 文書をコンパイルする方法のデフォルトは Sweave であり，これを **knitr** に変更することです (その方法はこの投稿 http://stackoverflow.com/q/27592837/559676 を確認してください)．

`Rnw` 文書は LaTeX のフルパワーをあなたにもたらします．Markdown ではほんとうに解決の難しい組版の問題があるのなら，これは最終手段となるでしょう．ただし，Markdown をやめる前に，カスタム Pandoc LaTeX テンプレート (6.10 節参照) もまた役に立つかもしれない，ということも覚えておいてください．

第7章

HTML 出力

LaTeX と比べて HTML はおそらくページに分けた出力の組版が苦手です. しかし, 特に CSS や JavaScript と連携すれば, 結果を見せつける際にははるかに強力になります. 例えば HTML にインタラクティブアプリケーションを埋め込んだり, 動的に HTML ページの外観や, 内容すら変えることができます. CSS と JavaScript の小ワザは, HTML 出力においては有用ながらもシンプルですが, LaTeX 出力で再現するのがとても難しいこともあります (しばしば不可能なこともあります).

この章では, カスタム CSS の適用方法, カスタム HTML テンプレートの使い方, コードブロックのスタイル変更や折りたたみ, 表の内容の並び替え, そして HTML ページへのファイル埋め込みといった, R Markdown の HTML 出力を改良するテクニックを紹介します.

7.1 カスタム CSS を適用する

HTML 文書の外観をカスタマイズしようと思うのなら, CSS と JavaScript を少しでも勉強することを強くお勧めします. **blogdown** 本 (Xie Hill, and Thomas, 2017) の Appendix B[*1] には HTML, CSS, JavaScript の簡単なチュートリアルがあります.

初心者にとっては, CSS のセレクタと優先度のルールを理解することは極めて重要です. さもなければ自分のカスタム CSS が意図したように機能しないことに混乱することになるでしょう (おそらく優先度が十分でないから).

Rmd 文書に 1 つかそれ以上のカスタムスタイルシートを読み込ませるには, 次の例のような `css` オプション を使うことができます.

[*1] https://bookdown.org/yihui/blogdown/website-basics.html

```
output:
  html_document:
    css: "style.css"
```

複数のスタイルシートを読み込ませるには, このようにブラケットで囲んだリストを使います.

```
output:
  html_document:
    css: ["style-1.css", "style-2.css"]
```

あるいは, css コードチャンク を使って, Rmd 文書に 直接 CSS のルールを埋め込むこともできます. 例えばこのように.

````
ここに `css` コードチャンクを埋め込む.

```{css, echo=FALSE}
p {
  font-size: 32px;
}
```
````

チャンクオプション `echo = FALSE` は CSS コードを出力にそのまま表示させないことを意味しますが, CSS コードを含む `<style>` タグは HTML 出力ファイルには生成されます.

7.2 セクションヘッダを中央揃えにする

7.1 節で言及した応用方法のように, CSS を見出しのアラインメントに使うことができます. 例えば以下のような CSS コードを使って, レベル 1 から 3 の見出しを中央揃えにできます.

```
h1, h2, h3 {
  text-align: center;
```

```
}
```

Rmd 文書に CSS を適用する方法は 7.1 節を見てください.

7.3 コードチャンクのスタイルを変更する

チャンクオプションの class.source と class.output を使えば, それぞれコードチャンクおよびそのテキスト出力のスタイルをカスタマイズできます. これらのオプションはクラス名の文字列ベクトルを取ります (11.13 節参照). 例えば class.source = "important" は, コードチャンクを含んでいる HTML 要素が出力される時に important というクラス名を持たせます. そこでこのクラスに CSS ルールを定義できます.[*2] このルールは特定のコードチャンクやテキスト出力を強調したいときに役に立ちます.

R Markdown の HTML 出力は, デフォルトで, Bootstrap フレームワークを読み込みます. Bootstrap は "bg-primary", "bg-success", "bg-info", "bg-warning", "bg-danger" といったいくつかの 背景に対する CSS クラス[*3] が定義済みのため, コードと出力の外観の変更を容易にしてくれます.

以下はチャンクオプション class.source = "bg-danger" と class.output = "bg-warning" を使った例で, その出力は図 7.1 で見られます.

```
---
title: チャンクのスタイルを変更する
output: html_document
---

データフレームの一部を取りだすとき，必ずしもデータフレームが返されるとは限りません．例えば 2 つの列を取りだすなら，データフレームを得ますが，1 つの列を取り出そうとするときは，ベクトルを得ます．

```{r class.source="bg-danger", class.output="bg-warning"}
mtcars[1:5, "mpg"]
```
```

[*2] CSS ではクラスは先頭にピリオド (.) を付けるため, この場合はルールは .important から始まります.

[*3] https://getbootstrap.com/docs/3.4/css/#helper-classes

```
常に確実にデータフレームを得るようにするには，`drop = FALSE` 引数を使わなければなりませ
↪ ん．ここで，チャンクオプション `class.source = "bg-success"` を使います．

```{r df-drop-ok, class.source="bg-success"}
mtcars[1:5, "mpg", drop = FALSE]
```
```

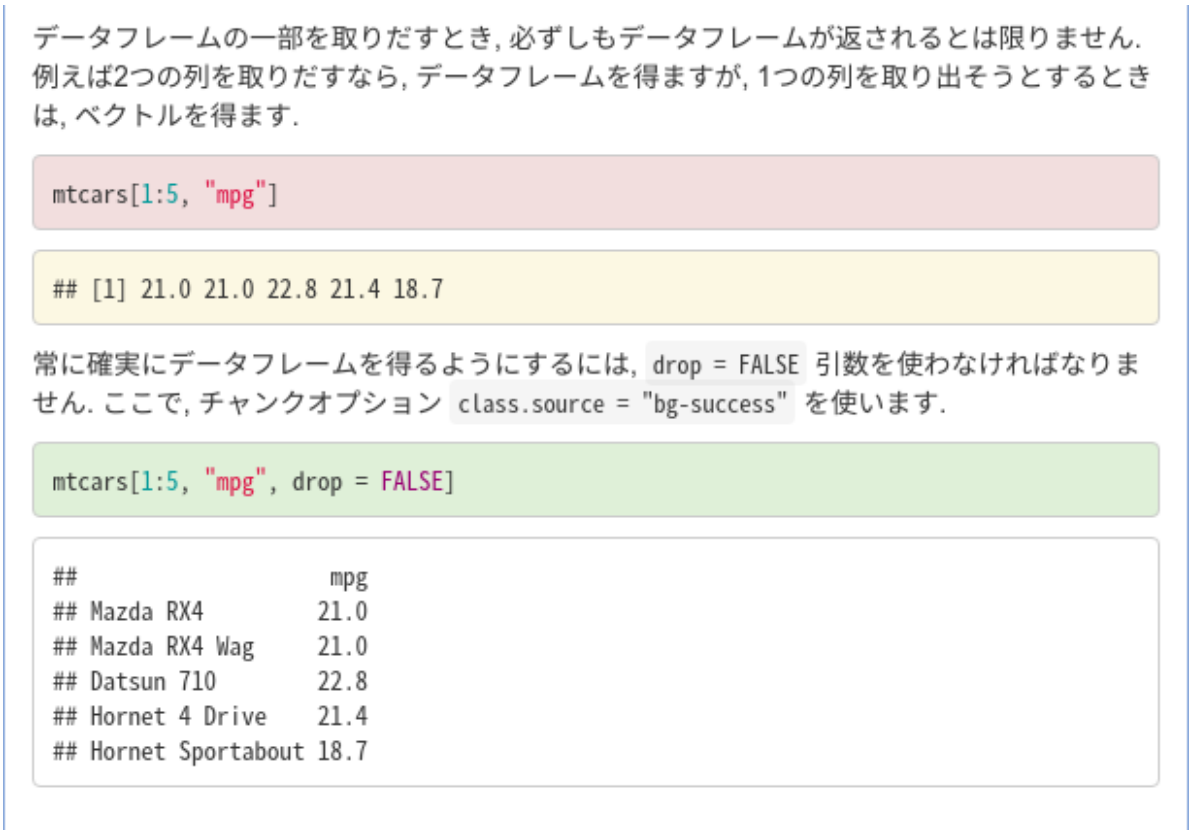


図 7.1: Bootstrap で定義された背景色を使ったコードチャンクと出力ブロック

任意のクラスを使って対応する CSS ルールを定義することもできます．この場合以下の例のように，7.1 節で言及した方法を使ってカスタム CSS ルールを読み込ませなければなりません．

```
---
title: チャンクにカスタムクラスを割り当てる
output: html_document
```

````
---

まず `watch-out` というクラスにいくつか CSS を ルールを定義します.

```{css, echo=FALSE}
.watch-out {
  background-color: lightpink;
  border: 3px solid red;
  font-weight: bold;
}
```

それからチャンクオプション `class.source` で `watch-out` クラスをコードチャンクに割り当て
↪ ます.

```{r class.source="watch-out"}
mtcars[1:5, "mpg"]
```
````

図 7.2 が出力されたスタイルです.

まず watch-out というクラスにいくつか CSSを ルールを定義します.

それからチャンクオプション class.source で watch-out クラスをコードチャンクに割り当てます.

```
mtcars[1:5, "mpg"]
```

```
## [1] 21.0 21.0 22.8 21.4 18.7
```

図 7.2: 明桃色の背景, 赤い太枠線をもつコードチャンク

文書内の全てのコードブロックにカスタムスタイルを適用したいなら, グローバルな **knitr** オプションで class.source を設定します. 例えばこのように.

```
knitr::opts_chunk$set(class.source = "watch-out")
```

複数のクラスをコードブロックに適用できます．例えば `class.source = c("important", "warning")` でコードブロックに “important” と “warning” という 2 つのクラスを持たせることができます．

コードブロック全体ではなく，内部の個別の要素を装飾したいならば，**flair** パッケージ (Bodwin and Glanz, 2020) の使用を検討するとよいでしょう．このパッケージでコードの個別の部分 (特定の文字，関数名，引数など) をカスタムスタイル (例えば色，フォントサイズ，あるいはフォントのウエイト) で強調できます．

7.4 コードブロックをスクロール可能にする (*)

大量のコードやテキスト出力を HTML ページに表示するとき，表示範囲の高さを制限したほうがよいでしょう．そうしないとページはとてつもなく長くなり，コードやテキストの出力を細かく読む気のない人が読み飛ばしづらくなります．この問題の解決法は複数あります．一つは，`html_document` フォーマットの `code_folding` オプションを使います．このオプションは文書のコードブロックを折りたたんで出力し，読者はボタンを押して折りたたみを展開することができます (詳細は 7.5 節)．

他の解決法としては，コードブロックが長すぎるとき，高さを固定しスクロール可能にすることです．これは CSS プロパティの `max-height` と `overflow-y` で実現できます．以下はその使用例の全容で，出力は図 7.3 になります．

````
---
title: スクロール可能なコードブロック
output: html_document
---

```{css, echo=FALSE}
pre {
  max-height: 300px;
  overflow-y: auto;
}
```
````

````
pre[class] {
  max-height: 100px;
}
```

コードブロックの高さを制限する CSS ルールをいくつか定義しました．これらのルールがコードブロックとテキスト出力に対して機能するのかテストすることができます．

```{r}
# このチャンクに多くのコードがあるように見せかける
if (1 + 1 == 2) {
  # もちろん真になる
  print(mtcars)
  # 単に長いデータを表示させるだけ
}
```

次に高さを 100px に制限するために `scroll-100` という新しいクラスにルールを追加し， チャンクオプション `class.output` でこのクラスをコードチャンクの出力に追加します．

```{css, echo=FALSE}
.scroll-100 {
  max-height: 100px;
  overflow-y: auto;
  background-color: inherit;
}
```

```{r, class.output="scroll-100"}
print(mtcars)
```
````

上記の例では全てのコードブロックに大域的に 300px の高さの上限を定義しています．HTML 出力時にはコードブロックが <pre> タグで囲まれていることを思い出してください．それから class 属性を用いて <pre> ブロックの高さを 100px に制限します．これは CSS セレクタ pre[class] が
````

図 7.3: カスタム CSS を使用したスクロール可能なコードブロック

意味するところです. デフォルトではテキスト出力は `<pre> </pre>` に含まれ, R コードブロックは `<pre class="r"> </pre>` に含まれます (`<pre>` タグが `class` 属性を持っていることに注意).

２つ目の R コードチャンクからのテキスト出力の高さも 100px です. これは出力に対して, カスタムクラス名 `scroll-100` を割り当て, 高さの上限を 100px に定義したためです.

個別のコードブロックに対して異なる最大の高さを指定したいならば, 12.3 節の例を見てください.

7.5　全コードブロックを折りたたみ, かついくつかは表示する

出力文書に書かれたコードブロックが読者に嫌がられるおそれがあるなら, はじめは折りたたんでおくという選択がいいでしょう. 読者はボタンを押して表示を選ぶことができます.

```
output:
  html_document:
    code_folding: hide
```

全てのコードブロックを最初から展開しておくこともできます (よって読者は折りたたみもできます).

```
output:
  html_document:
    code_folding: show
```

最初から全てのコードブロックを折りたたむなら, 特定のブロックだけチャンクオプション `class.source = "fold-show"` を使い最初から展開させておくこともできます. このように.

````
---
title: Hide all code blocks and show some initially
output:
  html_document:
    code_folding: hide
---

```{r}
1  # code is hidden initially
```

```{r class.source = 'fold-show'}
````
````

```
2  # code is shown initially
```

```{r}
3  # also hidden
```
```

反対のこともできます. つまり, 最初から全てのコードブロックを表示するものの, 一部を非表示にします. 例えばこのように.

```
---
output:
  html_document:
    code_folding: show

```{r}
1  # code is shown initially
```

```{r class.source = 'fold-hide'}
2  # code is hidden initially
```
```

code_folding は R と他の一部の言語 (デフォルトでは r, python, bash, sql, cpp, stan, and julia) コードチャンクの表示・非表示の挙動を制御します. これがあなたにとって制約になるなら, それ以外の言語のチャンクオプションの class.source に foldable クラスを追加するか, または いずれかのチャンクで knitr::opts_chunk$set(class.source = "foldable") を使ってグローバルに折りたたみの適用範囲を広げることができます.

```
---
title: CSS コードチャンクを隠す
output:
```
```

```
  html_document:
    code_folding: hide
---
```

これは `` `css` `` エンジンを使用した CSS コードチャンクです．これは `` `code_folding = 'hide'` `` を指定しても隠すことができません．

````
```{css}
pre {
  background-color: lightblue;
}
```
````

しかし `` `foldable` `` クラスをチャンクのソースコードに追加すれば可能になります．

````
```{css, class.source = 'foldable'}
pre.foldable {
  background-color: lightgreen;
}
```
````

7.6 内容をタブブラウジングさせる

HTML レポートで並列しているセクションをうまくまとめるには，タブセットが自然な方法として使えます．これは読者がページをスクロールして戻したり進めたりするかわりに，タブのタイトルをクリックすることで異なるセクションの内容を閲覧することを可能にします．

セクションをタブにするには，タブに変換する見出しより 1 レベル上の見出しにクラス属性 `.tabset` を追加します．例えばレベル 2 の見出しに `.tabset` を追加すると，それ以降のレベル 3 の見出しが全てタブに変換されます．以下は用例の全容です．

```
---
title: 内容をタブにまとめる
```

````
output: html_document
---

`html_document` 出力で並列するセクションをタブにできます.

## 結果 {.tabset}

### グラフ

このセクションでは散布図を表示します.

```{r, fig.dim=c(5, 3)}
par(mar = c(4, 4, .5, .1))
plot(mpg ~ hp, data = mtcars, pch = 19)
```

### 表

このタブではデータを表示します.

```{r}
head(mtcars)
```
````

出力を図 7.4 に示します. 実際には一度に 1 つのタブしか見られないことに注意してください. この図は両方のタブがみられるよう 2 つのスクリーンショットを連結したものです.

タブに “pill” 効果を付けるため, さらに別の属性 `.tabset-pills` を上位レベルの見出しに追加することができます. これでタブは暗青色の背景になります.

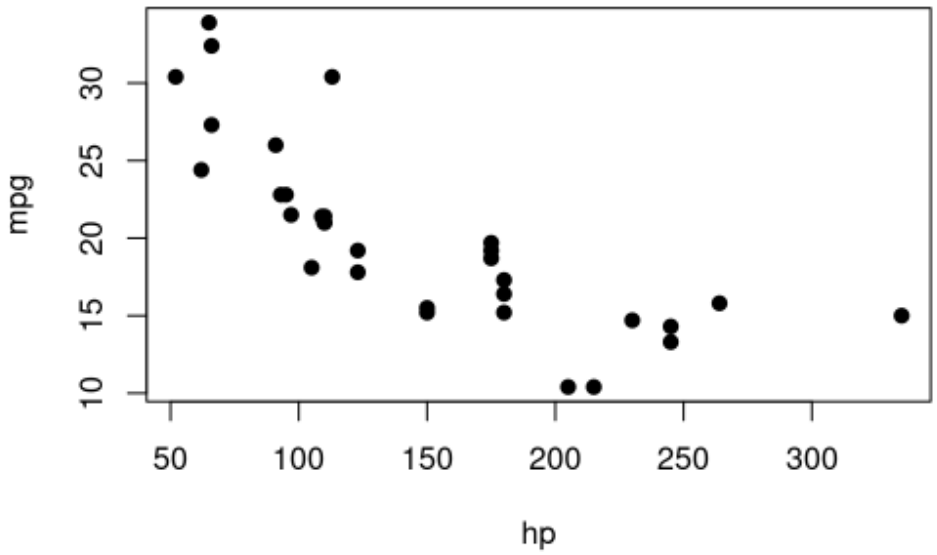
```
## Results {.tabset .tabset-pills}
```

デフォルトでは最初のタブがアクティブ (つまり表示されている) です. 他のタブを最初に表示させたいなら, そのセクションに `.active` 属性を追加できます. 以下の例では, 文書を開くか再読み込みしたときに第 2 のタブ (背景情報) がアクティブ, つまり表示されている状態になります.

グラフ　表

このセクションでは散布図を表示します.

```
par(mar = c(4, 4, .5, .1))
plot(mpg ~ hp, data = mtcars, pch = 19)
```



グラフ　表

このタブではデータを表示します.

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

図 7.4: 複数のセクションをタブに

```
## 結果 {.tabset}

### 内容

ここに文章.

### 背景情報 {.active}

ここに文章.
```

タブセットを終わらせるには, 上位レベルのセクション見出しを新しく開始する必要があります. 新しいセクションの見出しは空にします. 例えばこのように.

```
## Results {.tabset}

### Tab One

### Tab Two

## {-}

上記のように番号なし（`{-}`）で空のセクション見出しがあれば,
タブセットを終了しさらなる段落を続けることができます.
```

7.7 Rmd ソースファイルを HTML に埋め込む

他の人と HTML 出力ページを共有するとき, Rmd ソースファイルもほしいかもしれません. 例えば 自分自身で Rmd ソースを変更し, レポートをコンパイルしたいかもしれません. オプション `code_download` を使えば, Rmd ソースファイルのコピーを HTML に埋め込むことができます.

```
output:
  html_document:
```

```
    code_download: true
```

オプションを有効にすると, HTML 出力ページにはダウンロードボタンが現れ, 読者はそのボタンを押してソースファイルのダウンロードができます.

7.8 HTML 出力に好きなファイルを埋め込む

7.7 節で言及したように, HTML 出力には Rmd ソース文書のコピーを埋め込めます. Rmd ファイル単体ではレポートを再現するのに不十分なこともあります. 例えばレポートに外部のデータファイルが必要かもしれません. HTML 出力ファイルに好きなファイルを埋め込んでくれる一連の関数が **xfun** パッケージ (Xie, 2021g) にあります. これらの関数を使うために, 以下の R パッケージを用意しておきます.

```
xfun::pkg_load2(c("htmltools", "mime"))
```

これにより, コードチャンク内で `xfun::embed_file()`, `xfun::embed_files()`, `xfun::embed_dir()` のどれかを使って, 1 つ以上のファイルやディレクトリを HTML 出力に埋め込めます. 例えばこのように.

````
```{r echo=FALSE}
# a single file
xfun::embed_file('source.Rmd')

# multiple files
xfun::embed_files(c('source.Rmd', 'data.csv'))

# a directory
xfun::embed_dir('data/', text = 'Ðownload full data')
```
````

プログラミング的にファイルのリストを与えることもできます. 例えばこのように.

```
# embed all Rmd and csv files
xfun::embed_files(list.files(".", "[.](Rmd|csv)$"))
```

複数のファイルに対しては, これらの関数はまず zip ファイルに圧縮してから, その zip ファイルを埋め込みます. これらの関数はリンクを返し, 読者は HTML ページのリンクをクリックして埋め込んだファイルをダウンロードすることができます.

これらの関数のより詳細な技術的情報は, ヘルプページ `?xfun::embed_file` またはブログ投稿 https://yihui.org/en/2018/07/embed-file/ で学ぶことができます. 同様のアイディアにより, **downloadthis** package (Mattioni Maturana, 2020) はダウンロードボタンを実装したことで, ユーザーはリンクではなくダウンロードボタンをクリックしてダウンロードできるようになります. ボタンを使うほうがお好みなら, こちらを使うことも検討するとよいでしょう.

7.9 カスタム HTML テンプレートを使う (*)

既に 6.10 節で LaTeX テンプレートについて話しました. カスタム HTML テンプレート を指定すると, Pandoc が Markdown を HTML へと変換できます. 以下は簡単なテンプレートの例です.

```
<html>
  <head>
    <title>$title$</title>
    $for(css)$
    <link rel="stylesheet" href="$css$" type="text/css" />
    $endfor$
  </head>
  <body>
  $body$
  </body>
</html>
```

テンプレートに `$title$`, `$body$` といったいくつかの変数が含まれているのがわかるでしょう. Pandoc 変数の完全なリストとそれぞれの意味については https://pandoc.org/MANUAL.html#templates で検索できます.

テンプレートによってあなたは HTML 出力をカスタマイズする究極の力を得ることになります.

例えば好きな CSS スタイルシートや JavaScript コード, あるいはライブラリを <head> 内で読み込ませたりできます. あるいは次のように, 文書が下書きか最終稿かを示すブーリアン変数 draft も使えます.

```
<head>
<style type="text/css">
.logo {
  float: right;
}
</style>
</head>

<body>
<div class="logo">
$if(draft)$
<!-- use draft.png to show that this is a draft -->
<img src="images/draft.png" alt="Draft mode" />
$else$
<!-- insert the formal logo if this is final -->
<img src="images/logo.png" alt="Final version" />
$endif$
</div>

$body$
</body>
```

すると Rmd 文書の YAML メタデータ内で draft 変数に true または false を設定できます. 例えばこのように.

```
---
title: "An Important Report"
draft: true
---
```

テンプレートを Rmd 文書に適用するためには, テンプレートをファイルに保存した上で, `html_document` の `template` オプションにファイルパスを与えます. 例えばこのように.

```
output:
  html_document:
    template: my-template.html
```

rmarkdown パッケージに同梱したカスタム HTML テンプレートを読み込んで使用しており, これは Pandoc のデフォルトテンプレートとは異なります. Pandoc のデフォルトを使うには `template: null` で指定できます.

7.10 既存の HTML ファイルの内容を読み込む (*)

`html_document` フォーマット (あるいはこのオプションをサポートしている他のフォーマット) の `includes` オプション があれば, 既存の HTML ファイルの本文を 出力する HTML 文書内の 3 箇所のいずれかに読み込むことができます. それらは

タグ内部,

の前, そして本文の後でかつ

の直前です.

```
output:
  html_document:
    includes:
      in_header: header.html
      before_body: before.html
      after_body: after.html
```

HTML にあまり詳しくないなら, 7.9 節がこのオプションをより理解するのに役に立つかもしれません.

`in_header` オプションを使うなら, CSS と JavaScript コードを `<head>` タグ内に挿入することができます. `before_body` を使うなら, バナーやロゴを表示するヘッダを埋め込むこともできます. `after_body` を使うなら, フッタを読み込ませることができます. 例えばこのように.

```
<div class="footer">Copyright &copy; John Doe 2020</div>
```

外部 HTML ファイルの内容を本文の好きな位置に読み込みたいときもあるでしょう．これは `htmltools::includeHTML()` を使えば可能です．HTML ファイルパスをこの関数に与えます．関数はこのファイルを読み込み, 出力文書にたいしてこのファイルの中身を書き込みます．9.5 節で使用したようなテクニックをを使っても良いかもしれません．例えばこのように．

````
```{=html}
```{r, echo=FALSE, results='asis'}
xfun::file_string('file.html')
```
```
````

HTML ファイル内に読み込めるのは別の HTML の一部分だけであり, HTML ファイルそのものを読み込んではならないことに注意してください．完全な HTML ファイルとは `<html>` タグを含むものであり, これは他の `<html>` タグ内に埋め込むことができません．以下は HTML 文書に別の完全な HTML 文書が 埋め込まれた場合の無効な HTML 文書です．

```
<html>
  <head>  </head>

  <body>
  親 HTML ファイル.

  <!-- 以下 htmltools::includeHTML() -->
    <html>
      <head>  </head>
      <body>
      子 HTML ファイル.
      </body>
    </html>
  <!-- 読み込み終わり -->
```

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

HTML ファイルを別の HTML 出力文書に読み込む時に問題が発生したなら, HTML ファイルに `<html>` タグが含まれていないか確認するとよいでしょう.

rmarkdown パッケージには `html_fragment` という出力フォーマットがあり, 完全な HTML 文書の代わりに HTML の一部を生成します. Rmd 文書内で別のコンパイルされた Rmd 文書の結果を読み込みたい場合, 後者の Rmd 文書の出力には 通常用いる `html_document` フォーマットの代わりに `html_fragment` フォーマットを使うとよいでしょう.

HTML ファイルの代わりに Rmd または Markdown 文書を読み込ませたいなら, 16.4 節で紹介されている子文書を使うこともできます.

7.11 ブラウザアイコンをカスタマイズする

7.10 節では `html_document` フォーマットの `includes` オプションを使って, 追加のコードを HTML のヘッダ, 本文, フッタに挿入できることを実演しました. このテクニックはファビコンというカスタムブラウザアイコンを HTML 出力に追加することにも使えます.

ファビコンはブラウザのアドレスバー, タブタイトル, ブックマークに表示されるウェブサイトのロゴです. 例えば Google Chrome で CRAN のウェブサイト (`https://cran.r-project.org`) を開いてブラウザのタブを見ると, 小さな R ロゴがあります. 携帯端末ならばファビコンはウェブサイトをホームスクリーンに固定表示した際にアプリアイコンの代わりに使われます.

HTML 文書にファビコンを追加するには, 以下のコードをカスタムヘッダファイル (7.10 節で言及したように, `header.html` といった名前のファイル) を追加します.

```
<link rel="shortcut icon" href="{ファビコン画像ファイルへのパス}" />
```

YAML メタデータを使って, このファイルを文書の `<head>` 領域に挿入できることを思い出してください.

```
output:
  html_document:
```

```
  includes:
    in_header: header.html
```

<link> の href 属性に与えるパスは, 他のアセット (例えば画像やデータ・セット) を参照するときと同じように, 相対パス構造を前提とすべきです. 使用する画像は, 最も小さい正方形の PNG ファイルがよく機能します. 典型的なウェブブラウザは画像を 16 x 16 ピクセルの領域に表示するため, シンプルなデザインがより良いということに留意してください.

あなたの文書を表示する各種のブラウザやプラットフォームが, 特定のレイアウトに対して最適な解像度のバージョンのアイコンを用いるようにしたいのであれば, https://realfavicongenerator.net といったサービスを使って, ファビコンセットとやや複雑な HTML ヘッダのコードを生成するとよいでしょう. このサービスは現在 **pkgdown** パッケージ (Wickham and Hesselberth, 2020) の pkgdown::build_favicon() 関数で R パッケージロゴからファビコンセットを作り出すのに使用されています.

7.12 折りたたみ要素 <details> を使う

7.4 節で言及したように, html_document フォーマットの code_folding: hide オプションでソースコードチャンクを折りたたむことができます. 現在は出力ブロックを折りたたむことはできませんが, JavaScript の小ワザが使えれば出力を折りたたみできます. これは出力が比較的長く, しかしさほど重要でないときに役に立つでしょう. 初期状態で折りたたみ, 読者が興味を持てば内容を見るために展開することができます. 図 7.5 はその例です.「詳細」ボタンをクリックして出力を展開できるでしょう.

あなたがご覧になっているのが本書の HTML バージョンなら, 以下のチャンクで実際に動くのを見ることができます. PDF または印刷版を読んでいるのなら, このような対話的機能 (「詳細」ボタンを押すこと) はもちろん不可能です.

```
01 1:100
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
## [13] 13 14 15 16 17 18 19 20 21 22 23 24
## [25] 25 26 27 28 29 30 31 32 33 34 35 36
## [37] 37 38 39 40 41 42 43 44 45 46 47 48
## [49] 49 50 51 52 53 54 55 56 57 58 59 60
```

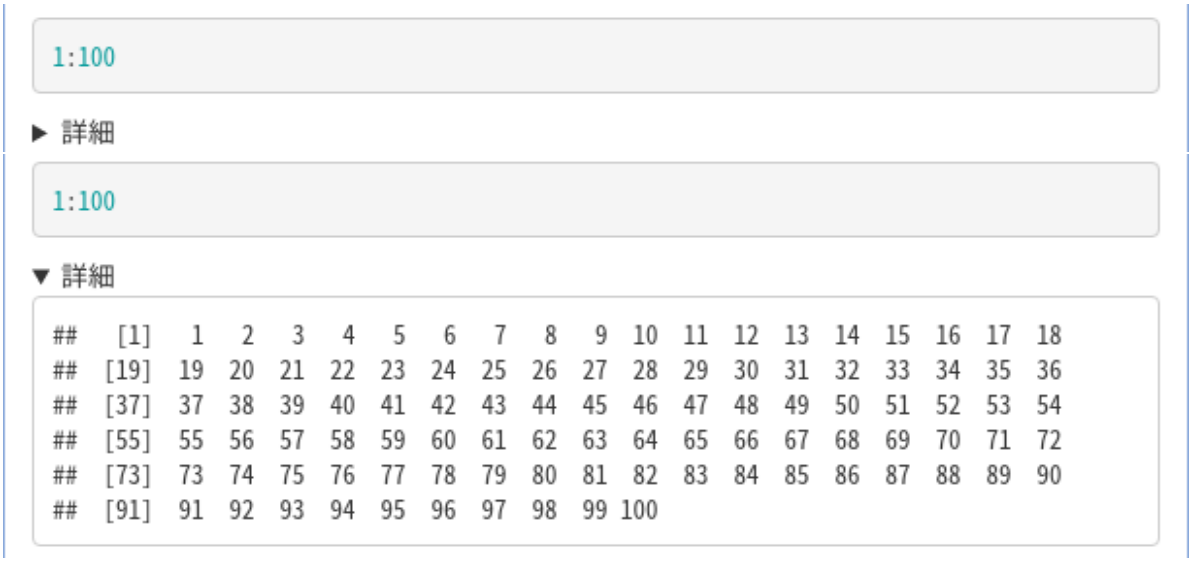


図 7.5: details 要素でテキスト出力を囲む

```
## [61] 61 62 63 64 65 66 67 68 69 70 71 72
## [73] 73 74 75 76 77 78 79 80 81 82 83 84
## [85] 85 86 87 88 89 90 91 92 93 94 95 96
## [97] 97 98 99 100
```

以下の Rmd 文書は, 出力ブロックを検出しそれを `<details>` タグで囲む JavaScript コードを読み込ませた, 完全なソースです.

````
---
title: 折りたたみ要素 `<details>` を使う
output: html_document
---

この例ではテキスト出力を `<details>` タグ内に表示します.
JavaScript でテキスト出力ブロックを `<details></details>` で囲みます.
Javascript コードはこの文書の末尾で実行する必要があるため,
最後に配置します. 以下はテスト用のコードチャンクです.

```{r}
1:100
```
````

実際の JavaScript コードは以下になります.

```{js, echo=FALSE}
(function() {
  var codes = document.querySelectorAll('pre:not([class])');
  var code, i, d, s, p;
  for (i = 0; i < codes.length; i++) {
    code = codes[i];
    p = code.parentNode;
    d = document.createElement('details');
    s = document.createElement('summary');
    s.innerText = '詳細';
    // <details><summary>詳細</summary></details>
    d.appendChild(s);
    // コードを <details> 内に移動
    p.replaceChild(d, code);
    d.appendChild(code);
  }
})();
```

上記の JavaScript コードを自分で適用することもできます. ポイントは要素を <details> で囲むということです.

```
document.querySelectorAll('pre:not([class])');
```

CSS セレクタの pre:not([class]) は class 属性のない全ての <pre> 要素を意味します. 他の要素のタイプを選択することもできます. CSS セレクタについてより知りたいなら, https://www.w3schools.com/css/css_selectors.asp を見てください. HTML タグ <details> と <summary> をより知りたいなら, https://www.w3schools.com/tags/tag_details.asp を見てください.

7.13 HTML 出力を Web で共有する

R Markdown を HTML ファイルにレンダリングする魅力的な側面の 1 つは, これらのファイルをとても簡単にインターネットでホストし, 他のウェブサイトと同様に共有できるということです. この節ではあなたの作成した HTML 文書を共有するオプションを簡単に要約します.

7.13.1 R 特化のサービス

RStudio は R Markdown で作られた様々な種類のコンテンツをインターネットで公開するためのサービスをいくつか提案しています. これらのサービスは特に RStudio IDE または **rsconnect** パッケージ (McPherson and JJ Allaire, 2021) を使って簡単に公開できます.

- **RPubs**[*4] は静的な単一の R Markdown コンテンツの無料ホスティングを可能とします. RStudio IDE の `Publish` ボタンまたは `rsconnect::rpubsUpload()` 関数で簡単に公開できます. 詳細は “Getting Started” のページ (https://rpubs.com/about/getting-started) をご覧ください.
- **ShinyApps.io**[*5] は R を実行するサーバを要する動的コンテンツのホスティングを可能にします. 例えば Shiny コンポネントを含んでいる[*6]インタラクティブな R Markdown 文書をホストできます. ShinyApp.io は Shiny アプリケーション用の RPubs の類似サービスです. アプリとインタラクティブな R Markdown 文書は RStudio IDE の push ボタンか `rsconnect::deployApp()` 関数を使って公開できます. 詳細はユーザーガイド (https://docs.rstudio.com/shinyapps.io/) をご覧ください.
- **bookdown.org**[*7] は **bookdown** パッケージで書かれた本の無料ホスティングを提案します. `bookdown::publish_book()` 関数であなたの書籍の静的な出力ファイルを簡単に公開できるでしょう.
- **RStudio Connect**[*8] は組織団体が自前のサーバで動作させるような企業向け製品です. 作成された広範な種類のコンテンツ (R Markdown 文書, Shiny アプリケーション, API など) を文書レベルでのアクセス制御, 閲覧履歴などといった機能を使いセキュアな環境でホ

[*4] https://rpubs.com

[*5] https://www.shinyapps.io

[*6] R Markdown 文書に Shiny コンポネントを含むには, YAML メタデータで `runtime: shiny` または `runtime: shiny_prerendered` オプションを設定することもできます. この文書を以前のように HTML 文書にレンダリングすることはできないでしょうが, 代わりに `rmarkdown::run()` で文書を実行することになります. 詳しく知るには Xie J.J. Allaire, and Grolemund (2018) (Chapter 19, https://bookdown.org/yihui/rmarkdown/shiny-documents.html) をご覧ください.

[*7] https://bookdown.org/home/about/

[*8] https://rstudio.com/products/connect/

ストできます. コンテンツは RStudio Connect に手動でアップロードするか, **rsconnect** パッケージか, または git ベースのデプロイによって公開できます.

7.13.2 Static website services

端的に言うなら, 単純な静的ウェブサイトは数個の HTML ファイル (典型的にはホームページである `index.html`), JavaScript, CSS ファイル, そして画像などの追加のコンテンツで構成されます. 一連のファイルは web サーバにそのままホストし, web ブラウザに表示させることができます.

R Markdown が HTML 出力フォーマットでレンダリングされた場合, その結果は静的なウェブサイトとして扱われます. ウェブサイトは単一のスタンドアロンな HTML ファイル (デフォルトオプション `self_contained: true` を使った場合に得られます) から, ファイルのセット, **blogdown** パッケージ (静的なウェブサイトジェネレータに依存しています) に基づいたウェブサイトのような洗練されたプロジェクトまで複雑さの点で多岐に及びます. より詳しく知りたいなら, **blogdown** 本 (Xie Hill, and Thomas, 2017) の Section 2.1 on Static Sites[*9] を見てください.

結論として, あなたは R 特化のサービスに加え, 多くの無料で使用可能な静的ウェブサイトホスティングサービスを使って HTML 文書をホストできるでしょう. R コミュニティでのよくある選択としては以下があります.

- **GitHub Pages**[*10] は Github リポジトリから Markdown と HTML コンテンツをそのまま公開する場合は特に簡単です. main ブランチのルートか, main ブランチの `docs/` ディレクトリ, あるいは特定の `gh-pages` ブランチからコンテンツをホストすることを指定することになるでしょう. 新しいコンテンツの公開は git でリポジトリに新しい HTML ファイルをプッシュするだけで可能です.
- **GitLab Pages**[*11] は GitHub Pages と類似の機能を GitLab リポジトリに対して提案します. GitLab はリポジトリの `public/` ディレクトリに保存されたコンテンツをデプロイします. コンテンツをビルドし公開するには, `.gitlab-ci.yml` という YAML ファイルで指示を与えなければなりませんが, GitLab は多くの便利なテンプレートを提供してくれます. レンダリングされた HTML コンテンツをホストする例として, https://gitlab.com/pages/plain-html/-/tree/master をご覧ください.
- **Netlify**[*12] は静的な web コンテンツをビルドしデプロイするプラットフォームです. **blogdown** と **pkgdown** で作成された web コンテンツに対する選択としてはよく知られていますが, これはあらゆる種類の HTML ファイルをホスティングできます. 公開方法と

[*9] https://bookdown.org/yihui/blogdown/static-sites.html
[*10] https://pages.github.com
[*11] https://docs.gitlab.com/ce/user/project/pages/
[*12] https://www.netlify.com

して, ドラッグ・アンド・ドロップ, コマンドライン, あるいは GitHub および GitLab レポジトリから自動公開するといったいくつもの選択があります. 加えて, プルリクエストから web サイトをプレビューするといった多くの役立つ機能も提案されています. 詳細は Netlify のドキュメント (https://docs.netlify.com) や RStudio webinar “Sharing on Short Notice”[*13] をご覧ください.

7.14 HTML ページのアクセシビリティを向上させる

HTML 出力文書に, 何らかの視覚的な障害を持つ読者に対するアクセシビリティをもたせることは重要です. たいていの場合, こういった読者は文書を視覚的に読む代わりに, スクリーンリーダ (音声読み上げソフト) といった**聞く**ための特殊なツールを使います. 通常はスクリーンリーダはテキストを読み上げることができるだけで, (ラスタ) 画像を読み上げられません. つまりあなたはスクリーンリーダに十分なテキストでヒントを与える必要があるということです. 良いニュースは, わずかな労力があればあなたの文書のアクセシビリティを格段に向上できるということです. Jonathan Godfrey が R Markdown 文書のアクセシビリティのためのいくつかのヒントを https://r-resources.massey.ac.nz/rmarkdown/ で記事にしています.[*14] この記事に基づいて, 本書の読者にとっての利便性になるいくつかのヒントを以下に提示します.

- HTML 文書はしばしば PDF よりアクセシビリティが優れている.
- 可能ならば HTML 出力文書に Rmd ソース文書を同梱するようにする (例えば 7.7 節でその方法の 1 つを実演しています). HTML 文書にアクセシビリティがない場合, 視覚障害者は Rmd ソースから内容を理解できるかもしれませんし, ソースを修正することもできるかもしれません.
- グラフのタグにテキストで情報を含ませる. 2014 年の useR! カンファレンスで, 私は Jonathan からこの問題を個人的に教えてもらいました. web ページ上の画像の `alt` 属性の大切さを, 私は初めて理解しました.

  この問題を理解するために, まずは web ページの画像が HTML タグ `<img />` によって生成されることを知らなければなりません. このタグは `src` 属性を持ち, 画像ファイルのソースの場所を指定しています. 例えば `<img src="foo_figures/image.png" />` のように. 視力のある読者はこの画像を見ることができますが, 視覚障害者には描かれていることを知るのは難しいのです. それはスクリーンリーダは, 特にラスタ画像の場合, 画像を読むことが出来ないためです (SVG のようなベクタ画像は多少ましかもしれませんが). この場合テキスト

[*13] https://rstudio.com/resources/webinars/sharing-on-short-notice-how-to-get-your-materials-online-with-r-markdown/

[*14] JooYoung Seo も, 視覚障害を持つ人の手助けになる R パッケージを https://jooyoungseo.com/post/ds4blind/ で紹介しています. これは R Markdown と直接関係しませんが, 視覚障害者がどうグラフを読み取っているのかを学ぶのに役に立つでしょう.

でのヒントを与えると, スクリーンリーダは読み上げることができるので便利です. このテキストでのヒントは画像の代替 (alternate) テキストを意味する `alt` 属性で与えることできます.
R Markdown のコードチャンクから生成された画像の場合は, もしチャンクオプション `fig.cap` (つまり 画像のキャプション, figure caption) が設定されているなら `alt` 属性はここから生成されます. 代わりに Markdown 構文 `![]()` を使って画像を挿入することもできます. 画像パスをパーレン `()` 内に入力し, `alt` テキストをブラケット `[]` 内に入力, 例えば `![`**テキスト情報**`](`**パス** `/image.png)` のように.
`alt` テキストは視覚のある読者にとっては HTML ページ上に表示されません. しかし画像にキャプションや代替テキストを与えた場合, `rmarkdown::html_document` フォーマットはデフォルトでキャプション要素を表示します. もし実際にキャプションを表示させたくないなら, このように `fig_caption` をオフにすることができます.

```
output:
  html_document:
    fig_caption: false
```

このケースでは `alt` 属性はまだ生成されますが, 表示されることはありません.

- 数学的なコンテンツを書くには画像の代わりに LaTeX 構文を使う (例: `$ $`, あるいは `$$ $$`). デフォルトでは, R Markdown は数学的なコンテンツのレンダリングに MathJax ライブラリを使い, 結果としてスクリーンリーダが読み上げられるものになります.
- チャンクオプション `comment = ""` を設定してコードチャンクのテキスト出力の行頭の `##` を除く (11.12 節参照).

我々はアクセシビリティの専門家ではありませんので, 詳細を知りたければ元の記事を読むことをお薦めします.

7.15 ハードコア HTML ユーザー向けの話 (*)

6.12 節では, Markdown がシンプルであるがゆえにその制約が強すぎると感じるときは, Markdown の代わりに純粋な LaTeX 文書にコードチャンクを埋め込むことができる, という話をしました. 同様に, 直接 HTML コードを書くことに慣れていて快適さを感じるなら, HTML にコードチャンクを混ぜ合わせることもまた可能です. そのような文書は慣習的に `.Rhtml` というファイル拡張子を持ちます.

`Rhtml` 文書では, コードチャンクは `<!--begin.rcode` と `end.rcode-->` の間に埋め込まれ, インラ

イン R コードは <!--rinline --> 内に埋め込まれます．以下は Rhtml 全体の例です．これを test.Rhtml というファイル名で保存し，knitr::knit("test.Rhtml") を使ってコンパイルできます．出力は 1 つの HTML (.html) ファイルになります．RStudio では，ツールバーの Knit ボタンを押すことでもコンパイルできます．

```
<!ÐOCTYPE html>
<html>
<head>
  <title>HTML の最低限の knitr 使用例</title>
</head>
<body>
<!--begin.rcode
  knitr::opts_chunk$set(fig.width=5, fig.height=5)
  end.rcode-->

  <p>これは <strong>knitr</strong> が純粋な HTML ページでどのように動作するかの最低限の例
  ↪ です.</p>

  <p>お決まりの退屈な使用例.</p>

<!--begin.rcode
    # 単純な計算
    1 + 1
    # 退屈な乱数生成
    set.seed(123)
    rnorm(5)
    end.rcode-->

  <p>図を生成することもできます (<code>fig.align='center'</code>オプションで中央揃えして
  ↪ います).</p>

<!--begin.rcode cars-scatter, fig.align='center'
    plot(mpg ~ hp, data = mtcars)
    end.rcode-->
```

```
  <p>エラー，メッセージ，警告文はそれぞれ異なる<code>class</code>を持つ <code>div</code>
  ↪ 内に配置されます．</p>

<!--begin.rcode
    sqrt(-1)  # 警告
    message('knitr says hello to HTML!')
    1 + 'a'  # ミッションインポッシブル
    end.rcode-->

  <p>さて全てうまくいっているようです．R に &pi; の値を聞いてみましょう．もちろん答えは
  ↪ <!--rinline pi --> です．</p>

</body>
</html>
```

第8章

Word

R Markdown から Word 文書を生成するには, 出力フォーマット `word_document` が使えます. 文書に相互参照を含めたいなら, 4.7 節でも言及された `bookdown::word_document2` を検討するとよいでしょう.

```
---
output:
  word_document: default
  bookdown::word_document2: default  # 相互参照のため
---
```

われわれの経験上, Word 出力に関する最もよくある質問は以下のようなものです.

1. 文書へのカスタム Word テンプレートはどうすれば適用できるのか?
2. Word 側で元の R Markdown 文書を適切に変更にはどうすればいいのか?
3. 文書の要素の個別のスタイルの設定はどうすればいいのか?

この章ではこれらの質問に答えていきます.

8.1 カスタム Word テンプレート

Word テンプレート文書で定義されたスタイルを R Markdown で新たに生成された Word 文書に適用することができます. テンプレート文書は「スタイル参照文書」“style reference document” とも呼ばれています. ポイントは, まずはこのテンプレート文書を Pandoc から作成し, それから

スタイル定義を変更しなければならないということです. それからこのテンプレートファイルのパスを word_document の reference_docx オプションに与えてください. 例えばこのように.

```
---
output:
  word_document:
    reference_docx: "template.docx"
---
```

たった今言及したように, template.docx は Pandoc から生成されたものでなければなりません. このテンプレートは word_document 出力フォーマットを使ったどのような R Markdown 文書からでも (この文書の実際の内容はなんでも問題ありませんが, スタイルを適用したい要素の種類を含んでいるべきです) 作ることができます[*1]. それから .docx ファイルを開き, スタイルを編集します.

図 8.1 は Word の「ホーム」タブから「スタイル」ウィンドウを開くと見つけられます. カーソルを文書の特定の要素上に動かすと, スタイルリストの項目が強調されます. ある要素のスタイルを変更したいならば, 強調された項目上でドロップダウンメニューの「変更」をクリックして図 8.2 のようなダイアログボックスを見ることができます.

スタイルの編集を終わったら, 文書を保存し (誤って上書きしないようなファイル名にしてください), 今後の Word 文書のテンプレートとして使用することができます. Pandoc が参照文書テンプレートを与えられて新しい Word 文書をレンダリングするとき, テンプレートのスタイルが読み出されて新しい文書に適用されます.

カスタムスタイル付きの Word テンプレートを作成する方法の詳細については, https://vimeo.com/110804387 で短い動画を見るか, https://rmarkdown.rstudio.com/articles_docx.html の記事を読むとよいでしょう.

文書の要素に対するスタイル名がすぐには見つからないこともあります. 複数のスタイルが同じ要素に適用され, それらのうち 1 つだけが強調されてみえることもあります. 修正したいスタイルが実際になんであるかは, 当て推量やネット検索で解決することが求められることもあります. 例えば「スタイルの管理」ボタン (図 8.1 のスタイルリストの下部にある, 左から 3 番目のボタン) をク

[*1] **訳注**: この記述はややわかりにくいかもしれませんが, R Markdown もまた Pandoc によってファイルを生成していることを思い出してください. 具体的な手順は次のようになります. 一旦 (1) R Markdown 文書をコンパイルし docx ファイルを生成し, (2) このファイルを Word で開いて変更したい箇所のスタイル設定を行い, 保存します. (3) そして上記のように reference_docx フィールドに保存したファイルを指定してから, 再びファイルをコンパイルします. このとき参照されるのはスタイル情報のみなので, 既に保存した文書の内容は出力に影響しません.

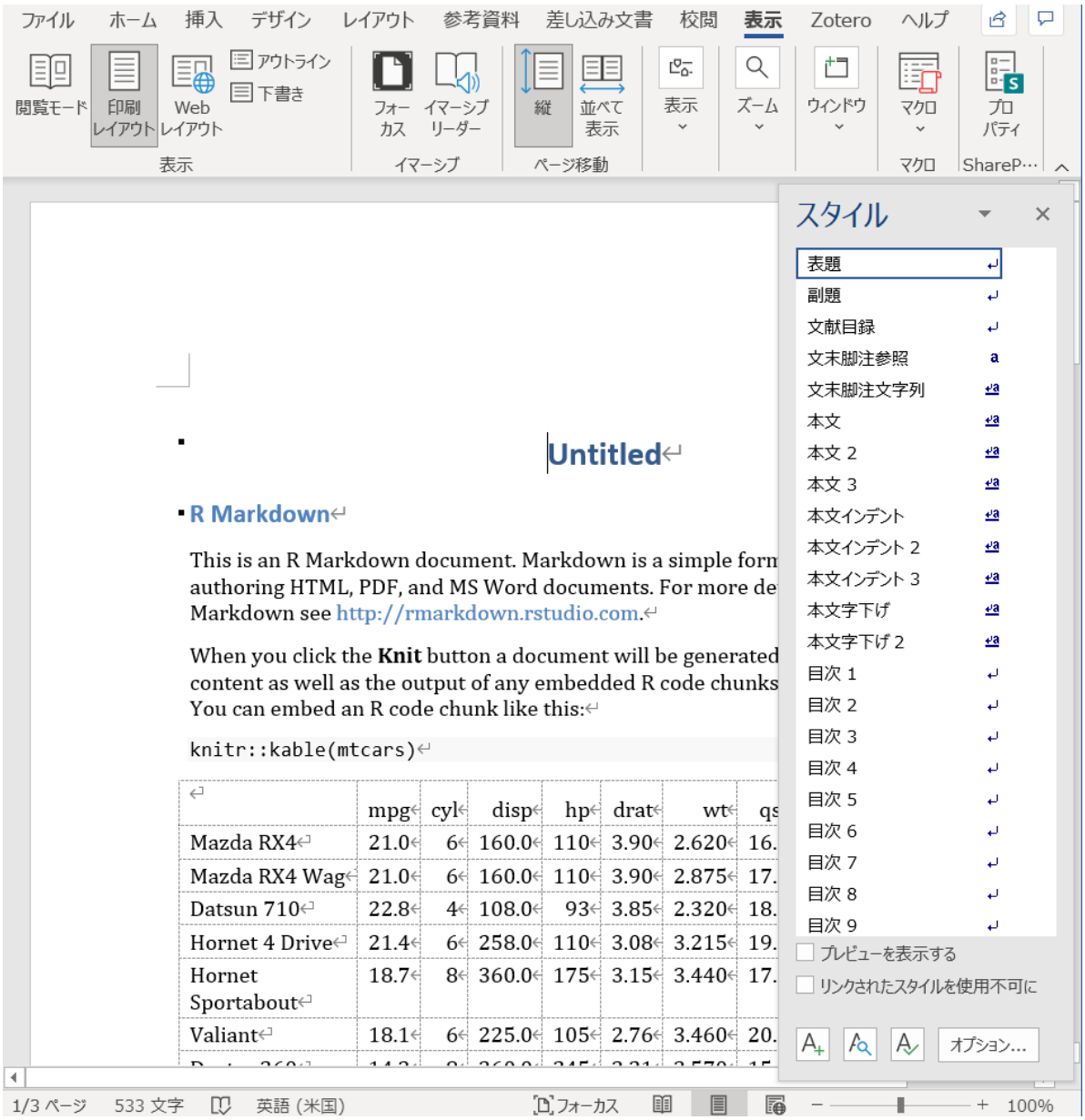


図 8.1: 特定の文書要素のスタイルを見つける

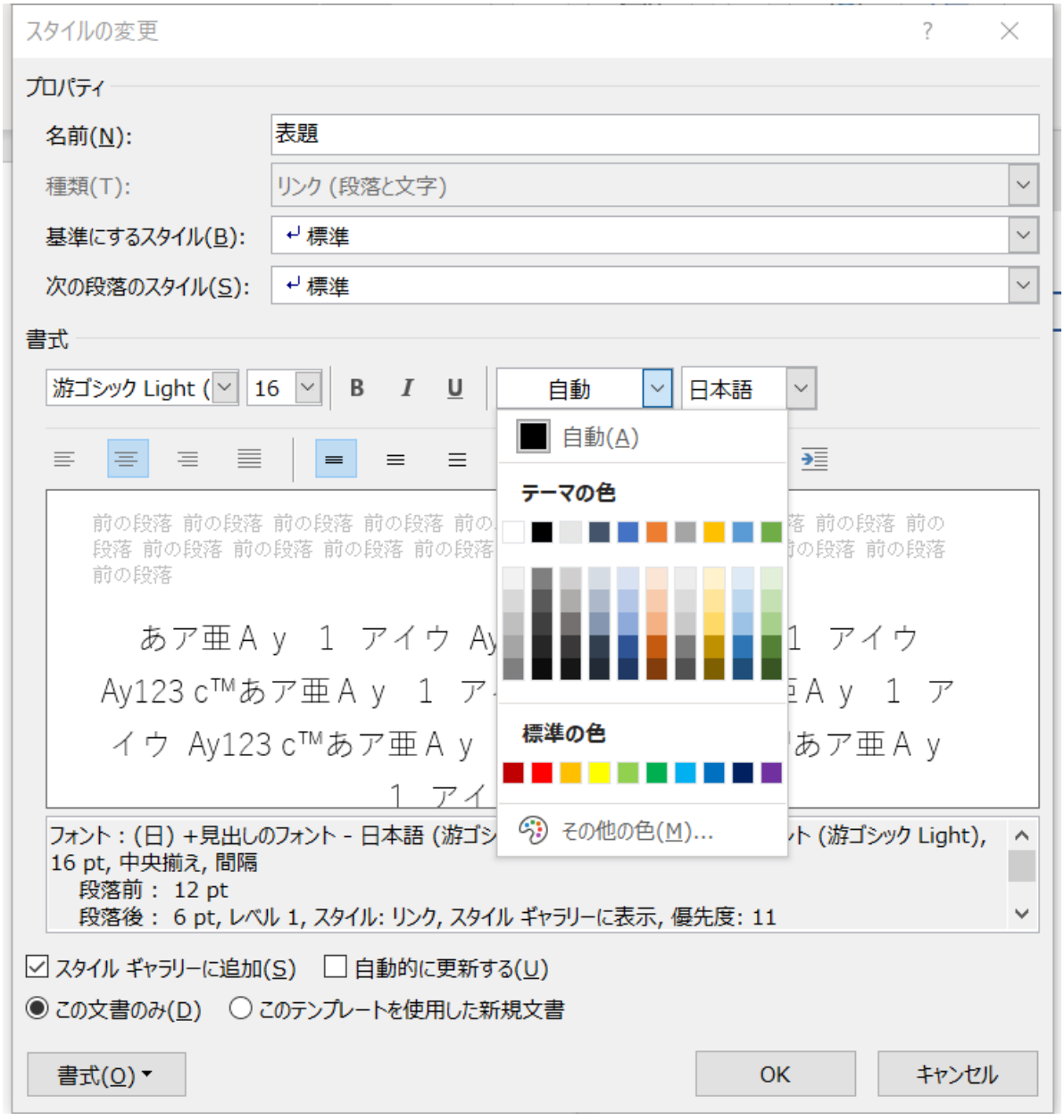


図 8.2: Word 文書の要素のスタイルを変更する

リックし, “Table” スタイル (図 8.3 参照) を見つけるまでには多数のスタイル名をスクロールして飛ばさなければならりません. これでようやく, 枠線などの表のスタイルを修正できます.

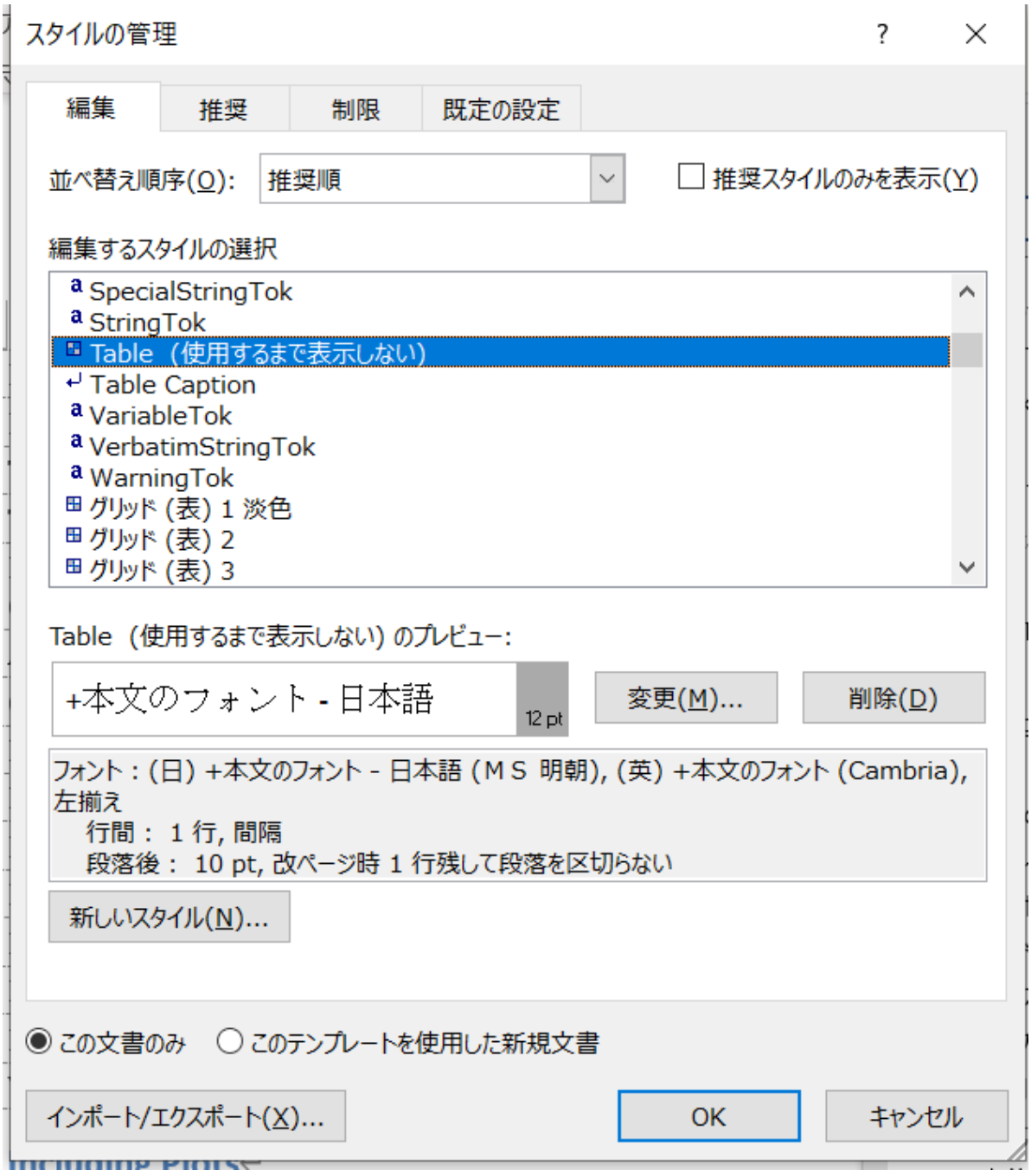


図 8.3: Word 文書の表のスタイルを修正する

訳注

Word の日本語版へのローカライズの過程によるものか, バージョンアップによるものかは分かりませんが, Word のスタイル名は必ずしもこの章の例とは一致しないかもしれません.

8.2 R Markdown と Word 間の双方向ワークフロー

R Markdown から Word 文書を生成するのは簡単ですが, その一方, 誰かが Word 文書を編集したものを元の R Markdown 文書に手動で反映しなければならないときは, 特に苦痛が伴います. 幸いにも Noam Ross がこの問題に対して有望な解決策を提示しています. **redoc** パッケージ (https://github.com/noamross/redoc) は Word 文書を生成し, 文書を校正してから R Markdown に再度変換することを可能にします. この原稿を書いている現時点 (2020 年 6 月) では **redoc** パッケージはまだ試験的であり, さらに不運なことに, 彼は開発を中止しています. どちらにせよこれを試して見たいなら, GitHub からインストールすることができます.

```
remotes::install_github("noamross/redoc")
```

パッケージがインストールされたら, 出力フォーマット `redoc::redoc` を使うこともできます.

```
---
output: redoc::redoc
---
```

この出力フォーマットは, 実際には元の Rmd 文書を保存した Word 文書を生成するので, Word 文書を Rmd に変換して戻すこともできます. Word 上の変更履歴は CriticMarkup 構文 (http://criticmarkup.com) で書かれたテキストへ変換されます 例えば `{++ 重要 ++}` はテキストに「重要」という単語が挿入されたことを表現しています.

`redoc::redoc` で生成された Word 文書は, `redoc::dedoc()` 関数で Rmd に変換できます. 例えば `redoc::dedoc("file.docx")` は `file.Rmd` を生成します. この処理では Word 上の変更履歴にどう対処するかを `track_changes` 引数で決めることができます. 例えば変更を受け入れるか破棄するか, 変更履歴を CriticMarkup に変換するかなどです. 変更履歴が完全に失われてしまわないように, `track_changes = 'criticmarkup'` を使うことを推奨します.

Word 文書を編集する際には, R Markdown のコードチャンクやインライン R コードによって自動生成されて**いない**箇所を編集すると想定されています. 例えばコードチャンク内で `knitr::kable()` を使って自動生成された表は編集してはなりません. そのような変更は `dedoc()` で Word から Rmd に変換する際に失われるからです. コードチャンクで自動生成された出力を誤って編集することを避けるために, `redoc::redoc` フォーマットの `highlight_outputs` オプションを `true` に設定してください. これは自動出力した箇所を Word 上で背景色で強調表示することを意味します. あなたの共同編集者には Word 文書上の強調表示された箇所に触れないよう伝えるべきでしょう.

繰り返しになりますが, **redoc** パッケージは未だ試験的であり現時点ではその機能がはっきりしないため, ここでの導入はあえて簡潔なものとしています. 信用できない場合, GitHub 上のパッケージのドキュメントを読むことをお薦めします[*2].

8.3 個別の要素にスタイルを設定する

Markdown のシンプルさにより, Word 文書全体に対してグローバルなスタイル設定ができます (8.1 節参照) が, ある単語に色を着けたりある段落の中央揃えなど, 個別の要素に直接スタイルを設定することはできません.

David Gohel は, R 上で Office 文書で作業するのをより簡単にするという努力を続ける中で, 2018 年に **officedown** パッケージ (Gohel and Ross, 2021) の開発を始めました. これは **officer** (Gohel, 2021b) パッケージの機能のいくつかを R Markdown に持ち込むのが目的です. 本書の執筆時点ではこのパッケージの初期のバージョンが CRAN で公開されていますが, まだ実験的です. CRAN あるいは GitHub どちらからでもインストールできます.

```
# CRAN からインストール
install.packages("officedown")

# GitHub からインストール
remotes::install_github("davidgohel/officedown")
```

パッケージがインストールされたら, R Markdown 文書内で読み込む必要があります. 例えばこのように.

[*2] **訳注**: 2021/5/7 現在, **redoc** パッケージは開発を中止しています. 動作確認は使用者自身でお願いします.

````
```{r, setup, include=FALSE}
library(officedown)
```
````

officedown パッケージには `rdocx_document` という出力フォーマットがあります．これはデフォルトでは `rmarkdown::word_document` を基礎にして，表やグラフにスタイル付けする機能が追加されています．

officedown パッケージは **officer** パッケージを介して特定の Word 要素にスタイルを適用することを可能にします．例えば `officer::fp_text()` 関数でスタイルを作成し，インライン R コードの `ftext()` でテキストの一部にそのスタイルを適用できます．

````
---
title: officedown でテキストにスタイルを適用する
output:
  officedown::rdocx_document: default
---

```{r}
library(officedown)
library(officer)
ft <- fp_text(color = 'red', bold = TRUE)
```

# テスト

**officedown** パッケージは `r ftext('すごい', ft)`!
````

officer の関数とは別に，**officedown** パッケージは特殊な HTML コメントを使って **officer** のタスクが実現できます．例えば `officer::block_pour_docx()` は外部の Word 文書を現在の文書にインポートするのに用いますが，代わりに R Markdown 上で HTML コメントを使うこともできます．

```
<!---BLOCK_POUR_DOCX{file: 'my-file.docx'}--->
```

これはインライン R コードで以下のように書くのと等価です.

```
`r block_pour_docx(file = 'my-file.docx')`
```

officedown と **officer** パッケージで他にできることとして, 以下のようなものがあります.

- 改ページの挿入
- 多段組みレイアウトの配置
- 段落設定の変更
- 目次の挿入
- あるセクションのページの向きを変える (縦向きか横向きか)

officedown についてもっと学ぶには, 公式ドキュメント https://davidgohel.github.io/officedown/ をご覧ください.

:::.infobox .memo data-latex="{memo}"}

訳注

翻訳者は **officedown** のテンプレートを日本語化した簡易的なパッケージを作成しています.

https://github.com/Gedevan-Aleksizde/wordja :::

第9章

複数の出力フォーマット

R Markdown の主な利点は 1 つのソースから複数種類の出力フォーマットを作成できることです. これは 1 つ以上の Rmd ファイルでも可能です. 例えば本書は R Markdown で書かれ, 2 種類のフォーマットでコンパイルされています. 印刷用の PDF と, オンライン版の HTML です.

コードチャンクの出力を全ての出力フォーマットに対応させるのは時には難題になることがあります. 例えばたった 1 つの CSS ルール (`img { border-radius: 50%; }`) で HTML 出力に対して円の画像を作成するのは非常に単純ですが, LaTeX 出力ではこれとそのまま同じようにはいきません. 典型的には Tikz グラフィックスに関係する問題になります.

ある出力要素がすべての出力フォーマットに対して動作するわけではありません. 例えば **gifski** パッケージ (Ooms, 2021a) (4.14 節参照) で GIF アニメを簡単に作ることができ, これは HTML 出力では完璧に動作しますが, LaTeX 出力に埋め込んだものは追加の GIF ファイルの処理と LaTeX パッケージなしでは不可能です.

この章では複数のフォーマットで動作する例を少しだけ提示します. ある機能が特定の出力フォーマットでのみ有効なら, その出力フォーマットに基づいて条件付きで有効・無効にする方法を提示します.

9.1 LaTeX か HTML か

LaTeX と HTML はどちらもよく使われるフォーマットです. `knitr::is_latex_output()` 関数は出力フォーマットが LaTeX かどうか (Pandoc 出力フォーマットの `latex` および `beamer`) を教えてくれます. 同様に `knitr::is_html_output` 関数は出力フォーマットが HTML かどうか教えてくれます. デフォルトでは Pandoc 出力フォーマットのうち `markdown`, `epub`, `html`, `html4`, `html5`, `revealjs`, `s5`, `slideous`, そして `slidy` が HTML 出力とみなされます. ある Pandoc 出力が HTML であると思えないなら, そのフォーマットを除外するために, 例えばこのように `excludes` 引数を使えます.

```
# markdown を HTML として扱わない
knitr::is_html_output(excludes = "markdown")
## [1] FALSE
```

ある出力要素が LaTeX または HTML のみで生成されるのなら, これらの関数を条件つきで生成できるように使えます. 例えば, PDF のページには大きすぎる表はフォントサイズを小さくした環境内に表を入れるとよいでしょうが, そういった LaTeX 環境は HTML 出力では機能しませんので, HTML 出力に含めるべきでありません (HTML 出力でフォントサイズを調整したいなら, CSS を使うこともできます). 以下はその例です.

````
---
title: tiny 環境内で表をレンダリングする
output:
  pdf_document:
    latex_engine: lualatex
  html_document: default
documentclass: ltjsarticle
---

```{r, setup, include=FALSE}
knitr::opts_chunk$set(echo = FALSE)
options(knitr.table.format = function() {
  if (knitr::is_latex_output()) 'latex' else 'pandoc'
})
```

LaTeX 環境の `tiny` は LaTeX 出力でのみ生成されます.

```{r, include=knitr::is_latex_output()}
knitr::asis_output('\n\n\\begin{tiny}')
```

```{r}
knitr::kable(mtcars)
````
````

```
```

```{r, include=knitr::is_latex_output()}
knitr::asis_output('\\end{tiny}\n\n')
```

比較のため，以下に通常のフォントサイズの表を配置します．

```{r}
knitr::kable(mtcars)
```

上記の例でのポイントはチャンクオプション `include = knitr::is_latex_output()` です．`\begin{tiny}` `\end{tiny}` 環境は出力フォーマットが LaTeX の場合のみ含まれます．この例の 2 つの表は出力が LaTeX でない場合は同じ見た目になるでしょう．

5.1 節では HTML と LaTeX 出力のテキストの色を変更する関数を使用しました．4.14 節では，アニメーションの例を提示しました．これにも今回の小ワザを使うことができます．HTML 出力に対してアニメーションを生成し，LaTeX 出力に対しては静止画を生成するコードチャンクはこのようになります．

```{r animation.hook=if (knitr::is_html_output()) 'gifski'}
for (i in 1:2) {
  pie(c(i %% 2, 6), col = c('red', 'yellow'), labels = NA)
}
```

これらの条件付けのための関数はどこでも使えます．他のチャンクオプションにも使えます (例えばチャンクの評価に条件を付けるため `eval` に使うなど) し，あるいはこの例のように R コード内にも使えます．

```{r, eval=knitr::is_html_output(), echo=FALSE}
cat('これは HTML 出力でのみ見えます')
```

```
```

```{r}
if (knitr::is_latex_output()) {
  knitr::asis_output('\n\n\\begin{tiny}')
}
```
```

## 9.2 HTML ウィジェットを表示する

HTML ウィジェット (https://htmlwidgets.org) はインタラクティブな JavaScript アプリケーションの典型で, HTML 出力でのみ動作します. HTML ウィジェットを含んだ Rmd 文書を, PDF や Word など HTML でないフォーマットへと knit すると, このようなエラーメッセージが返ってくるでしょう.

```
Error: Functions that produce HTML output found in document
targeting X output. Please change the output type of this
document to HTML. Alternatively, you can allow HTML output in
non-HTML formats by adding this option to the YAML front-matter
of your rmarkdown file:

  always_allow_html: yes

Note however that the HTML output will not be visible in
non-HTML formats.
```

上記のエラーメッセージに示された解決法よりも良い方法があるのですが, 追加のパッケージが絡んできます. R に **webshot** パッケージ (Chang, 2019) をインストールし, さらに PhantomJS をインストールしてください.

```
01 install.packages("webshot")
02 webshot::install_phantomjs()
```
```

それから HTML ウィジェットつきの Rmd 文書を非 HTML フォーマットで knit すれば, HTML ウィジェットは静的なスクリーンショットとして表示されます. スクリーンショットは **knitr** によって自動的に作られます. **bookdown** 本の Section 2.10[*1] に, スクリーンショットの詳しい操作方法が書かれています.

9.3 Web ページの埋め込み

webshot パッケージ (Chang, 2019) と PhantomJS をインストール (9.2 説参照) していれば, `knitr::include_url()` でどんな web ページも出力文書に埋め込むことができます. コードチャンク内でこの関数に URL を与えれば, 出力フォーマットが HTML ならば `<iframe>` (インラインフレーム) が生成され, 他のパッケージならばスクリーンショットが埋め込まれます. 例えば図 9.1 は, 本書のオンライン版を読んでいるなら私の個人サイトが, それ以外なら代わりに静的なスクリーンショットが現れているはずです.

```
01 knitr::include_url("https://yihui.org")
```

`out.width` や `fig.cap` といった図に関連するほとんどのチャンクオプションが `knitr::include_url()` でも機能します.

サーバ上で Shiny アプリを公開しているなら, `knitr::include_app()` を使えばこれを文書に含めることができます. これは `include_url()` と同じように動作します. **bookdown** 本 (Xie, 2016) の Section 2.11[*2] には `include_app()` と `include_url()` に関する詳細な話が書かれています.

9.4 複数の図を横並びに

`fig.show="hold"` と `out.width` option オプションを併用して複数の図を並べることができます. 以下の例では `out.width="48%"` を設定し, 出力は図 9.2 になります.

[*1] https://bookdown.org/yihui/bookdown/html-widgets.html
[*2] https://bookdown.org/yihui/bookdown/web-pages-and-shiny-apps.html

About Blog 关于 日志

I’m a software engineer working at RStudio, PBC. I earned my PhD from the Department of Statistics, Iowa State University. My thesis was Dynamic Graphics and Reporting for Statistics, advised by Di Cook and Heike Hofmann. I have developed a few R packages either seriously or for fun (or both), such as knitr, animation, bookdown, blogdown, pagedown, xaringan, and tinytex. I founded a Chinese website called “Capital of Statistics” in 2006, which has grown into a large online community on statistics. I initiated the Chinese R conference in 2008. I’m a big fan of GitHub, LyX and Pandoc. I hate IE. I fall asleep when I see beamer slides, and I yell at people who use `\textbf{}` to write `\title{}`. I know I cannot eat code, so I cook almost every day to stay away from my computer for two hours.

这是谢益辉的个人主页。2013 年底我从 Ames 村办大学统计系毕业，终于解决了人生前 30 年被问最多的问题：“你怎么还没毕业？”目前就职于 RStudio。我支持开源，喜欢折腾网站和代码，是一个高度自我驱动的人。打羽毛球爱勾对角，打乒乓球像太极，网球满场子捡球，篮球容易被撞飞，攀岩一次，腿软。宅，口重，嗜辣，屡教不改。智商中等偏下，对麻将和三国杀有不可逾越的认知障碍，实变函数课上曾被老师叫醒。略好读书，偶尔也在网上乱翻帖子，对诗词楹联比较感兴趣，目前比较中意的一联是：千秋邈矣独留我；百战归来再读书。最喜欢的一首词是：

> 深情似海，问相逢初度，是何年纪？依约而今还记取，不是前生夙世。放学花前，题诗石上，春水园亭里。逢君一笑，人间无此欢喜。
> 无奈苍狗看云，红羊数劫，惘惘休提起。客气渐多真气少，汩没心灵何已。千古声名，百年担负，事事违初意。心头阁住，儿时那种情味。

図 9.1: iframe または screenshot による Yihui’s のホームページ

```
```{r, figures-side, fig.show="hold", out.width="48%"}
par(mar = c(4, 4, .1, .1))
plot(cars)
plot(mpg ~ hp, data = mtcars, pch = 19)
```
```

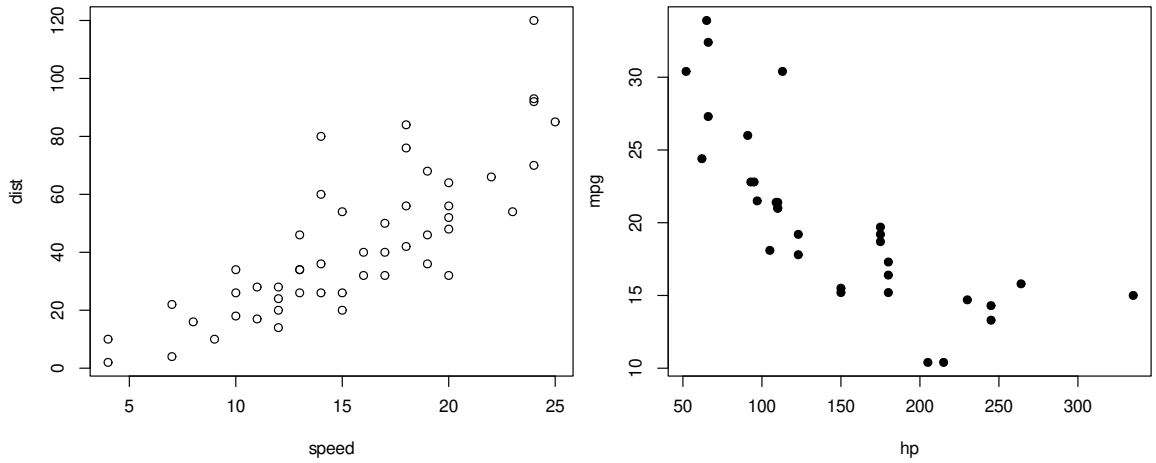


図 9.2: 横に並べた図

この単純なアプローチは PDF でも HTML 出力でも動作します.

訳注

この方法は, PDF では必ず横並びになるとは限りません. 余白にはみ出す大きさならば, 自動で折り返されます. これは LaTeX 側の文書スタイルの設定にも依存し, 多くの場合は欧文と和文でよく使われるレイアウトが異なることが原因です. よって, ここでは原著とは異なり画像サイズを 48% と少し小さくしています.

図の内部に複数のプロットがあり, サブ図を使いたいなら, 6.6 節を見てください. しかしサブ図は LaTeX 出力に対してのみサポートされているので気をつけてください.

9.5 生のコードを書く (*)

6.11 節で紹介したテクニックは実に広く使えます. いかに複雑な生のコードであっても Markdown 内で「生の」コンテンツとして保護するよう指定できます. 例えば HTML を直接書いたなら, `=html` 属性を使用することができます.

```
```{=html}
<p>どんな<strong>生の</strong> HTML コンテンツでもここでは動作します.
例えば, ここにユーチューブのビデオがあります.</p>

<iframe width="100%" height="400"
  src="https://www.youtube.com/embed/s3JldKoA0zw?rel=0"
  frameborder="0" allow="autoplay; encrypted-media"
  allowfullscreen></iframe>
```
```

属性名は Pandoc 出力の名前です. 出力フォーマット名を知りたいなら, Rmd 内で以下のコードチャンクの出力をみてください.

```
```{r}
knitr::pandoc_to()
```
```

生のコンテンツは特定の出力フォーマットでのみ表示されることに注意してください. 例えば生の LaTeX コンテンツは出力フォーマットが HTML の場合は無視されます.

9.6 カスタムブロック (*)

bookdown 本の 2.7 節[*3] では, どうすれば R Markdown でブロックの見た目をカスタマイズできるかを話しました. これはレポートや本の中でコンテンツを目立たせる便利な方法で, 読者があなたの著作の中の要点を確実に取りせるようにできます. これらのブロックの使い方の例として, 次のようなものがあります.

- あなたの分析コードを実行する前に, ユーザが最新のパッケージを使用しているか確認するための警告メッセージを表示する.
- ソースコードのある GitHub リポジトリへのリンクを文書の冒頭に追加する.
- あなたの分析から得られた要点や知見を強調する.

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

この節では PDF と HTML の両方でカスタムブロックを作成する方法を説明します．どちらのフォーマットでも R Markdown 上で同じ整形の構文を使用できますが, 要求される設定は異なります．

9.6.1 構文

カスタムブロックの構文は Pandoc の fenced `Div` blocks[*4] に基づいています．`Div` ブロックはとても強力ですが 1 つだけ問題があります．これはおもに HTML 出力に対して動作しますが, LaTeX 出力に対しては動作しないことです．

バージョン 1.16 以降の **rmarkdown** パッケージは `Div` ブロックを HTML と LaTeX どちらに対しても変換するようになりました．HTML 出力に対してはブロックの全ての属性が `<div>` タグの属性になります．例えば `Div` は ID (`#` の後のに続くもの), 1 つまたは複数のクラス (クラス名は `.` の後に書かれるものです), そしてそれ以外の属性を持ちます．以下の `Div` ブロック,

```
::: {#hello .greeting .message style="color: red;"}
Hello **world**!
:::
```

は以下の HTML コードに変換されます．

```
<div id="hello" class="greeting message" style="color: red;">
  Hello <strong>world</strong>!
</div>
```

LaTeX 出力に対しては, 最初のクラス名が LaTeX 環境名として使われます．また, `Div` ブロックに `data-latex` と名付けた属性を与え, 環境の引数としましょう．環境が引数を必要としないなら, この属性は空白にすることができます．2 つの単純な例を以下にお見せします．1 つ目の例は LaTeX で `verbatim` 環境を使用します．これは引数を必要としません．

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

```
::: {.verbatim data-latex=""}
ここに _verbatim_ テキストを表示.
:::
```

LaTeX 出力はこうなります.

```
\begin{verbatim}
ここに \emph{verbatim} テキストを表示.
\end{verbatim}
```

ブロックが HTML へと変換される場合は, HTML コードはこのようになります.

```
<div class="verbatim">
ここに <em>verbatim</em> テキストを表示.
</div>
```

2 つ目の例では `center` と `minipage` 環境を使い, ページ幅の半分の大きさに中央揃えしたボックス内にテキストを表示しています.

```
:::: {.center data-latex=""}

::: {.minipage data-latex="{.5\linewidth}"}
この段落は中央揃えされ，親要素の半分の幅で表示されます.
:::

::::
```

`center` ブロックの中に `minipage` ブロックをネストしていることに注意してください. 親ブロックに子ブロックを入れるには, さらにコロンが必要です. 上記の例では `center` ブロックに 4 つのコロンを使用していますが, 5 個以上書くことも可能です. 2 つのブロックは以下の LaTeX コードに変換されます.

```
\begin{center}
\begin{minipage}{.5\linewidth}
この段落は中央揃えされ，親要素の半分の幅で表示されます．
\end{minipage}
\end{center}
```

HTML 出力では, ユーザーの好みで CSS によって `<div>` ブロックの外見を定義することもできます．LaTeX 出力の場合は, 環境が未定義ならば `\newenvironment` を, 既存の環境を再定義するならば `\renewenvironment` コマンドを LaTeX 上で使うこともできます．LaTeX 上での定義で PDF 上でのブロックの見た目を決定できます．これらのカスタマイズは通常は `style.css` や `preamble.tex` といったファイルを内に記述して, YAML オプションで読み込みます．

```
---
output:
  html_document:
    css: style.css
  pdf_document:
    includes:
      in_header: preamble.tex
---
```

次に, CSS ルールや LaTeX 環境を使用したいくつか発展的なカスタムブロックの実例をお見せします．さらなる使用例としては, 5.8 節に, 多段組みレイアウト内で複数ブロックを並べるものがあります．

9.6.2 影付きブロックを追加する

まず, 影付きボックスの内部にコンテンツを入れる方法を紹介します．ボックスは黒の背景色とオレンジ色の枠があり, 角は丸めます．ボックス内のテキストは白色です．

HTML 出力に対しては, CSS ファイル内でそのルールを定義します．CSS にあまり詳しくなくても, 無料で見られるオンラインチュートリアルが豊富にあります．例えば https://www.w3schools.com/css/ とか[*5]．

[*5] **訳注**: このサイト相当の日本語のサイトを翻訳者は知らないので Moziila のサイトなどを参考にしてください

```css
.blackbox {
  padding: 1em;
  background: black;
  color: white;
  border: 2px solid orange;
  border-radius: 10px;
}
.center {
  text-align: center;
}
```

LaTeX 出力に対しては, LaTeX パッケージの **framed** を基にして `blackbox` という名前で新しい環境を作成し, 黒い背景色と白い文字色にします.

```
\usepackage{color}
\usepackage{framed}
\setlength{\fboxsep}{.8em}

\newenvironment{blackbox}{
  \definecolor{shadecolor}{rgb}{0, 0, 0}  % black
  \color{white}
  \begin{shaded}}
 {\end{shaded}}
```

本書で **framed** パッケージを使うのはこれがかなり軽量だからですが, このパッケージは色付きで角の丸い枠を描画することができません. それを実現するには, より洗練された LaTeX パッケージである **tcolorblox** (https://ctan.org/pkg/tcolorbox) が必要です. このパッケージには影付きボックスを作るためのとても柔軟な一連のオプションがあります. パッケージのドキュメントからは多くの使用例を見ることができます. 以下の LaTeX 環境は上記の CSS の例と似た影付きボックスを作成できます.

---

https://developer.mozilla.org/ja/docs/Learn/Getting_started_with_the_web/CSS_basics (一部未翻訳のページもあるかもしれません)

```
\usepackage{tcolorbox}

\newtcolorbox{blackbox}{
  colback=black,
  colframe=orange,
  coltext=white,
  boxsep=5pt,
  arc=4pt}
```

これで PDF と HTML 出力の両方のフォーマットでカスタムボックスを使用できるようになりました．ボックスのソースコードはこのようになります．

```
:::: {.blackbox data-latex=""}
::: {.center data-latex=""}
**注意!**
:::

この**新しい注意書き**を見てくれてありがとう！あなたがこれを見ていることは監視されてお
↪ り，当局に報告される!
::::
```

出力はこうなります．

注意!

この**新しい注意書き**を見てくれてありがとう！あなたがこれを見ていることは監視されており，当局に報告される!

9.6.3 アイコンを加える

カスタムボックス内に画像を含めることで，より見た目で注意を引く作りにできます．画像はブロックの内容をより効果的に伝える方法にもなりえます．以下に続く例では，次のようなディレクトリ構造で動作させるという前提にしています．これは本書を作成するために使ったものを簡略化

したものです．

```
directory/
├── your-report.Rmd
├── style.css
├── preamble.tex
└── images/
    ├── important.png
    ├── note.png
    ├── caution.png
    └── ...
```

全体がどのように動作するかを説明する前に，この例のソースコードと出力をお見せしましょう．

```
::: {.infobox .caution data-latex="{caution}"}
**注意!**

この**新しい注意書き**を見てくれてありがとう！あなたがこれを見ていることは監視されてお
↪ り，当局に報告される!
:::
```

出力はこうなります．

注意!
この**新しい注意書き**を見てくれてありがとう！あなたがこれを見ていることは監視されており，当局に報告される!

HTML 出力では，CSS の background-image プロパティ内にボックスの画像を追加することができます．背景に画像を挿入し，左側に十分な幅のパディングを追加することでテキストと画像の重なりを避けます．ローカルの画像ファイルを使用するなら，ファイルパスは CSS との相対パスで与えます．これが例です．

```
.infobox {
  padding: 1em 1em 1em 4em;
  margin-bottom: 10px;
```

```
  border: 2px solid orange;
  border-radius: 10px;
  background: #f5f5f5 5px center/3em no-repeat;
}

.caution {
  background-image: url("images/caution.png");
}
```

ブロックに .infobox と .caution という 2 つのクラス名を使用していることに注意してください. infobox クラスは色付きの外枠のある影付きボックスを定義するのに使用し, caution クラスは画像を入れるために使用されています. 2 つのクラスを使用する利点は影付きボックスの設定を繰り返すことなく, いろいろなアイコンの付いたブロックを定義できるということです. 例えば, warning のボックスが必要ならば, .infobox のルールを繰り返し書くことなく, 以下のように定義するだけで十分です.

```
.warning {
  background-image: url("images/warning.png");
}
```

すると以下の Markdown ソースコードで warning ボックスを作成できます.

```
:::: {.infobox .warning data-latex="warning"}

実際のコンテンツをここに表示

::::
```

PDF 出力に対しては, 以前の例で定義した blackbox 環境を基に infobox 環境を作成し, ボックスの左側に画像を追加できます. LaTeX 環境に画像を追加する方法はいくつもあります. これはそのうちの 1 つにすぎません. なお, これは上記の CSS で定義したスタイルを正確に再現するものではありません.

```
\newenvironment{infobox}[1]
  {
  \begin{itemize}
  \renewcommand{\labelitemi}{
    \raisebox{-.7\height}[0pt][0pt]{
      {\setkeys{Gin}{width=3em,keepaspectratio}
        \includegraphics{images/#1}}
    }
  }
  \setlength{\fboxsep}{1em}
  \begin{blackbox}
  \item
  }
  {
  \end{blackbox}
  \end{itemize}
  }
```

以下に, 様々なアイコンを付けたブロックを掲載します.

注意!
この**新しい注意書き**を見てくれてありがとう! あなたがこれを見ていることは監視されており, 当局に報告される!

注意!
この**新しい注意書き**を見てくれてありがとう! あなたがこれを見ていることは監視されており, 当局に報告される!

注意!
この**新しい注意書き**を見てくれてありがとう! あなたがこれを見ていることは監視されており, 当局に報告される!

注意!
この**新しい注意書き**を見てくれてありがとう! あなたがこれを見ていることは監視されており, 当局に報告される!

代替案として, LaTeX パッケージの **awesomebox**[*6] を使って PDF にアイコン付きのボックスを生成することもできます. このパッケージがあれば非常に多くのアイコンを選ぶことができます. 以下に簡単な例をお見せします. 使用可能な LaTeX 環境と引数についてはパッケージのドキュメントを参照してください.

```
---
title: Awesome Boxes
output:
  pdf_document:
    latex_engine: lualatex
    extra_dependencies: awesomebox
documentlass: ltjsarticle
mainfont: Noto Serif CJK JP
sansfont: Noto Sans CJK JP
---

"note" 型のボックス:

::: {.noteblock data-latex=""}
この**新しい注意書き**を見てくれてありがとう! あなたがこれを見ていることは監視されてお
↪ り, あなたがこれを見ていることは監視されており, _当局に報告される_!
:::

このボックスの引数を生成するための R 関数 `box_args()` を定義しました.

```{r}
box_args <- function(
  vrulecolor = 'white',
  hrule = c('\\abLongLine', '\\abShortLine', ''),
```

[*6] https://ctan.org/pkg/awesomebox
```

```
  title = '', vrulewidth = '0pt',
  icon = 'Bomb', iconcolor = 'black'
) {
  hrule <- match.arg(hrule)
  sprintf(
    '[%s][%s][\\textbf{%s}]{%s}{\\fa%s}{%s}',
    vrulecolor, hrule, title, vrulewidth, icon, iconcolor
  )
}
```

インライン R コード内で `` `awesomeblock` `` 環境に引数を与えます.

::: {.awesomeblock data-latex="`r box_args(title = '注意!')`"}
この**新しい注意書き**を見てくれてありがとう!

あなたがこれを見ていることは監視されており，_当局に報告される_!
:::

訳注: 翻訳者の開発した **rmdja** パッケージでは, デフォルトでこの節で紹介されているようなカスタムブロックが定義されており, より簡単にアイコン付きブロックを記述できます. 詳細はこのパッケージのドキュメントをご覧ください.

第10章

表

表は, レポート上で結果を伝えることができる主要な手段です. 表を独自の要件に合った外見に調整したいことはよくあります. この章では表のカスタマイズに使えるテクニックを紹介します. この章のねらいは以下のとおりです.

- 表生成関数 `knitr::kable()` の全ての特徴を紹介する
- **kableExtra** パッケージ (Zhu, 2021) を使用したより発展的な表のカスタマイズに焦点を当てる
- 表を生成してくれる他のパッケージの一覧を提示する

10.1 `knitr::kable()` 関数

knitr パッケージの `kable()` 関数はとてもシンプルな表生成用の関数で, その設計もシンプルです. 行列やデータフレームのように厳密に矩形状のデータに対してのみ表を生成します. 表のセルを細かく整形したりセルを結合したりはできません. しかしこの関数は表の外見をカスタマイズする多くの引数を持っています.

```
kable(x, format, digits = getOption("digits"), row.names = NA,
  col.names = NA, align, caption = NULL, label = NULL,
  format.args = list(), escape = TRUE, ...)
```

10.1.1 サポートする表形式

データオブジェクト x を単純な表で表すことだけが必要ならば, ほとんどの場合, knitr::kable(x) で十分でしょう. format 引数は **knitr** のソース文書フォーマットに従って自動的に設定されます. 引数が取り得る値は列をパイプで区切った pipe, Pandoc 式の単純な表である simple, LaTeX の表 latex, HTML の表 html, reStructuredText (rst) 形式の rst です. R Markdown 文書に対して kable() はデフォルトで pipe フォーマットを使用し, このような外見になります.

```
01 knitr::kable(head(mtcars[, 1:4]), "pipe")
```

```
|                  |  mpg| cyl| disp|  hp|
|:-----------------|----:|---:|----:|---:|
|Mazda RX4         | 21.0|   6|  160| 110|
|Mazda RX4 Wag     | 21.0|   6|  160| 110|
|Ðatsun 710        | 22.8|   4|  108|  93|
|Hornet 4 Ðrive    | 21.4|   6|  258| 110|
|Hornet Sportabout | 18.7|   8|  360| 175|
|Valiant           | 18.1|   6|  225| 105|
```

単純な表, そして HTML, LaTeX, reStructuredText での表を生成できます.

```
01 knitr::kable(head(mtcars[, 1:4]), "simple")
```

```
                   mpg   cyl   disp    hp
-----------------  -----  ----  -----  ----
Mazda RX4           21.0     6    160   110
Mazda RX4 Wag       21.0     6    160   110
Ðatsun 710          22.8     4    108    93
Hornet 4 Ðrive      21.4     6    258   110
```

```
Hornet Sportabout    18.7     8    360   175
Valiant              18.1     6    225   105
```

```
knitr::kable(mtcars[1:2, 1:2], "html")
```

```
<table>
 <thead>
  <tr>
   <th style="text-align:left;">   </th>
   <th style="text-align:right;"> mpg </th>
   <th style="text-align:right;"> cyl </th>
  </tr>
 </thead>
<tbody>
  <tr>
   <td style="text-align:left;"> Mazda RX4 </td>
   <td style="text-align:right;"> 21 </td>
   <td style="text-align:right;"> 6 </td>
  </tr>
  <tr>
   <td style="text-align:left;"> Mazda RX4 Wag </td>
   <td style="text-align:right;"> 21 </td>
   <td style="text-align:right;"> 6 </td>
  </tr>
</tbody>
</table>
```

```
knitr::kable(head(mtcars[, 1:4]), "latex")
```

```
\begin{tabular}{l|r|r|r|r}
\hline
  & mpg & cyl & disp & hp\\
\hline
Mazda RX4 & 21.0 & 6 & 160 & 110\\
\hline
Mazda RX4 Wag & 21.0 & 6 & 160 & 110\\
\hline
Datsun 710 & 22.8 & 4 & 108 & 93\\
\hline
Hornet 4 Drive & 21.4 & 6 & 258 & 110\\
\hline
Hornet Sportabout & 18.7 & 8 & 360 & 175\\
\hline
Valiant & 18.1 & 6 & 225 & 105\\
\hline
\end{tabular}
```

```
knitr::kable(head(mtcars[, 1:4]), "rst")
```

```
==================  ====  ===  ====  ===
\                    mpg  cyl  disp   hp
==================  ====  ===  ====  ===
Mazda RX4           21.0    6   160  110
Mazda RX4 Wag       21.0    6   160  110
Datsun 710          22.8    4   108   93
Hornet 4 Drive      21.4    6   258  110
Hornet Sportabout   18.7    8   360  175
Valiant             18.1    6   225  105
==================  ====  ===  ====  ===
```

pipe と simple のフォーマットのみが移植可能だと覚えておいてください．つまり，これだけがどの出力文書フォーマットでも動作します．それ以外の表形式は特定のフォーマットに対してのみ，例えば format = 'latex' は LaTeX 出力に対してのみの動作です．特定の表形式を使うことでより細かい操作ができますが，代わりに移植性を犠牲にします．

特定の 1 つの表形式だけが必要で，それが文書のデフォルト形式でないなら，knitr.table.format という R のグローバルオプションで一括設定できます．例えばこのように．

```
options(knitr.table.format = "latex")
```

このオプションには，表形式を表す文字列か NULL を返す関数を与えることもできます．NULL の場合は **knitr** は適切な表形式を自動的に決定しようとします．例えば出力フォーマットが LaTeX の場合のみ latex を使用できます．

```
options(knitr.table.format = function() {
  if (knitr::is_latex_output())
    "latex" else "pipe"
})
```

10.1.2 列名を変更する

データフレームの列の名前と読者に見せたいものとが一致するとは限りません．R のデータの列名でよくあるのは，単語を区切るのにスペースを使わずドットやアンダースコアで代用します．これは表を読む上で不自然に感じるでしょう．col.names 引数を使うと列名を新しい名前のベクトルで置き換えることができます．例えば iris データの列名のドットをスペースに置換します．

```
iris2 <- head(iris)
knitr::kable(iris2, col.names = gsub("[.]", " ", names(iris)))
```

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

`col.names` 引数には必ずしも `gsub()` ような関数で列を与える必要はなく, 元のデータオブジェクトの列数と同じ長さであれば, 以下の例のように好きな文字列ベクトルを与えることができます.

```
knitr::kable(
  iris,
  col.names = c('ここ', 'には', '5つの', '名前が', '必要')
)
```

10.1.3 列のアラインメントを指定する

表の各列のアラインメントを変更するには, 左揃え `l`, 中央揃え `c`, 右揃え `r` のどれかと一致する 1 文字づつの文字ベクトルまたは, 1 つの文字列で指定できます. よって `kable(..., align = c('c', 'l'))` は `kable(..., align = 'cl')` に省略できます. デフォルトでは, 数値列は右揃えで, それ以外は左揃えになります. これが使用例です.

```
# 左，中央，中央，中央，右，右揃え
knitr::kable(iris2, align = "lccrr")
```

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

表 10.1: 表のキャプションの例

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

10.1.4 表にキャプションを追加する

`caption` 引数で表にキャプションを追加できます. 以下が例です (表 10.1 参照).

```
knitr::kable(iris2, caption = "表のキャプションの例")
```

4.7 節で言及したように, 出力フォーマットが **bookdown** パッケージ由来のものであれば, キャプションのある表を相互参照することができます.

10.1.5 数値列を整形する

小数点以下の最大表示桁数を `digits` 引数で指定できます. 値は `round()` 関数に与えられるものと同じです. それ以外の整形用の引数は base R の `format()` 関数に与えられるものを `format.args` に与えられます. まず `round()` や `format()` を使ったいくつかの簡単な例をお見せすれば, この後の `kable()` 引数がどう動作するか理解できることでしょう.

```
round(1.234567, 0)
## [1] 1
round(1.234567, digits = 1)
## [1] 1.2
round(1.234567, digits = 3)
## [1] 1.235
format(1000, scientific = TRUE)
```

```
## [1] "1e+03"
format(10000.123, big.mark = ",")
## [1] "10,000"
```

それでは表の数値を丸め整形します.

```
d <- cbind(X1 = runif(3), X2 = 10^c(3, 5, 7), X3 = rnorm(3,
  0, 1000))
# 最大で 4 桁表示
knitr::kable(d, digits = 4)
```

X1	X2	X3
0.8827	1e+03	-479.4422
0.7700	1e+05	284.2424
0.7413	1e+07	-367.3443

```
# 列ごとにそれぞれ丸める
knitr::kable(d, digits = c(5, 0, 2))
```

X1	X2	X3
0.88271	1e+03	-479.44
0.77000	1e+05	284.24
0.74128	1e+07	-367.34

```
# 指数表記を使わせない
knitr::kable(d, digits = 3, format.args = list(scientific = FALSE))
```

X1	X2	X3
0.883	1000	-479.442
0.770	100000	284.242
0.741	10000000	-367.344

```
# 大きな数に対してカンマ区切りする
knitr::kable(d, digits = 3, format.args = list(big.mark = ",",
  scientific = FALSE))
```

X1	X2	X3
0.883	1,000	-479.442
0.770	100,000	284.242
0.741	10,000,000	-367.344

10.1.6 欠損値を表示する

デフォルトでは欠損値 (NA) は表の上で NA という文字で表示されます．これを R のグローバルオプション knitr.kable.NA で他の値に置き換えたり何も表示させない，つまり NA を空白にする，といったことができます．例えば以下の 2 つ目の表では NA を空白に置き換え，3 つ目の表で ** で表示しています．

```
d[rbind(c(1, 1), c(2, 3), c(3, 2))] <- NA
knitr::kable(d)  # デフォルトでは NA は表示される
```

X1	X2	X3
NA	1e+03	-479.4
0.7700	1e+05	NA
0.7413	NA	-367.3

```
# NA を空白に置き換え
opts <- options(knitr.kable.NA = "")
knitr::kable(d)
```

X1	X2	X3
	1e+03	-479.4
0.7700	1e+05	
0.7413		-367.3

```
options(knitr.kable.NA = "**")
knitr::kable(d)
```

X1	X2	X3
**	1e+03	-479.4
0.7700	1e+05	**
0.7413	**	-367.3

```
options(opts)  # グローバルオプションを元に戻す
```

10.1.7 特殊文字をエスケープする

あなたがもし HTML や LaTeX に詳しいなら, これらの言語にいくつかの特殊文字があることを知っているでしょう. 安全に出力するために, `kable()` はデフォルトでは `escape = TRUE` 引数によって特殊文字をエスケープし, これは全ての文字がそのまま表示され, 特殊文字はその特別な意味を失います. 例えば `>` は HTML の表に対しては `>` に置き換えられ, LaTeX の表に対しては `_` は `\_` としてエスケープされます. あなたが専門家で, 特殊文字を適切に扱う方法を知っているなら, `escape = FALSE` 引数によってこれを無効化することもできます. 以下の 2 つ目の表では, 特殊文字である `$`, `\`, `_` を含む LaTeX の数式表現を与えています.

```
m <- lm(dist ~ speed, data = cars)
d <- coef(summary(m))
knitr::kable(d)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-17.579	6.7584	-2.601	0.0123
speed	3.932	0.4155	9.464	0.0000

```
# 行と列の名前に数式表現を与える
rownames(d) <- c("$\\beta_0$", "$\\beta_1$")
colnames(d)[4] <- "$P(T > |t|)$"
```

```
04 knitr::kable(d, escape = FALSE)
```

	Estimate	Std. Error	t value	$P(T > \|t\|)$
β_0	-17.579	6.7584	-2.601	0.0123
β_1	3.932	0.4155	9.464	0.0000

escape = FALSE なしでは特殊文字はエスケープされるか置き換えられます．例えば $ は \$ に，_ は _ に，\ は \textbackslash{} にエスケープされます．

```
01 knitr::kable(d, format = "latex", escape = TRUE)
```

```
\begin{tabular}{l|r|r|r|r}
\hline
  & Estimate & Std. Error & t value & \$P(T > |t|)\$\\
\hline
\$\textbackslash{}beta\_0\$ & -17.579 & 6.7584 & -2.601 & 0.0123\\
\hline
\$\textbackslash{}beta\_1\$ & 3.932 & 0.4155 & 9.464 & 0.0000\\
\hline
\end{tabular}
```

LaTeX で他によく知られた特殊文字として，#, %, &, {, } があります．HTML のよく知られた特殊文字は &, <, > そして " です．escape = FALSE で表を生成する際には，正しい方法で特殊文字を使うよう注意深くなるべきです．とてもよくある失敗として，LaTeX で escape = FALSE を使いつつ，% や _ が特殊文字であると気づかずに表の列名やキャプションに含んでしまうというものがあります．

特殊文字のエスケープの方法を正しく知っている自信がないなら **knitr** には 2 つのヘルパー内部関数があります．以下はその例です．

```
01 knitr:::escape_latex(c("100%", "# コメント", "列名"))
```

表 10.2: 横に並べられた 2 つの表

speed	dist
4	2
4	10
7	4

	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

```
## [1] "100\\%"       "\\# コメント" "列名"
```

```
knitr:::escape_html(c("<アドレス>", "x = \"文字列\"",
  "a & b"))
```

```
## [1] "&lt; アドレス&gt;"       "x = &quot; 文字列&quot;"
## [3] "a &amp; b"
```

10.1.8 複数の表を横に並べる

データフレームや行列のリストを kable() に与えて, 複数の表を並べて生成することができます. 例えば表 10.2 は以下のコードから生成された 2 つの表を含んでいます.

```
d1 <- head(cars, 3)
d2 <- head(mtcars[, 1:3], 5)
knitr::kable(
  list(d1, d2),
  caption = '横に並べられた 2 つの表',
  booktabs = TRUE, valign = 't'
)
```

この機能は HTML と PDF 出力でのみ機能することに注意してください.

表を横に並べて個別の表をカスタマイズできるようにしたいと考えているなら, kables() 関数 (つ

表 10.3: knitr::kables() によって作成された 2 つの表.

速さ	距離
4	2
4	10
7	4

	mpg	cyl	disp
Mazda RX4	21	6	160
Mazda RX4 Wag	21	6	160
Datsun 710	23	4	108
Hornet 4 Drive	21	6	258
Hornet Sportabout	19	8	360

まり, kable() の複数形を意味しています) を使い, kable() オブジェクトのリストを与えることもできます. 例えば, 表 10.3 の左の表の列名を変更し, かつ右の表の表示桁数をゼロに変更します.

```
# データオブジェクト d1, d2 は以前のコードチャンクのもの
knitr::kables(
  list(
    # 第 1 の kable() は列名を変更する
    knitr::kable(
      d1, col.names = c('速さ', '距離'), valign = 't'
    ),
    # 第 2 の kable() は表示桁数を設定する
    knitr::kable(d2, digits = 0, valign = 't')
  ),
  caption = 'knitr::kables() によって作成された 2 つの表.'
)
```

10.1.9 for ループから複数の表を作成する (*)

kable() に関してよく混乱することの 1 つは, for ループ内では動作しないということです. この問題は kable() に限らず他のパッケージにも存在します. 原因は少々複雑です. 技術的な話に関心があるなら, “The Ghost Printer behind Top-level R Expressions.”[*1] というブログ記事で解説されています.

以下のコードチャンクは 3 つの表を生成する, とあなたは予想するかもしれませんが, そうはなり

[*1] https://yihui.org/en/2017/06/top-level-r-expressions/

ません.

```
```{r}
for (i in 1:3) {
  knitr::kable(head(iris))
}
```
```

明示的に kable() の結果をプリントし, チャンクオプション results = 'asis' を適用しなければなりません. 例えばこのように.

```
```{r, results='asis'}
for (i in 1:3) {
  print(knitr::kable(head(iris)))
}
```
```

一般に, for ループ内で出力を生成するときは, 出力する要素をそれぞれ区別するためにそれぞれの直後に改行コード (\n) または HTML のコメント行 (<!-- -->) を加えることをおすすめします. これが例です.

```
```{r, results='asis'}
for (i in 1:3) {
  print(knitr::kable(head(iris), caption = 'A caption.'))
  cat('\n\n<!-- -->\n\n')
}
```
```

セパレータがないと Pandoc は個別の要素を検出するのに失敗します. 例えばグラフのすぐ後に表を続けて書いたとき, 表が認識されなくなります.

```
![](logo.png)
                     mpg   cyl   disp    hp
------------------  -----  ----  -----  ----
Mazda RX4            21.0     6    160   110
Mazda RX4 Wag        21.0     6    160   110
```

しかし明示的に分離した場合はこうなります．以下では画像の直後に空白行を挟んでいることに気をつけてください．

```
![](logo.png)

                     mpg   cyl   disp    hp
------------------  -----  ----  -----  ----
Mazda RX4            21.0     6    160   110
Mazda RX4 Wag        21.0     6    160   110
```

あるいはこのように．

```
![](logo.png)

<!-- -->

                     mpg   cyl   disp    hp
------------------  -----  ----  -----  ----
Mazda RX4            21.0     6    160   110
Mazda RX4 Wag        21.0     6    160   110
```

10.1.10　LaTeX の表をカスタマイズする (*)

必要なのが LaTeX の出力のみなら，さらにいくつか `kable()` のオプションがあります．これらは HTML 等，他の種類のフォーマットでは無視されることに注意してください．表のフォーマットオプションをグローバルに設定 (10.1.1 節参照) していない限り，この節の例では `kable()` の `format`

引数を明示的に使わなければなりません.

```
knitr::kable(iris2, format = "latex", booktabs = TRUE)
```

表のキャプションを設定 (10.1.4 節参照) している場合, kable() は表を table 環境で囲みます. つまりこうなります.

```
\begin{table}
% the table body (usually the tabular environment)
\end{table}
```

この環境は table.envir 引数で次のように変更できます.

```
knitr::kable(cars[1:2, ], format = "latex", table.envir = "figure")
```

```
\begin{figure}
\begin{tabular}{r|r}
\hline
speed & dist\\
\hline
4 & 2\\
\hline
4 & 10\\
\hline
\end{tabular}
\end{figure}
```

表のフロート位置は position 引数によって制御されます. 例えば position = "!b" によって表のフロートをページ下部に置くことを強制できます.

```
knitr::kable(cars[1:2, ], format = "latex", table.envir = "table",
  position = "!b")
```

```
\begin{table}[!b]
\begin{tabular}{r|r}
\hline
speed & dist\\
\hline
4 & 2\\
\hline
4 & 10\\
\hline
\end{tabular}
\end{table}
```

表にキャプションがある場合, caption.short 引数でこの例のようにキャプションの短縮形を与えることもできます.

```
knitr::kable(iris2, caption = "長い長いキャプション",
  caption.short = "短いキャプション")
```

キャプションの短縮形は LaTeX 上では \caption[]{} コマンドのブラケット ([]) 内に与えられ, ほとんどの場合は出力された PDF の表一覧で使用されます. 短縮形がない場合は, キャプション全文が表示されます.

出版物レベルのクオリティで作表するための LaTeX パッケージ **booktabs**[*2] に詳しいなら, この例のように booktabs = TRUE を設定できます.

```
iris3 <- head(iris, 10)
knitr::kable(iris3, format = "latex", booktabs = TRUE)
```

[*2] https://ctan.org/pkg/booktabs

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

R Markdown で **booktabs** のような LaTeX パッケージが追加で必要なら, YAML で宣言しなければならないことを忘れないでください (やり方は 6.4 節参照).

引数 `booktabs` が `TRUE` か `FALSE` (デフォルト) であるかに依存して表の外見は変わります.

`booktabs = FALSE` の場合

- 表の列が垂直線で区切られます. `vline` 引数を使って垂直線を削除することができます. 例えば `knitr::kable(iris, vline = "")` と言うふうにします. デフォルトは `vline = "¦"` です. このオプションをグローバルに設定することもでき, 表ごとに指定する必要はありません. 例えば `options(knitr.table.vline = "")` とします.
- 水平線を `toprule`, `midrule`, `linesep`, `bottomrule` 引数で定義できます. これらのデフォルト値は `\hline` です.

`booktabs = TRUE` の場合

- 表に垂線はありませんが, `vline` 引数で追加することができます.
- テーブルのヘッダと末尾にのみ水平線が描かれます. デフォルトの引数の値は `toprule = "\\toprule"`, `midrule = "\\midrule"`, `bottomrule = "\\bottomrule"` です. デフォルトでは 1 行分の空きが 5 行ごとに挿入されます. これは `linesep` 引数で制御でき, このデフォルトは `c("", "", "", "", "\\addlinespace")` となっています. 3 行ごとに空白を与えたいなら, このようにできます.

```
knitr::kable(iris3, format = "latex", linesep = c("", "",
  "\\addlinespace"), booktabs = TRUE)
```

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

行空けを完全に削除したいなら, `linesep = ''` とこともできます.

表がページよりも長くなってしまうもともあるでしょう. そのような場合は `longtable = TRUE` を使用できます. このオプションは LaTeX パッケージ **longtable**[*3] を使い表を複数ページに分割します.

table 環境に含まれた場合, つまり表にキャプションを設定した場合は表はデフォルトで中央揃えになります. 表を中央揃えにしたくないなら, `centering = FALSE` 引数を使用してください.

10.1.11 HTML の表をカスタマイズする (*)

`knitr::kable(format = "html")` で生成した表をカスタマイズしたいなら, 前節で紹介した共通の引数の他に, 1 つだけ `table.attr` という特別な引数があります. この引数で任意の属性を `<table>` タグに追加することができます. 例えばこのように.

```
01 knitr::kable(mtcars[1:2, 1:2], table.attr = "class=\"striped\"",
02   format = "html")
```

```
<table class="striped">
 <thead>
```

[*3] https://ctan.org/pkg/longtable

```
  <tr>
   <th style="text-align:left;">   </th>
   <th style="text-align:right;"> mpg </th>
   <th style="text-align:right;"> cyl </th>
  </tr>
 </thead>
<tbody>
  <tr>
   <td style="text-align:left;"> Mazda RX4 </td>
   <td style="text-align:right;"> 21 </td>
   <td style="text-align:right;"> 6 </td>
  </tr>
  <tr>
   <td style="text-align:left;"> Mazda RX4 Wag </td>
   <td style="text-align:right;"> 21 </td>
   <td style="text-align:right;"> 6 </td>
  </tr>
</tbody>
</table>
```

表に striped クラスを追加しています．しかしクラス名だけでは表の外見を変更するのに不十分です．クラスに対して CSS ルールを定義しなければなりません．例えば偶数列と奇数列とで色の異なるストライプ背景の表を作るには，明灰色の背景を偶数または奇数列に追加できます．

```
.striped tr:nth-child(even) { background: #eee; }
```

上記の CSS ルールの意味は，striped クラスを持つ要素の子要素ですべての行（つまり <tr> タグ）のうち行番号が偶数属性の (:nth-child(even)) 要素は，背景色が #eee になるということです．

少しの CSS の記述だけでプレーンの HTML の表の見栄えをよくできます．図 10.1 は，以下の CSS ルールを適用した HTML 表のスクリーンショットです

```
table {
```

```
  margin: auto;
  border-top: 1px solid #666;
  border-bottom: 1px solid #666;
}
table thead th { border-bottom: 1px solid #ddd; }
th, td { padding: 5px; }
thead, tfoot, tr:nth-child(even) { background: #eee; }
```

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

図 10.1: HTML と CSS で作成したストライプ背景の表

10.2 kableExtra パッケージ

kableExtra package (Zhu, 2021) は `knitr::kable()` (10.1 節参照) を使用して作成した表の基本機能を拡張するために設計されました. `knitr::kable()` はシンプルな設計なので (これは Yihui が怠け者であるという意味にとるのはご随意に！), 他のパッケージで見られるような機能の多くが決定的に失われてしまっています. そして **kableExtra** はこのギャップを完全に埋めてくれます. **kableExtra** について最も驚異することは, 表のほとんどの機能, 例えば, 図 10.1 のようなストライプ背景の表をつくるなどが HTML でも PDF でも動作することです.

このパッケージはいつものように CRAN からインストールできますし, GitHub (https://github.com/haozhu233/kableExtra) から開発版をインストールすることもできます.

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

```
# install from CRAN
install.packages("kableExtra")

# install the development version
remotes::install_github("haozhu233/kableExtra")
```

発展的なドキュメントが https://haozhu233.github.io/kableExtra/ にあり，kable() の出力を HTML や LaTeX 出力でどうカスタマイズするかについて多くの使用例が掲載されています．我々としてはご自分でドキュメントを読むことをおすすめし，ここでは一部の例だけを提示します．

kableExtra パッケージはパイプ演算子 %>% を前面に出しています．kable() の出力に **kableExtra** のスタイル関数を接続することができます．例えばこのように．

```
library(knitr)
library(kableExtra)
kable(iris) %>%
  kable_styling(latex_options = "striped")
```

10.2.1 フォントサイズを設定する

kableExtra パッケージの kable_styling() 関数によってテーブル全体のスタイルを設定できます．例えばページ上での表のアラインメント，幅，フォントサイズなどです．以下は小さいフォントサイズを使う例です．

```
kable(head(iris, 5), booktabs = TRUE) %>%
  kable_styling(font_size = 8)
```

10.2.2　特定の行・列のスタイルを設定する

関数 row_spec() と column_spec() はそれぞれ個別の行と列のスタイル設定に使うことができます. 以下の例では第 1 行をボールドイタリックにし, 第 2, 第 3 行を黒色背景と白色文字にし, 第 4 行にアンダーラインを引きタイプフェースを変更し, 第 5 行を回転させ, そして第 5 列に打ち消し線を引きます.

```
kable(head(iris, 5), align = 'c', booktabs = TRUE) %>%
  row_spec(1, bold = TRUE, italic = TRUE) %>%
  row_spec(2:3, color = 'white', background = 'black') %>%
  row_spec(4, underline = TRUE, monospace = TRUE) %>%
  row_spec(5, angle = 45) %>%
  column_spec(5, strikeout = TRUE)
```

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

同様に, cell_spec() 関数で個別のセルにスタイル設定できます.

10.2.3　行・列をグループ化する

行や列をそれぞれ, pack_rows() と add_header_above() 関数でまとめることができます. collapse_rows() 関数で行を崩し, セルを複数行にまたがらせることができます. 以下は行をグループ化したカスタムテーブルヘッダの例です.

```
iris2 <- iris[1:5, c(1, 3, 2, 4, 5)]
names(iris2) <- gsub('[.].+', '', names(iris2))
kable(iris2, booktabs = TRUE) %>%
```

```
  add_header_above(c("長さ" = 2, "幅" = 2, " " = 1)) %>%
  add_header_above(c("Measurements" = 4, "More attributes" = 1))
```

Measurements				More attributes
長さ		幅		
Sepal	Petal	Sepal	Petal	Species
5.1	1.4	3.5	0.2	setosa
4.9	1.4	3.0	0.2	setosa
4.7	1.3	3.2	0.2	setosa
4.6	1.5	3.1	0.2	setosa
5.0	1.4	3.6	0.2	setosa

add_header_above() 内の名前付きベクトルに対して, 名前がテーブルヘッダにテキストとして表示され, 整数値のベクトルが対応する名前の列の数を表します. 例えば "Length" = 2 が Length が 2 列にまたがることを意味します.

以下は pack_rows() の例です. index 引数の意味は既に説明した add_header_above() の引数と似ています.

```
iris3 <- iris[c(1:2, 51:54, 101:103), ]
kable(iris3[, 1:4], booktabs = TRUE) %>% pack_rows(
  index = c("setosa" = 2, "versicolor" = 4, "virginica" = 3)
)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
setosa				
1	5.1	3.5	1.4	0.2
2	4.9	3.0	1.4	0.2
versicolor				
51	7.0	3.2	4.7	1.4
52	6.4	3.2	4.5	1.5
53	6.9	3.1	4.9	1.5
54	5.5	2.3	4.0	1.3
virginica				
101	6.3	3.3	6.0	2.5
102	5.8	2.7	5.1	1.9
103	7.1	3.0	5.9	2.1

10.2.4 LaTeX で表を縮小する

HTML や LaTeX 出力特有の機能もいくつかあります．例えば横向きページは LaTeX でのみ意味をなすので，**kableExtra** の `landscape()` 関数は LaTeX でのみ機能します．以下はページに合わせて表を縮小する例です．縮小しなければ横に長すぎる表になります．

```
tab <- kable(tail(mtcars, 5), booktabs = TRUE)
tab  # 長すぎる元の表
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.5	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.5	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.6	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.6	1	1	4	2

```
tab %>%
  kable_styling(latex_options = "scale_down")
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.5	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.5	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.6	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.6	1	1	4	2

HTML 版をご覧なら, 上の 2 つの表に違いが見られないでしょう.

10.3 その他の表作成パッケージ

多くの作表用 R パッケージがあります. `kable()` (10.1 節) と **kableExtra** (10.2 節) を紹介した) 主な理由は他のパッケージより良いからではなく, 私がこれらにのみ詳しかったからです.[*4] 存在は知っていますがあまり詳しくないパッケージを次に列挙します.[*5] ご自分で確認し, 目的に最も合っているものを決めることができます.

- **flextable** (Gohel, 2021a) と **huxtable** (Hugh-Jones, 2021): 幅広い種類の出力フォーマットをサポートするパッケージを探しているなら, **flextable** と **huxtable** が最善の 2 つの選択です. HTML, LaTeX. そして Office フォーマットを全てサポートし, よく使われる表の機能 (例えば条件付き書式とか) のほとんどをサポートしています. **flextable** の詳細は `https://davidgohel.github.io/flextable/` で, **huxtable** のドキュメントは `https://hughjonesd.github.io/huxtable/` で見られます.
- **gt** (Richard Iannone Cheng, and Schloerke, 2021): 表のヘッダ, (題名・副題), 列のラベル, 表の本体, 行グループのラベル, 表のフッタといった異なる表のパーツをまとめて表を構成することができます. 数字のフォーマットを指定したり, セルの背景色に影をつけたりもできます. 現在は **gt** は主に HTML 出力をサポートしています.[*6] 詳細は `https://gt.rstudio.com` で見られます.
- **formattable** (Ren and Russell, 2021): `percent()`, `accounting()` といった数値を整形するものや, テキストの書式, 背景色やカラーバー, アイコンの追加などで数値を強調するなど, 表の列のスタイルを設定する関数を提供してくれます. **gt** のように, このパッケー

[*4] 平たく言うと, 自分では表を全く使いませんから, 洗練された表を作る方法を学ぶ強いモチベーションがありませんでした.

[*5] **訳注**: これらの差異について, 最新の情報とは限りませんし, 必ずしも網羅的ではないですが, 翻訳者自身の作成したドキュメントでいくらか言及しています. `https://gedevan-aleksizde.github.io/rmdja/advanced-tabulate.html`

[*6] LaTeX や Word といった他の出力フォーマットへのサポートが必要ならば, **gtsummary** パッケージ (Sjoberg Curry, et al., 2021) はとても有望な **gt** を下地に拡張しています. `https://github.com/ddsjoberg/gtsummary`

ジも主に HTML フォーマットをサポートしています．詳細は GitHub プロジェクトの https://github.com/renkun-ken/formattable で見ることができます．

- **DT** (Xie Cheng, and Tan, 2021): 作者なのでこのパッケージには精通していると思っていますが, HTML フォーマットのみのサポートのため独立した節を設けて紹介したりはしません．**DT** は JavaScript ライブラリの **DataTables** を下地に構築されたもので, HTML ページ上で静的な表をインタラクティブな表に変えることができます．表をソートしたり, 検索したり, ページ移動したりできるでしょう．**DT** はセルの整形もサポートしており, インタラクティブなアプリケーションの構築のため Shiny と連携して動作し, 多くの **DataTables** の拡張を導入します．例えばエクセルへのエクスポート, 列の並び替えなどです．詳細はパッケージのリポジトリ https://github.com/rstudio/DT を見てください．
- **reactable** (Lin, 2020): **DT** と同様にこのパッケージは JavaScript ライブラリを元にしてインタラクティブな表を作成します．正直に言うと, 私が見る限り, 行のグループ化や HTML ウィジェットの埋め込み機能などいくつかの観点で **DT** より優れているようです．もし **reactable** が 2015 年時点で存在していれば, 私は **DT** を開発していなかったと思います．とは言うものの, **reactable** は **DT** にあるすべての機能を揃えていません．よってあなたはこのパッケージのドキュメント https://glin.github.io/reactable/ を読み, どちらが目的に合ったものかを知ることもできるでしょう．
- **rhandsontable** (Owen, 2021): これも **DT** と似ており, そして表上でデータを直接編集できるなど Excel っぽさがあります．詳しく学ぶには https://jrowen.github.io/rhandsontable/ を見てください．
- **pixiedust** (Nutter, 2021): **broom** パッケージ (Robinson Hayes, and Couch, 2021) を介した統計モデル (線形モデルとか) 向けの表を作るのが特徴です．Markdown, HTML, LaTeX 出力フォーマットをサポートしています． リポジトリは https://github.com/nutterb/pixiedust です．
- **stargazer** (Hlavac, 2018): 回帰モデルと要約統計量の表を整形するのが特徴です．このパッケージは CRAN の https://cran.r-project.org/package=stargazer にあります．
- **xtable** (Dahl Scott, et al., 2019): おそらく最古の作表パッケージです．最初のリリースは 2000 年になります．LaTeX と HTML フォーマットの両方をサポートしています．パッケージは CRAN の https://cran.r-project.org/package=xtable にあります．

その他のパッケージは紹介しませんが, 名前だけ挙げておきます． **tables** (Murdoch, 2020), **pander** (Daróczi and Tsegelskyi, 2021), **tangram** (Garbett, 2020), **ztable** (Moon, 2020), **condformat** (Oller Moreno, 2020) があります．

第11章

チャンクオプション

図 2.1 に描かれているように, **knitr** パッケージは R Markdown においてきわめて重要な役割を担います. この章と後に続くの 3 つの章では **knitr** に関連するレシピをお見せします.

R のチャンクを処理する際には, **knitr** の挙動を細かく調整するのに 50 を超すチャンクオプション (chunk options) が使えます. 完全なリストは https://yihui.org/knitr/options/ のオンラインドキュメントをご覧ください.[*1] 利便性のため, 本書の 付録 A としてこのドキュメントのコピーを掲載しました.

このあと続く各節では, チャンクオプションを個別のコードチャンクに適用する例のみを示します. ただし, どのチャンクオプションもグローバル設定で文書全体に適用できるので, コードチャンク 1 つ 1 つに繰り返しオプションを書かなくても良いということも知っておいてください. グローバルにチャンクオプションを設定するには, いずれかのコードチャンクで `knitr::opts_chunk$set()` を呼び出してください. ふつうは文書の最初のチャンクオプションに設定します. 例えばこのように.

````
```{r, include=FALSE}
knitr::opts_chunk$set(
  comment = "#>", echo = FALSE, fig.width = 6
)
```
````

[*1] **訳注**: 翻訳者による日本語訳はこちら: https://gedevan-aleksizde.github.io/knitr-doc-ja/options.html

11.1　チャンクオプションに変数を使う

大抵の場合, 例えば `fig.width = 6` のようにチャンクオプションは定数をとりますが, 簡単であるか複雑であるかに関わらず, 任意の R コードからの値をとることもできます. 特殊なケースはチャンクオプションに通せる変数です. 変数もまた R コードであることに注意してください. 例えば文書の冒頭で図の幅を変数で定義して, その変数を後の他のコードチャンクで使えば, それ以降の幅を簡単に変更できます.

````
```{r}
my_width <- 7
```

```{r, fig.width=my_width}
plot(cars)
```
````

以下はチャンクオプショで `if-else` 文を使う例です.

````
```{r}
fig_small <- FALSE  # 大きい図に対しては TRUE に変更
width_small <- 4
width_large <- 8
```

```{r, fig.width=if (fig_small) width_small else width_large}
plot(cars)
```
````

さらに以下にもう 1 つの例として, 必要なパッケージが使用可能な場合のみコードチャンクを評価する (つまり実行する) ものを示します.

```
```{r, eval=require('leaflet')}
library(leaflet)
leaflet() %>% addTiles()
```
```

意図が分からない方のために説明しますと, `require('package')` はパッケージが使用可能なら `TRUE` を返し, そうでないなら `FALSE` を返します.

11.2 エラーが起こっても中止しない

時として, 例えば R のチュートリアルのために, わざとエラーを見せたいこともあるかもしれません. デフォルトでは, Rmd 文書のコードチャンクでのエラーは R の処理を停止させます. R の処理を停めることなくエラーを見せたいなら, 例えばこのように `error = TRUE` チャンクオプションを使うこともできます.

```
```{r, error=TRUE}
1 + "a"
```
```

Rmd 文書をコンパイルすると, 出力文書上でのエラーメッセージはこのような見た目になります.

```
Error in 1 + "a": 二項演算子の引数が数値ではありません
```

R Markdown では `error = FALSE` がデフォルトであり, これはコードチャンクの実行時のエラーは処理を停止させます.

11.3 同じグラフを複数の出力フォーマットに

ほとんどの場合, 1 つの図に対して `png` や `pdf` といった 1 つの画像フォーマットにしたいでしょう. 画像フォーマットはチャンクオプション `dev` で操作できます. つまり, グラフをレンダリングするグラフィックデバイスを意味します. このオプションはデバイス名のベクトルをとることができます. これが例です.

```
```{r, dev=c('png', 'pdf', 'svg', 'tiff')}
plot(cars)
```
```

出力文書には最初のフォーマットのみが使われますが, 残りのフォーマットに対応する画像も生成されます. 例えば, レポートでは png 画像を掲載するが, 同じ画像の tiff 形式も求められるときなど, 追加で異なるフォーマットの図の提出が要求されるような場合に便利でしょう.

デフォルトでは, 通常なら画像ファイルは出力文書がレンダリングされた後に削除されます. ファイルを保持する方法は 16.5 節を参照してください.

11.4 時間のかかるチャンクをキャッシュする

コードチャンクの実行に時間がかかる場合, チャンクオプション cache = TRUE で結果をキャッシュすることを検討するとよいでしょう. キャッシュが有効な場合, このコードが以前にも実行され, その後コードに変更がないならば, **knitr** はこの実行を飛ばします. コードチャンクを変更し, つまりコードまたはチャンクオプションを修正したなら, 過去のキャッシュは自動的に無効になり **knitr** はもう一度チャンクをキャッシュします.

キャッシュされたコードチャンクに関しては, チャンクが再度実行されたかのように, 過去の実行結果から出力とオブジェクトが自動的に読み込まれます. キャッシュを読み込むほうが結果を計算するよりはるかに速いという場合に役に立ちます. しかしながら, うまい話というのは世に存在しません. あなたの使う場面によっては, キャッシュがどのように動作するかをより学びぶ必要があるでしょう. 特に cache invalidation[*2] を学べば, **knitr** がしょっちゅうキャッシュを無効化したり, あるいは時に無効化が十分しないことに振り回されることなく, キャッシュの利点を最大限活かすことができます.

最も適切なキャッシュの使用例は, コードチャンク内での計算に非常に時間がかかる R オブジェクトの保存と再読込に使うことですが, コードが options() を使って R のグローバルオプションを変更するといった副産物 (このような変更はキャッシュされません) があってはなりません. コードチャンクに副産物があるなら, キャッシュを使わないことをお薦めします.

最初のほうで簡単に書いたように, キャッシュの動きはチャンクオプションに依存します. もし include 以外のチャンクオプションを変更したら, キャッシュは無効化されます. この仕様はよくある問題を解決してくれます. それは外部データファイルを読み込むときに, ファイルが更新されていたならキャッシュを無効化したい, というような場合です. 単純に cache = TRUE を使うだけで

[*2] https://yihui.org/en/2018/06/cache-invalidation/

は不十分です.

```
```{r import-data, cache=TRUE}
d <- read.csv('my-precious.csv')
```
```

knitr にデータファイルが変更されたかどうかを教えなければなりません. 1 つの方法としては別のチャンクオプション `cache.extra = file.mtime('my-precious.csv')` を加えることですが, あるいはより厳密には `cache.extra = tools::md5sum('my-precious.csv')` を追加することがあります. 前者はファイルの更新時刻が変更されたらキャッシュを無効化する, という意味です. 後者はファイルの中身が変更されたらキャッシュを更新するということです. `cache.extra` は **knitr** の組み込みのチャンクオプション名ではないということに注意してください. 他の組み込みのオプション名と競合しない限り, この用途のオプションには好きな名前を使うことができます.

同様に, 他の情報をキャッシュと関連付けることができます. 例えば R のバージョンなら `cache.extra = getRversion()`, システム日付なら `cache.extra = Sys.Date()`, オペレーティングシステムなら `cache.extra = Sys.info()[['sysname']]` というようにすると, これらの条件が変更されたときにキャッシュは正しく無効化されます.

文書全体で `cache = TRUE` を設定することはお薦めしません. キャッシュはかなり扱いにくいものです. 代わりに, 実行に時間がかかり副産物がないとはっきりしているコードチャンクに限って, 個別にキャッシュを有効化することをお薦めします.

knitr のキャッシュの設計に不満があるなら, 自分でオブジェクトのキャッシュを取ることもできます. 以下はごく簡単な例です.

```
01 if (file.exists("results.rds")) {
02   res <- readRDS("results.rds")
03 } else {
04   res <- compute_it()  # 実行時間のかかる関数
05   saveRDS(res, "results.rds")
06 }
```

この例では, キャッシュを無効化する唯一の, そして簡単な方法は `result.rds` ファイルを削除することです. この簡単なキャッシュのしくみが気に入ったなら, 14.9 節で紹介する `xfun::cache_rds()` を使うこともできます.

11.5　複数の出力フォーマットに対してチャンクをキャッシュする

cache = TRUE でキャッシュが有効化されたとき, **knitr** は R コードチャンクで生成された R オブジェクトをキャッシュデータベースに書き込みます. これで次回から再読込ができます. キャッシュデータベースのパスはチャンクオプション cache.path によって決まります. R Markdown はデフォルトで出力フォーマットごとに異なるキャッシュのパスを使用するので, 時間のかかるコードチャンクは出力フォーマットごとに丸ごと実行されることになります. これは不便かもしれませんが, これがデフォルトの挙動であることには理由があります. コードチャンクの出力を, 固有の出力フォーマットによって決められるからです. 例えばグラフを生成した時, 出力フォーマットが word_document なら ![text](path/to/image.png) のような Markdown 構文で図を掲載できますし, 出力フォーマットが html_document なら が使えます.

コードチャンクにグラフなど副産物が一切ないときは, 全ての出力フォーマットで同じキャッシュデータベースを使っても安全であり, 時間を節約できます. 例えば大きなデータオブジェクトを読み込むか時間のかかるモデルを実行するかというときは, 結果は出力フォーマットに依存しませんので, 同じキャッシュデータベースを使うことができます. コードチャンクに cache.path を指定することでデータベースのパスを指定できます. これが例です.

```
```{r important-computing, cache=TRUE, cache.path="cache/"}
```
```

R Markdown ではデフォルトでは cache.path = "INPUT_cache/FORMAT/" で, INPUT には入力ファイル名が, FORMAT には html, latex, docx といった出力フォーマット名が入ります.

11.6　巨大オブジェクトをキャッシュする

チャンクオプション cache = TRUE を使うと, キャッシュされたオブジェクトは R セッション内で遅延読み込みされます. これはオブジェクトが実際にコード内で使用されるまでキャッシュデータベースから読み込まれないことを意味します. このおかげで, 以降の文書内で全てのオブジェクトが使われるわけではない場合にメモリを多少節約することができます. 例えば大きなデータオブジェクトを読み込んだが, 以降の分析ではその一部しか使わないなら, 元のデータオブジェクトはキャッシュデータベースから読み込まれません.

```
```{r, read-data, cache=TRUE}
full <- read.csv("HUGE.csv")
rows <- subset(full, price > 100)
# next we only use `rows`
```

```{r}
plot(rows)
```
```

しかし, オブジェクトが大きすぎるときは, このようなエラーに遭遇するかもしれません.

```
Error in lazyLoadDBinsertVariable(vars[i], ...
  long vectors not supported yet: ...
Execution halted
```

この問題が発生したら, チャンクオプション `cache.lazy = FALSE` で遅延読み込みを無効にしてみてください. チャンク内の全てのオブジェクトが即座にメモリに読み込まれるでしょう.

11.7 コード, テキスト出力, メッセージ, グラフを隠す

デフォルトでは, **knitr** はコードチャンクから, ソースコード・テキスト出力・メッセージ・警告・エラー・グラフといった可能な全ての出力を表示します. 個別に対応するコードチャンクを使って, 隠すことができます.

ソースコードを隠す.

```
```{r, echo=FALSE}
1 + 1
```
```

テキスト出力を隠す. `` `results = FALSE` `` **を使うのも可.**

````
```{r, results='hide'}
print("テキスト出力はあなたには見えない.")
```

メッセージを隠す.

```{r, message=FALSE}
message("このメッセージはあなたには見えない.")
```

警告メッセージを隠す.

```{r, warning=FALSE}
# 警告を発生させるが抑制される
1:2 + 1:3
```

グラフを隠す.

```{r, fig.show='hide'}
plot(cars)
```

上記のチャンクではグラフが生成されることに注意してください. 出力に表示しなくするだけ
↪ です.
````

knitr に関するよくある質問の 1 つは, パッケージ読み込み時のメッセージを隠す方法です. 例えば `library(tidyverse)` や `library(ggplot2)` を使ったとき, いくつかの読み込みメッセージが現れます. このようなメッセージはチャンクオプション `message = FALSE` で抑制することもできます.

インデックスによってこれらの要素を表示したり隠したり選択することも出来ます. 以下の例では, ソースコードの 4 つ目と 5 つ目の式を表示し, 最初の 2 つのメッセージと 2 つ目と 3 つ目の警告を隠しています. コメントも式 1 つとして数えられることに注意してください.

```
```{r, echo=c(4, 5), message=c(1, 2), warning=2:3}
# 乱数 N(0, 1) を生成する方法の 1 つ
x <- qnorm(runif(10))
# だが rnorm() を使うほうが実用的
x <- rnorm(10)
x

for (i in 1:5) message('ここにメッセージ ', i)

for (i in 1:5) warning('ここにメッセージ ', i)
```
```

負のインデックスを使用することもできます. 例えば echo = -2 は出力部のソースコードの 2 つ目の式を除外します.

同様に, fig.keep オプションに対してインデックスを使うことでどのグラフを表示あるいは隠すかを選ぶこともできます. 例えば fig.keep = 1:2 は最初の 2 つのグラフを残すことを意味します. このオプションにはいくつかのショートカットがあります. fig.keep = "first" は最初のグラフのみを残し, fig.keep = "last" は最後のグラフのみを残し, fig.keep = "none" は全てのグラフを破棄します. 2 つのオプション fig.keep = "none" と fig.show = "hide" は異なることに注意してください. 前者はそもそも画像ファイルを生成しませんが, 後者はグラフを生成し隠すだけです.

html_document 出力のソースコードブロックに対して, echo = FALSE で完全に省略したくなければ, 7.5 節をご覧になれば, ページ上でブロックを折りたたみ, ユーザーが展開ボタンを押して展開させるようにする方法が書いてあります.

11.8 チャンクの出力を全て隠す

ときには出力を全く表示させずにコードチャンクを実行したいかもしれません. 11.7 節で言及したような方法で個別にオプションを使うのではなく, ただ 1 つ include = FALSE を使うことで出力全体を抑制できます. これが例です.

```
```{r, include=FALSE}
# ここに何らかの R コード
```
```

include=FALSE オプションがあると, eval = FALSE の指定がない限りコードチャンクは評価されますが, 出力は完全に抑制されます. コードも, テキスト出力も, メッセージもグラフも見えなくなります.

11.9 テキスト出力をソースコードとまとめる

テキスト出力ブロックとソースコードブロックの間隔が空きすぎていると感じたら, チャンクオプション collapse = TRUE でテキスト出力をソースブロックと連結することを検討するとよいでしょう. collapse = TRUE としたとき, 出力はこのようになります.

```
1 + 1
## [1] 2
1:10
##  [1]  1  2  3  4  5  6  7  8  9 10
```

以下は同じチャンクですが collapse = TRUE オプションがありません. デフォルトは FALSE です.

```
1 + 1
```

```
## [1] 2
```

```
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

11.10　R のソースコードを整形する

チャンクオプション `tidy = TRUE` を設定すると，**formatR** パッケージ (Xie, 2021b) の `tidy_source()` 関数によって R のソースコードが整形されます．`tidy_source()` 関数は，ほとんどの演算子の前後にスペースを追加する，適切なインデントをする，代入演算子 `=` を `<-` に置き換えるなど，いくつかの観点でソースコードを整形します．チャンクオプション `tidy.opts` には `formatR::tidy_source()` に与えられる引数のリストが使えます．これが例です．

````
```{r, tidy=TRUE, tidy.opts=list(arrow=TRUE, indent=2)}
# 乱雑な R コード...
1+           1
x=1:10# 代入演算子として '<-' を好むユーザーがいる
if(TRUE){
print('Hello world!') # スペース 2 個でインデントする
}
```
````

整形後の出力はこうなります．

```
01 # 乱雑な R コード...
02 1 + 1
03 x <- 1:10  # 代入演算子として '<-' を好むユーザーがいる
04 if (TRUE) {
05   print("Hello world!")  # スペース 2 個でインデントする
06 }
```

5.3 節ではテキストの幅を制御する方法について言及しました．ソースコードの幅を制御したいなら，`tidy = TRUE` としたときに `width.cutoff` 引数を試してください．これが例です．

````
```{r, tidy=TRUE, tidy.opts=list(width.cutoff=50)}
# 長い式
```
````

```
1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+
1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
```
```

出力はこうなります.

```
# 長い式
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +
  1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +
  1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +
  1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1
```

使用可能な引数を知るにはヘルプページ `?formatR::tidy_source` を読んでください. そして https://yihui.org/formatR/ で使用例とこの関数の限界を理解してください.

`tidy = styler` を設定したなら, コード整形には代わりに **styler** パッケージ (Müller and Walthert, 2021) が使われます. R コードは `styler::style_text()` 関数で整形されます. **styler** パッケージは **formatR** よりも豊富な機能を持ちます. 例えば, 引数のアラインメントができたりパイプ演算子 `%>%` のあるコードも対処できたりします. チャンクオプション `tidy.opts` は `styler::style_text()` へ引数を渡して使うこともできます. これが例です.

````
```{r, tidy='styler', tidy.opts=list(strict=FALSE)}
# 代入演算子のアラインメント
a   <- 1#one variable
abc <- 2#another variable
```
````

デフォルトでは `tidy = FALSE` であり, あなたのコードは整形されません.

11.11 テキストを生の Markdown として出力する (*)

デフォルトではコードチャンクからのテキスト出力は冒頭に 2 つハッシュを置いて, テキストをそのまま書き出します (11.12 節参照). **knitr** は出力をコードブロックで囲むため, テキストはその
````

まま表示されます. 例えば 1:5 というコードの生の出力はこうなります.

```
## [1] 1 2 3 4 5
```

時には生のテキストをそのまま出力するのではなく, Markdown 構文として扱いたいこともあるでしょう. 例えば cat('# **これは見出しです**') でセクション見出しを書きたい時があるかもしれませんが, 生の出力はこうなります.

```
## # これは見出しです
```

テキストをコードブロックで囲んでほしくない, あるいは冒頭のハッシュもいらない. つまり, 生の出力が cat() に与えた文字列そのままになるようにしたい, というわけです.

```
# This is a header
```

これを解決するのはチャンクオプション results = 'asis' です. このオプションは テキスト出力をコードブロックで囲むのではなく, “as is” (そのまま) 扱うよう **knitr** に指示します. R コードから動的にコンテンツを生成したい時に, このオプションは特に有用でしょう. 例えば以下のコードチャンクと results = 'asis' オプションで, iris データから列名のリストを生成します.

```
cat(paste0("- `", names(iris), "`"), sep = "\n")
```

- Sepal.Length
- Sepal.Width
- Petal.Length
- Petal.Width
- Species

ハイフン (-) は番号のない箇条書き意味する Markdown 構文です. バッククォートはオプションです. `results = 'asis'` オプションなしで上記のコードチャンクがそのまま出力されるのを見ることもできます.

```
cat(paste0("- `", names(iris), "`"), sep = "\n")
```

- `Sepal.Length`
- `Sepal.Width`
- `Petal.Length`
- `Petal.Width`
- `Species`

以下は, セクション見出し, パラグラフ, `mtcars` データの全ての列に対して `for` ループ内で作成したグラフを表示する例の全貌です

```
---
title: プログラミングでコンテンツを生成する
---

チャンクオプション `results = 'asis'` で生の Markdown コンテンツを書き出すことができます.
↪ これはプロットを含めることもできます.

```{r, mtcars-plots, results='asis'}
for (i in names(mtcars)) {
  cat('\n\n# 変数 `', i, '` の要約.\n\n')
  x <- mtcars[, i]
  if (length(unique(x)) <= 6) {
    cat('`', i, '` はカテゴリカル変数である.\n\n')
    plot(table(x), xlab = i, ylab = '度数', lwd = 10)
  } else {
    cat('連続変数 `', i, '` のヒストグラム.\n\n')
    hist(x, xlab = i, main = '')
  }
}
```
```

```
```

改行 (\n) を過剰に追加していることに注意してください．これは Markdown コンテンツ上でそれぞれの要素を明確に分離したいからです．要素間の改行は多すぎても無害ですが，改行が不十分だと問題が起こりえます．例えば以下の Markdown テキストには大いに曖昧さがあります．

```
# これは見出し?
これは段落?ヘッダの一部?
![この画像は?](foo.png)
# この行はどうなる?
```

`cat('\n')` で生成できていたように空白行を追加すると，この曖昧さは消えます．

```
# そうこれは見出し!

そしてこれは明らかに段落．

![これは画像](foo.png)

# 完全なる別の見出し
```

`cat()` だけがテキスト出力のできる関数ではありません．他のよく使われる関数には `print()` があります．`print()` はしばしばオブジェクトの表示のために**暗黙に**呼び出されることに注意してください．これが R コンソールでオブジェクトや値をタイプした直後に出力が表示される理由です．例えば R コンソールで `1:5` とタイプし `Enter` キーを押した時，R が実際には `print(1:5)` を暗黙に呼び出しているので出力が見えます．R コンソール上で入力していれば正常に表示されていたはずのオブジェクトや値が `for` ループなどのコード内では出力の生成に失敗するというのはとても混乱をもたらします．この話はかなり技術的に高度なので，私はブログに “The Ghost Printer behind Top-level R Expressions”[*3] という説明の記事を投稿しました．技術的な詳細に関心がないなら，このルールだけは覚えてください．「for ループ内の出力が表示されなかったら，おそらく `print()` 関数で明示的に表示させるべきです」

[*3] https://yihui.org/en/2017/06/top-level-r-expressions/

11.12　テキストの先頭のハッシュ記号を消す

デフォルトでは R コードのテキスト出力の先頭には 2 つのハッシュ記号 ## が付きます．この挙動はチャンクオプション comment で変更することができます．このオプションのデフォルトは "###" という文字列です．ハッシュを消したいなら，空の文字列を使うことができます．これが例です．

```
```{r, comment=""}
1:100
```
```

もちろん，comment = "#>" などと他の文字列はなんでも使うことができます．なぜ comment オプションのデフォルトはハッシュ記号なのか？ その理由は # が R ではコメントを意味するからです．テキスト出力がコメントアウトされていれば，レポートに掲載されたコードチャンクを全部まとめてコピーして自分で実行するのが簡単になり，テキスト出力が R コードとして扱われないということに悩むことがありません．例えば以下のコードチャンクの 4 つの行のテキスト全てをコピーして，R コードとして安全に実行することができます．

```
1 + 1
## [1] 2
2 + 2
## [1] 4
```

comment = "" でハッシュ記号を消したなら，2 つ目と 2 つ目のコードを手動で消さなければならないため，全てのコードをコピーして簡単に実行するということができなくなります．

```
1 + 1
[1] 2
2 + 2
[1] 4
```

comment = "" が好ましいという主張の 1 つには，テキスト出力が R コンソールのユーザーにとっ

て見慣れたものになるという点です．R コンソールではテキスト出力の行の先頭にはハッシュ記号が現れません．本当に R コンソールの挙動を模倣したいのであれば，`comment = ""` を `prompt = TRUE` と組み合わせて使うことができます．これが例です．

````
```{r, comment="", prompt=TRUE}
1 + 1
if (TRUE) {
  2 + 2
}
```
````

ソースコードにプロンプト記号 `>` と行の継続を表す記号 `+` が含まれているので，出力は R コードをタイプして実行するときのものにかなり近づいているはずです．

```
> 1 + 1
[1] 2
> if (TRUE) {
+   2 + 2
+ }
[1] 4
```

11.13　テキスト出力ブロックに属性を与える (*)

7.3 節では，`class.source` と `class.output` を使い，ソース・テキスト出力のブロックにスタイルを定義する例をいくつかお見せしました．実際には **knitr** には同様の様々なオプションがあります．それらは `class.message`, `class.warning`, `class.error` といったものです．これらのオプションはクラス名を対応するテキスト出力ブロックに追加するために使うことができます．例えば `class.error` はチャンクオプション `error = TRUE` (11.2 節参照) が設定されているとき，エラーメッセージに対してクラスを追加します．これらのオプションがもっともよく使われるのは，クラス名に対応して定義された CSS ルールで出力にスタイルを適用するときでしょう．この例の実演は 7.3 節でなされています．

典型的には，テキスト出力ブロックは最低限コードブロックに囲まれており，Markdown のソースはこのようになります．

```{.className}
出力された行
```

出力フォーマットが HTML ならば, たいていの場合で[*4]このように変換されます.

```
<pre class="className">
<code>出力された行</code>
</pre>
```

class.* オプションは <pre> 要素の class 属性を制御します. この要素は先述のテキスト出力ブロックを入れたコンテナです.

実際には, クラスは HTML の <pre> 要素の属性に使用可能なものの 1 つにすぎません. HTML 要素は幅や高さやスタイルなどと, 他にも多くの属性を持ちます. attr.source, attr.output, attr.message, attr.warning, attr.error を含む一連のチャンクオプション attr.* によって, 任意の属性をテキスト出力ブロックに追加することができます. 例えば attr.source = 'style="background: pink;"' を使えばソースブロックの背景をピンク色にできます. 対応するコードブロックはこのようになります.

```{style="background: pink;"}
...
```

そして HTML 出力はこのようになります.

```
<pre style="background: pink;">
...
</pre>
```

[*4] に変換される場合もあります. 万全を期すには HTML 出力された文書を確認することもできます.

5.7, 12.3 節でさらなる例を見ることができます.

技術的なことをいいますと, チャンクオプション `class.*` は `attr.*` の特殊形です. 例えば `class.source = 'numberLines'` は `attr.source = '.numberLines'` と同じです (後者は先頭にドットがあることに注意). しかし `attr.source` は任意の属性をとることができ, 例えば `attr.source = c('.numberLines', 'startFrom="11"')` も可能です.

これらのオプションはほとんどの HTML 出力で有効です. 属性が他の出力フォーマットでも有効な場合もありますが, そのような場合になるのは比較的珍しいです. 属性は Pandoc か, 何らかのサードパーティ製パッケージ のいずれかでサポートされている必要があります. `.numberLines` 属性は Pandoc によって HTML と LaTeX の両方で動作し, サードパーティ製パッケージというのは大抵は 4.20 節で紹介したような Lua フィルターを使ったものになります.

11.14 グラフに後処理をかける (*)

コードチャンクでグラフが生成された後, チャンクオプション `fig.process` によってグラフに後処理をかけることが出来ます. これはファイルパスを引数にとり, 生成された画像ファイルのパスを返す関数であるべきです. この関数はオプションで第 2 引数 `option` を取ることができ, これには現在のチャンクのオプションのリストが与えられます.

R のロゴをグラフに埋め込むために, とても強力な **magick** パッケージ (Ooms, 2021b) を使用する例を以下にお見せします. このパッケージに詳しくないなら, オンラインドキュメントか, 豊富な使用例を含むパッケージのヴィネットをを読むことをお薦めします. 初めに, 関数 `add_logo()` を定義します.

```
add_logo <- function(path, options) {
  # コードチャンクで作成された画像
  img <- magick::image_read(path)
  # R のロゴ
  logo <- file.path(R.home("doc"), "html", "logo.jpg")
  logo <- magick::image_read(logo)
  # デフォルトの重心は `northwest` (左上) で,
  # ユーザーはチャンクオプション `magick.gravity`
  # で変更できる
  if (is.null(g <- options$magick.gravity))
    g <- "northwest"
  # ロゴを画像に追加する
```

```
  img <- magick::image_composite(img, logo, gravity = g)
  # 新しい画像を書き出す
  magick::image_write(img, path)
  path
}
```

基本的にこの関数は R のグラフのパスをとり, R のロゴを追加し, 元画像のパスに新しい画像を保存します. デフォルトでは, ロゴはグラフの左上 (northwest) の隅に追加されますが, ユーザーはカスタムチャンクオプション `magick.gravity` で位置をカスタマイズできます. このオプション名は任意に決められます.

では上記の処理関数を `fig.process = add_logo` と `magick.gravity = "northwest"` オプションで以下のコードチャンクに適用します. よってロゴは右上の隅に追加されます. 実際の出力は図 11.1 になります.

```
par(mar = c(4, 4, 0.1, 0.1))
hist(faithful$eruptions, breaks = 30, main = "", col = "gray",
  border = "white")
```

あなたが **magick** パッケージにより詳しくなったら, R のグラフに後処理をするための, より創造的で有用なアイディアを思いつくことでしょう.

最後に, `fig.process` オプションのもう 1 つの応用例をお見せします. 以下の `pdf2png()` 関数は PDF 画像を PNG に変換します. 第 11.15 節ではグラフの生成のために `tikz` グラフィックデバイスを使用する例を見せました. この方法の問題は, デバイスが PDF を生成するため, LaTeX 以外の出力文書に対しては機能しないということです. チャンクオプション `dev = "tikz"` と `fig.process = pdf2png` を使えば, グラフの PNG 版を図 11.2 に示すことができます.

```
pdf2png <- function(path) {
  # LaTeX でない出力に対してのみ変換する
  if (knitr::is_latex_output())
    return(path)
  path2 <- xfun::with_ext(path, "png")
  img <- magick::image_read_pdf(path)
```

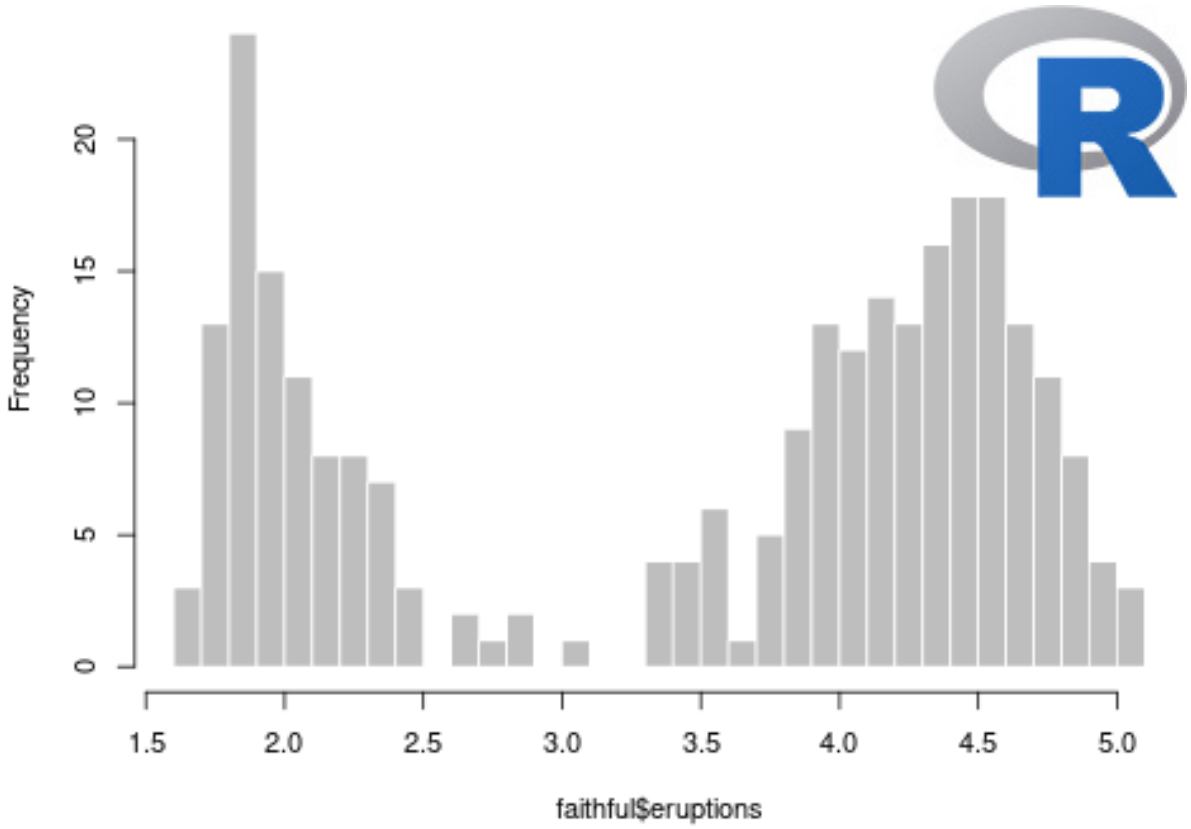


図 11.1: チャンクオプション fig.process でグラフに R のロゴを追加する

```
07    magick::image_write(img, path2, format = "png")
08    path2
09  }
```

11.15 高品質なグラフィック (*)

rmarkdown パッケージはそれぞれの出力フォーマットに対して妥当なデフォルトのグラフィックデバイスを設定しています．例えば HTML 出力に対しては `png()` を使うので，**knitr** は PNG 画像ファイルを生成し，PDF 出力に対しては `pdf()` デバイスを使う，などです．あなたがデフォルトのグラフィックデバイスの品質に不満なら，チャンクオプション `dev` によって変更することができます．**knitr** によってサポートされているグラフィックデバイスの一覧は次のようになります．`"bmp"`, `"postscript"`, `"pdf"`, `"png"`, `"svg"`, `"jpeg"`, `"pictex"`, `"tiff"`, `"win.metafile"`, `"cairo_pdf"`, `"cairo_ps"`, `"quartz_pdf"`, `"quartz_png"`, `"quartz_jpeg"`,

"quartz_tiff", "quartz_gif", "quartz_psd", "quartz_bmp", "CairoJPEG", "CairoPNG", "CairoPS", "CairoPDF", "CairoSVG", "CairoTIFF", "Cairo_pdf", "Cairo_png", "Cairo_ps", "Cairo_svg", "svglite", "ragg_png", "tikz"

大抵の場合, グラフィックデバイスの名前は関数名でもあります. デバイスについてもっと詳しく知りたいなら, あなたは R のヘルプページを読むことができます. 例えば R コンソールで `?svg` と打てば `svg` デバイスの詳細を知ることができます. このデバイスは base R に含まれています. さらに `quartz_XXX` デバイスは `quartz()` 関数を元にしたもので, macOS でのみ有効です. `CairoXXX` デバイスは **Cairo** (Urbanek and Horner, 2020) パッケージによるアドオンで, `Cairo_XXX` デバイスは **cairoDevice** package (Lawrence, 2020) から[*5], `svglite` デバイスは **svglite** パッケージ (Wickham Henry, et al., 2021) から, `tikz` は **tikzDevice** パッケージ (Sharpsteen and Bracken, 2020) からのデバイスです. アドオンパッケージ由来のデバイスを使いたいなら, そのパッケージをまずインストールしなければなりません.

大抵はベクタ画像はラスタ画像よりも高品質であり, ベクタ画像は品質を損なうことなく縮尺を変更できます. HTML 出力では, SVG のグラフのために `dev = "svg"` または `dev = "svglite"` を使うことを検討してください. SVG はベクタ画像形式で, デフォルトの `png` デバイスはラスタ画像形式であることに注意してください.

あなたが PDF 出力時のグラフ内の書体に対してこだわりが強い人なら, `dev = "tikz"` を使うこともできます. これは LaTeX がネイティヴでサポートしているからです. つまり, テキストや記号を含むグラフの全ての要素が LaTeX を介して高品質にレンダリングされるということです. 図 11.2 に, `dev = "tikz"` で R のグラフ内で LaTeX 数式表現を書く例を示します.

```
01 par(mar = c(4, 4, 2, .1))
02 curve(dnorm, -3, 3, xlab = '$x$', ylab = '$\\phi(x)$',
03       main = 'The density function of $N(0, 1)$')
04 text(-1, .2, cex = 3, col = 'blue',
05   '$\\phi(x)=\\frac{1}{\\sqrt{2\\pi}}e^{\\frac{-x^2}{2}}$')
```

base R は実は数式表現をサポートしていますが, LaTeX を介してレンダリングされていないことに注意してください (詳細は `?plotmath` を見てください). `tikz` デバイスの細かい組版を調整するいくつかの発展的なオプションがあります. `?tikzDevice::tikz` で, できることを確認できます. 例えばグラフにマルチバイト文字が含まれているなら, 次のオプションを設定するといいでしょう.

[*5] **訳注**: 名前のよく似た `cairo_pdf` は base R に含まれていることに注意してください.

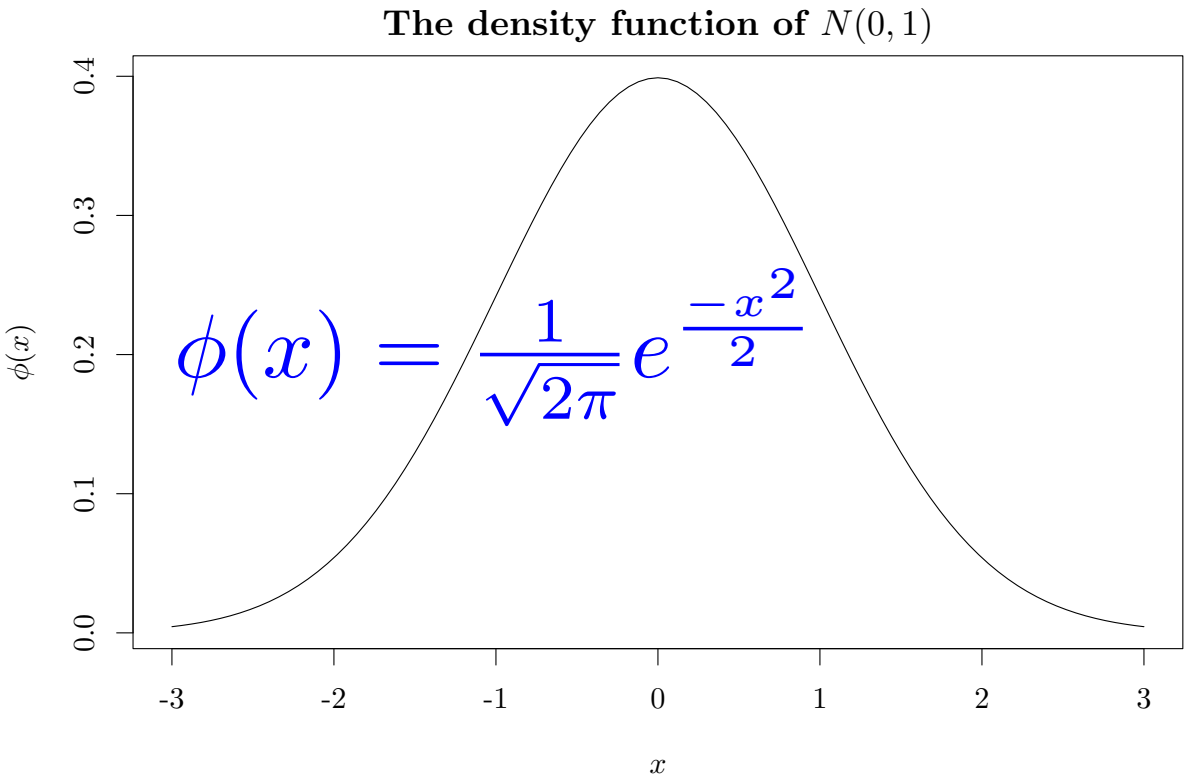


図 11.2: tikz デバイスでレンダリングされたグラフ

```
01 options(tikzDefaultEngine = "xetex")
```

これは, LaTeX 文書でマルチバイト文字を処理する観点では, xetex の方が大抵の場合はデフォルトのエンジン pdftex より優れているからです.

tikz の主な欠点が 2 つあります. 1 つ目は LaTeX のインストールが必要ということですが, これはそこまで深刻ではありません (1.2 節参照). 他にもいくつかの LaTeX パッケージが必要になりますが, TinyTeX を使用しているなら簡単にインストールできます.

```
01 tinytex::tlmgr_install(c("pgf", "preview", "xcolor"))
```

2 つ目の欠点は, デバイスが LaTeX ファイルを生成してから PDF にコンパイルするため, グラフのレンダリングが顕著に遅くなるということです. コードチャンクに時間がかかると感じるなら, チャンクオプション cache = TRUE でキャッシュを有効にすることもできます (第 11.4 節参照).

図 11.2 には, チャンクオプション `fig.process = pdf2png` が使われています. `pdf2png` は 11.14 節で定義された, 出力フォーマットが LaTeX でない時に PDF 画像を PNG に変換するものです. 変換しない場合, 本書のオンライン版をウエブブラウザで閲覧しても PDF グラフは見られないでしょう.

11.16 低水準作図関数で 1 つづつグラフを作る (*)

R グラフィックスには 2 種類の作図関数があります. 高水準作図関数は新たなグラフを作成し, 低水準作図関数は既存のグラフに要素を追加します. 詳細は R マニュアルの 12 章 *An Introduction to R*[*6] をご覧ください.

knitr はデフォルトで, 低水準作図関数がより前のグラフを次々と修正していく段階ではその途中段階のグラフを表示しません. 全ての低水準作図による変更が反映された最後のグラフのみが表示されます.

特に教育目的では, 中間グラフを表示することが有用になりえます. チャンクオプション `fig.keep = 'low'` を設定すれば, 低水準作図による変更を保存できます. 例えば 図 11.3, 11.4 は `fig.keep = 'low'` のオプションを設定した単一のコードチャンクからできたものですが, 2 つのコードチャンクから生成されたように見えます. また, チャンクオプション `fig.cap = c('... の散布図', '... に回帰直線を追加')` で異なる図のキャプションを割り当てています.

```
01 par(mar = c(4, 4, 0.1, 0.1))
02 plot(cars)
```

```
01 fit <- lm(dist ~ speed, data = cars)
02 abline(fit)
```

異なるコードチャンク間でグラフの変更を維持したいなら, 第 14.5 節を参照してください.

[*6] https://cran.r-project.org/doc/manuals/r-release/R-intro.html

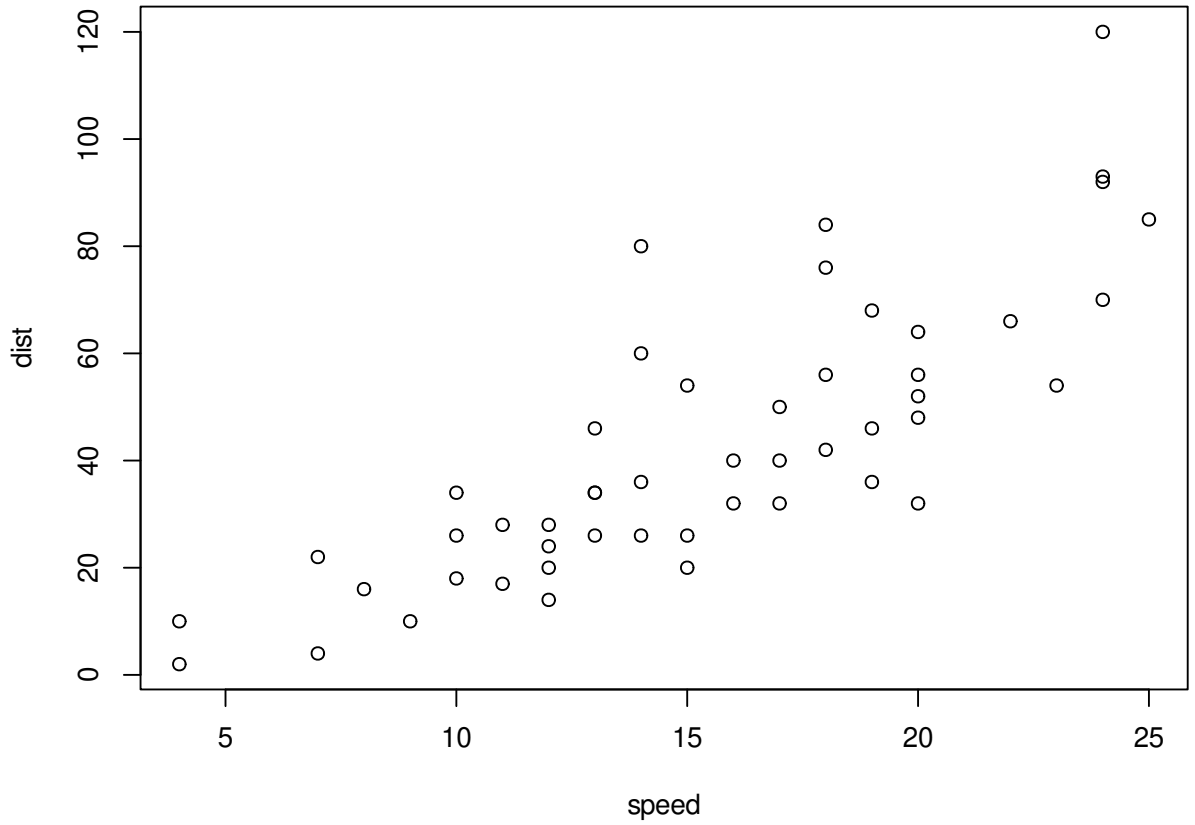


図 11.3: cars データの散布図.

11.17 チャンク内のオブジェクト表示をカスタマイズする (*)

デフォルトではコードチャンク内のオブジェクトは `knitr::knit_print()` 関数を通して表示され, これは概ね base R の `print()` と同じです. `knit_print()` 関数は S3 ジェネリック関数であり, 自分で S3 メソッドを登録して機能を拡張できることを意味します. 以下は `knitr::kable()` でデータフレームを表として自動表示する方法の例です.

```
---
title: データフレームの表示にカスタム `knit_print` メソッドを使う
---

初めに `knit_print` メソッドを定義して登録します.
```

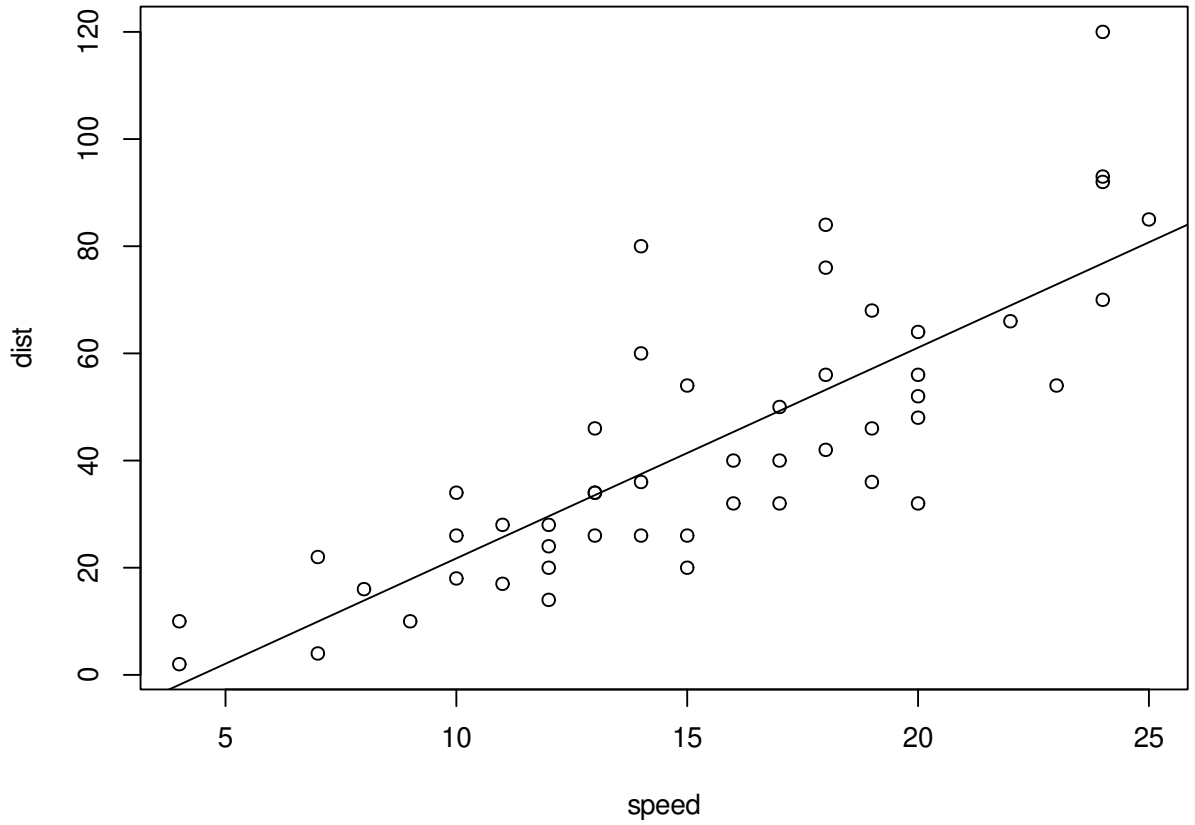


図 11.4: 既にある散布図に回帰曲線を追加

````
```{r}
knit_print.data.frame = function(x, ...) {
  res = paste(c("", "", knitr::kable(x)), collapse = "\n")
  knitr::asis_output(res)
}

registerS3method(
  "knit_print", "data.frame", knit_print.data.frame,
  envir = asNamespace("knitr")
)
```

これでデータフレームに対するカスタム表示メソッドをテストできます．もはや
↪ `knitr::kable()` を明示的に呼ぶ必要がないことに注意してください．
````

````
```{r}
head(iris)
```

```{r}
head(mtcars)
```
````

knit_print() 関数の詳細は **knitr** パッケージのビネットから学ぶことができます.

```
vignette("knit_print", package = "knitr")
```

printr パッケージ (Xie, 2021d) はいくつかの R オブジェクトを可能な範囲で自動的に表として表示する S3 メソッドをいくつか提供します. コードチャンクで library(printr) を実行するだけで十分で, 全てのメソッドが自動的に登録されます.

このテクニックがとても上級者向けだと感じたなら, html_document や pdf_document のような R Markdown の出力フォーマットに df_print オプションを与えてください. これでデータフレームの表示に関する挙動をカスタマイズできます. 例えばデータフレームを knitr::kable() で表示したいなら, このようにオプションを設定することもできます.

```
---
output:
  html_document:
    df_print: kable
---
```

出力フォーマット (?rmarkdown::html_document など) が df_print をサポートするかどうか, そこで使用可能な値が何であるかの判断は, 出力フォーマット関数のヘルプページを見てください.

実際には, render チャンクオプションで knit_print() 関数を完全に置き換えることができます. このオプションはオブジェクトを表示する任意の関数を取ることができます. 例えば **pander** パッケージ を使ってオブジェクトを表示したいなら, チャンクオプション render に pander::pander() を設定するとよいでしょう.

```
```{r, render=pander::pander}
iris
```
```

render オプションによって, あなたは R オブジェクトの表示方法に対する完全なる自由を手に入れるでしょう.

11.18 オプションフック (*)

あるチャンクオプションを, 他のチャンクオプションの値に応じて動的に変えたいことがあるかもしれません. これは, opts_hooks オブジェクトを使って**オプションフック**を設定すればできます. オプションフックはオプションと関連付けられた関数で, 対応するチャンクオプションが NULL でないときに実行されます. この関数は入力引数として現在のチャンクのオプションのリストを受け取り, そのリストを (変更も加えて) 返します. 例えば fig.width オプションを常に fig.height より小さくならないように調整することができます.

```
knitr::opts_hooks$set(fig.width = function(options) {
  if (options$fig.width < options$fig.height) {
    options$fig.width <- options$fig.height
  }
  options
})
```

fig.width が NULL になることはないので, このフック関数は必ずコードチャンクの直前に実行され, チャンクオプションを更新します. 以下のコードチャンクは, 上記のオプションフックが設定されていれば, fig.width が初期値の 5 の代わりに実際には 6 になります.

```
```{r fig.width = 5, fig.height = 6}
plot(1:10)
```
```

別の例として, 第 11.12 説の最後の例を書き換えて, 単一のチャンクオプション console = TRUE を

用いて comment = "" と prompt = TRUE を意味できるようにしました．console は **knitr** の固有のチャンクオプションでなく，任意の名前のカスタムオプションであることに注意してください．デフォルト値は NULL です．以下はその完全な例です．

````
```{r, include=FALSE}
knitr::opts_hooks$set(console = function(options) {
  if (isTRUE(options$console)) {
    options$comment <- ''; options$prompt <- TRUE
  }
  options
})
```
````

デフォルトの出力．

````
```{r}
1 + 1
if (TRUE) {
  2 + 2
}
```
````

`console = TRUE` で出力．

````
```{r, console=TRUE}
1 + 1
if (TRUE) {
  2 + 2
}
```
````

3 つ目の例はどうやって自動的にソースコード・テキスト出力・メッセージ・警告・エラーの出力ブロックに行番号を追加するかに関するものです．行番号を追加するために attr.source, attr.output といったチャンクオプションを使用する方法は 5.7 節で紹介しています．ここでは単一のチャンクオプション (この例では numberLines) で行番号を追加するかどうかを制御したいと

します.

```
knitr::opts_hooks$set(
  numberLines = function(options) {
    attrs <- paste0("attr.", options$numberLines)
    options[attrs] <- lapply(options[attrs], c, ".numberLines")
    options
  }
)

knitr::opts_chunk$set(
  numberLines = c(
    "source", "output", "message", "warning", "error"
  )
)
```

基本的に, オプションフック numberLines は .numberLines 属性を出力ブロックに追加します. opts_chunk$set() によってチャンクオプション numberLunes が設定されオプションフックが確実に実行されます.

上記の設定では, チャンクオプション numberLines をコードチャンクで使用して, そのチャンクの出力ブロックのどの部分に行番号を付けるかを決めることができます. 例えば numberLines = c('source', 'output') のように. numberLines = NULL は行番号を完全に削除します.

このアプローチはチャンクオプションを直接設定するのと何が違うのでしょうか. 例えば第 5.7 節のように, 単に knitr::opts_chunk$set(attr.source = '.numberLines') とする場合と違って, ここでオプションフックを使う利点は .numberLines 属性をチャンクオプションに**追加する**というのみですが, これは既にあるチャンクオプションの値を**上書きする**ことを意味しません. 例えば以下のチャンクのソースコードブロックは (既に設定してある) 行番号が付いていますが, 番号付を 2 行目から始めます.

````
```{r, attr.source='startFrom="2"'}
# このコメント行には番号がつかない
1 + 1
```
````

これは以下と同等です.

````
```{r, attr.source=c('startFrom="2"', '.numberLines'}
# このコメント行には番号がつかない
1 + 1
```
````

第12章

出力フック (*)

knitr パッケージによって, あなたはコードチャンクから出力されるものを各パーツ, ソースコード・テキスト出力・メッセージ・グラフといったものごとに制御しています. この制御は「出力フック」(output hook(s)) によって実現されています. 出力フックは出力の各パーツを入力 (典型的には文字列ベクトルとして扱います) とし, 文字列を出力文書に書き出すために返す一連の関数です. 現時点ではこのしくみを理解するのは簡単ではないでしょうが, これから説明する簡単な例を見ればこのアイディアがはっきりと理解できるものと思います. この例ではコードチャンクの出力が **knitr** の出力フックを介してレンダリングされる様子を表しています.

このような 1 行だけのコードチャンクについて考えてみてください.

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

knitr がコードチャンクを評価すると, 2 つの出力要素を得ます. 2 つとも文字列ベクトルとして保持されます. ソースコードの `"1 + 1"` と, テキスト出力の `"[1] 2"` です. これらの文字列は求められている出力フォーマットに応じて, チャンクフックによってさらなる処理がなされます. たとえば Markdown 文書では **knitr** はソースコードに言語名を付けてコードブロックで囲みます. これは `source` フックを介して行われ, だいたいこのような関数となります.

```
# 上記のケースでは, `x` は文字列 '1 + 1' に相当
function(x, options) {
  # ここの小文字 'r' は言語名を表す
```

```
  paste(c("```r", x, "```"), collapse = "\n")
}
```

同様に, テキスト出力はこのような output フック関数によって処理されます.

```
function(x, options) {
  paste(c("```", x, "```"), collapse = "\n")
}
```

よって上記のコードチャンクの最終的な出力はこのようになります.

````
```r
1 + 1
```

```
[1] 2
```
````

実際のフックは上記のような 2 つの関数よりも複雑ですが, 発想は同じです. knit_hooks オブジェクトから get() メソッドで実際のフック関数を取得できます. これが例です.

```
# 意味のある出力のため, 以下のコードは knitr
# 文書のコードチャンクの *内部で* 実行されるべき
knitr::knit_hooks$get("source")
knitr::knit_hooks$get("output")
# または knitr::knit_hooks$get(c('source', 'output'))
```

knitr パッケージの開発に貢献することに本当に関心がない方には, 組み込みのフック関数のソースコードを読むことをお薦めしません. 関心がある方は, このコードは https://github.com/yihui/knitr/tree/master/R にある hooks-*.R という形式で命名されたスクリプトファイルでご覧くださ

い．例えば hooks-md.R には R Markdown 文書に対するフックが含まれています．普通であれば **knitr** ユーザーにとっては，組み込みのフックを活かして使うカスタム出力フックの作り方を知っていれば十分です．この章ではそれが学べますし，以下では基本的な考え方を示します．

カスタム出力フックは knit_hooks の set() メソッドによって登録されます．このメソッドは既存のデフォルトのフックを上書きするので，既存のフックのコピーを保存し，好きなように出力要素を処理して，その結果をこのデフォルトのフックに与えるようにしておくことをお薦めします．構文はたいてい次のようになります．

```
# ここで local() を使うかは任意（ここでは単に
# `hook_old`
# のような不要なグローバル変数を作ることを避ける目的）
local({
  hook_old <- knitr::knit_hooks$get("NAME")  # 古いフックを保存する
  knitr::knit_hooks$set(NAME = function(x, options) {
    # ここで，x に何らかの処理を行い，それから 新しい
    # x を古いフックに与える
    hook_old(x, options)
  })
})
```

ここで，NAME は以下のいずれかのフックの名前を意味します．

- source: ソースコードを処理するフック．
- output: テキスト出力を処理するフック．
- warning: 警告 (たいていは warning() で発生するもの) を処理するフック．
- message: メッセージ (たいていは message() で発生するもの) を処理するフック．
- error: エラーメッセージ (たいていは stop() で発生するもの) を処理するフック．
- plot: グラフのファイルパスを処理するフック．
- inline: インライン R コードからの出力を処理するフック．
- chunk: チャンク全体の出力を処理するフック．
- document: 文書全体を処理するフック．

フック関数の引数 x の意味は上記のリストで説明されています．options 引数は現在のコードチャンクのオプションのリストを意味します．例えば foo = TRUE と設定したなら，フック関数内では options$foo でこの値を得ることができます．options 引数は inline および document フックでは

利用できません.

出力フックによって, チャンクと文書の出力の部品 1 つ 1 つに対して究極のコントロールを得られます. あらかじめ定義された目的を持つチャンクオプションと比較すると, 出力フックはユーザー定義関数なのではるかに強力であり, 関数内ではあなたが望むことはなんでもできます.

12.1 ソースコードを検閲する

ときにはレポートにソースコードの全文を掲載したくないこともあるでしょう. 例えばコードのある行にパスワードが書かれているかもしれません. 11.7 節ではチャンクオプション `echo` で R コードの文ごとに表示の有無を選べることを言及しました. 例えば `echo = 2` で 2 つ目の文を表示します. この節では, コードのインデックスを指定する必要のない, より柔軟な方法を提供します.

基本的なアイディアはコードに特殊なコメント, 例えば `# 秘密!!` のようなものを追加するということです. このコメントがコードのある行から検出されると, 行を省略します. 以下は `source` フックを使用した完全な例です.

````
---
title: "`source` フックを使用してコードのある行を隠す"
---

初めに，末尾に `# 秘密!!` という文字列を含むコードの行を排除する `source` フックを用意します.

```{r, include=FALSE}
local({
  hook_source <- knitr::knit_hooks$get('source')
  knitr::knit_hooks$set(source = function(x, options) {
    x <- x[!grepl('# 秘密!!$', x)]
    hook_source(x, options)
  })
})
```

これで新しいフックをテストできます．この文書を `knit` すると，特殊なコメント `# 秘密!!` のある行が見えなくなります.
````

```
```{r}
1 + 1  # 表示されるべき通常のコード

# 実際のユーザー名とパスワードを使ってみてください
auth <- httr::authenticate("user", "passwd")
auth <- httr::authenticate("yihui", "horsebattery")  # 秘密!!
httr::GET("http://httpbin.org/basic-auth/user/passwd", auth)
```
```

上記の source フックの重要な部分はこの行です．これはソースコードの入ったベクトル x から grepl() で追跡用のコメントの # **秘密**!! とマッチングしたものを排除しています．

```
x <- x[!grepl("# SECRET!!$", x)]
```

正確に言うなら，上記のフックは追跡用の # **秘密**!! というコメントのある行ではなく，**評価式** (expressions) を全て排除します．x は実際には R 評価式のベクトルだからです．例えば以下のコードチャンクを考えます．

```
1 + 1
if (TRUE) {
  # SECRET!!
  1:10
}
```

source フック内の x の値はこうなります．

```
c("1 + 1", "if (TRUE) { # SECRET!!\n  1:10\n}")
```

R 評価式ではなく行単位で隠したいなら，x を行ごとに分割しなければなりません．xfun::split_lines() の使用を検討するとよいでしょう． フック関数の本体はこうなり

ます.

```
x <- xfun::split_lines(x)  # 個別の行に分割する
x <- x[!grepl("# SECRET!!$", x)]
x <- paste(x, collapse = "\n")  # 結合して 1 つの行にする
hook_source(x, options)
```

この例はソースコードの文字列を操作する方法を, そして `grepl()` は決して文字列操作の唯一の方法ではない, ということを示しています. 12.2 節では他の例もお見せしています.

12.2 ソースコード内に行番号を追加する

この節では, ソースコードに行番号をコメントとして追加する `source` フックの定義の例を示します. 例えば, このコードチャンクに対するものを考えます.

````
```{r}
if (TRUE) {
  x <- 1:10
  x + 1
}
```
````

このような出力を求めているものとします.

```
if (TRUE) {    # 1
  x <- 1:10    # 2
  x + 1        # 3
}              # 4
```

完全な例は以下になります.

````
---
title: ソースコードに行番号を追加する
---

行番号をソースコードに追加する `source` フックを用意します．
番号は各行の末尾のコメントに現れます．

```{r, include=FALSE}
local({
  hook_source <- knitr::knit_hooks$get('source')
  knitr::knit_hooks$set(source = function(x, options) {
    x <- xfun::split_lines(x)
    n <- nchar(x, 'width')
    i <- seq_along(x)  # 行番号
    n <- n + nchar(i)
    s <- knitr:::v_spaces(max(n) - n)
    x <- paste(x, s, '  # ', i, sep = '', collapse = '\n')
    hook_source(x, options)
  })
})
```

ここで新しいフックのテストができます．この文書を knit するとき，
末尾のコメントに行番号が見られます．

```{r}
if (TRUE) {
  x <- 1:10
  x + 1
}
```
````

上記の例での主な小ワザは各行のコメントの前に必要なスペースの数を決めることです．これによってコメントが右揃えになっています．この数は各行のコードの文字列幅に依存しています．このフック関数の意味を咀嚼することは読者に任せます．内部で使われている関数

knitr:::v_spaces() は特定の長さのスペースを生成することに使われている点に注意してください．これが例です．

```
knitr:::v_spaces(c(1, 3, 6, 0))
```

```
## [1] " "      "   "    "      " ""
```

5.7 節で紹介した方法が，ソースコードに行番号を追加する方法としてあなたが本当に求めていたものかもしれません．その構文はより簡潔で，ソースコードでもテキスト出力ブロックでも動作します．上記の source フックの小ワザは，カスタム関数でソースコードを操作できることの一例を示すのが主な狙いです．

12.3 スクロール可能なテキスト出力

7.4 節ではコードブロックとテキスト出力ブロックの高さを CSS で制御する方法を紹介しました．実際には，もっと簡単なやり方で，チャンクオプション attr.source と attr.output を使って style 属性を Markdown のコードブロックに追加できます (これらのオプションの説明は 11.13 節参照)．例えば，このようなコードに対してチャンクオプション attr.output を使います．

````
```{r, attr.output='style="max-height: 100px;"'}
1:300
```
````

Markdown 出力はこうなります．

````
```r
1:300
```


```{style="max-height: 100px;"}
##   [1]   1   2   3   4   5   6   7   8   9  10
##  [11]  11  12  13  14  15  16  17  18  19  20
##  ... ...
````
````

```
```

そして, テキスト出力ブロックは Pandoc によって HTML へと変換されます.

```
<pre style="max-height: 100px;">
<code>##   [1]   1   2   3   4   5   6   7   8   9  10
##  [11]  11  12  13  14  15  16  17  18  19  20
##  ... ...</code>
</pre>
```

Pandoc の fenced code blocks についてより詳しく学ぶには, https://pandoc.org/MANUAL.html#fenced-code-blocks のマニュアルを読んでください.

attr.source と attr.output オプションによって個別のコードチャンクに対して最大の高さを指定することができます. しかしこの構文は少しばかり野暮ったく, CSS と Pandoc の Markdown 構文をより理解する必要があります. 以下にカスタムチャンクオプション max.height と連動するカスタム output フックの例を示します. attr.output = 'style="max-height: 100px;"' の代わりに max.height = "100px" のようにオプションを設定するだけでよいのです. この例では x 引数には手を付けず, options 引数のみを操作しています.

```
---
title: スクロール可能なコードブロック
output:
  html_document:
    highlight: tango
---

チャンクオプション `max.height` が設定されている時，テキスト出力に `style` 属性を追加する
↪ ような `output` フックを設定します.

```{r, include=FALSE}
options(width = 60)
local({
```
```

````
  hook_output <- knitr::knit_hooks$get('output')
  knitr::knit_hooks$set(output = function(x, options) {
    if (!is.null(options$max.height)) options$attr.output <- c(
      options$attr.output,
      sprintf('style="max-height: %s;"', options$max.height)
    )
    hook_output(x, options)
  })
})
```

`max.height` がない場合，出力の全体が表示されます．例えば...,

```{r}
1:100
```

ここで `max.height` に `100px` を設定します．この高さは 100px を超えているので，テキスト出力にスクロールバーが現れます．

```{r, max.height='100px'}
1:100
```

原則として `max.height` オプションは `attr.output` オプションに変換されます．`attr.output` が既に設定されていたとしても動作します．つまり `attr.output` オプションは上書きされません．例えば `.numberLines` 属性を付けてテキスト出力の端に行番号を表示させてみます．

```{r, max.height='100px', attr.output='.numberLines'}
1:100
```
````

図 12.1 がその出力です．最後のコードチャンクにあるチャンクオプション attr.output に注意してください．そのオプションは max.height によっては上書きされないのです．なぜなら，max.height
````

が生成する style 属性を使って既存の属性と結合することで, 既存の属性を尊重しているからです.

```
options$attr.output <- c(
  options$attr.output,
  sprintf('style="max-height: %s;"', options$max.height)
)
```

図 12.1: チャンクオプション max.height を指定した, スクロール可能なテキスト出力の例

source フックでも同様の小ワザを使って, ソースコードブロックの高さを制限できます.

12.4 テキスト出力を中断する

コードチャンクから出力されたテキストが長い時, 冒頭の数行だけを表示させたくなります. 例えば数千行のデータフレームを表示する時, データ全体を表示するのは不便で, 最初の数行だけで十

分でしょう．以下では output フックを再定義してカスタムチャンクオプション out.lines によって最大行数を制御できるようにしています．

```
# 組み込みの出力フックを保存
hook_output <- knitr::knit_hooks$get("output")

# テキスト出力を打ち切る出力フックを新規に作成
knitr::knit_hooks$set(output = function(x, options) {
  if (!is.null(n <- options$out.lines)) {
    x <- xfun::split_lines(x)
    if (length(x) > n) {
      # 出力を切断
      x <- c(head(x, n), "....\n")
    }
    x <- paste(x, collapse = "\n")
  }
  hook_output(x, options)
})
```

上記のフック関数の基本的なアイディアはテキスト出力の行数が，チャンクオプション out.lines で指定したしきい値 (関数本体では変数 n として保存されています) を上回ったら，最初の n 行だけを残し，省略記号 (....) を末尾に加え出力が打ち切られたことを表します．

以下のチャンクでチャンクオプション out.lines = 4 を設定し，この新たな output フックをテストできます．

```
print(cars)
```

```
##    speed dist
## 1      4    2
## 2      4   10
## 3      7    4
....
```

期待通りに 4 行の出力が現れました．元の output フックを hook_output に保存しているので，再度

set() メソッドを呼び出して復元することができます.

```
knitr::knit_hooks$set(output = hook_output)
```

読者への練習問題として, 異なる方法で出力を打ち切ることに挑戦してみてください. 最大行を決定するチャンクオプション out.lines を所与として, 末尾の行ではなく中間の行を打ち切ることができますか? 例えば out.lines = 10 なら, このように最初と最後の 5 行を残し, 中間に .... を追加します.

```
##    speed dist
## 1      4    2
## 2      4   10
## 3      7    4
## 4      7   22
....
## 46    24   70
## 47    24   92
## 48    24   93
## 49    24  120
## 50    25   85
```

出力の最終行, つまりフック関数の引数 x が空白行ならば, c(head(x, n/2), '....', tail(x, n/2 + 1)) のような処理が必要であることに注意してください. +1 は最後の空白行を考慮するためです.

12.5 HTML5 フォーマットで図を出力する

デフォルトでは R Markdown のグラフは HTML 上で <p> または タグ内の で読み込まれます. 以下の例は HTML5 の <figure> タグでグラフを表示する方法です.

```
---
title: "`<figure>` タグで図を出力する"
output: html_document
```

````
---

ファイルパス `x` とチャンクオプション `options$fig.cap` の
図のキャプションが与えられた状態で，このようなフォーム内に
HTML5 タグ内にグラフを描きたいとします．

```html
<figure>
  <img src="ファイルパス" alt="キャプション" />
  <figcaption>キャプション</figcaption>
</figure>
```

ここで出力フォーマットが HTML であるときのみ `plot` フックを再定義します．

```{r}
if (knitr::is_html_output()) knitr::knit_hooks$set(
  plot = function(x, options) {
    cap  <- options$fig.cap  # 図のキャプション
    tags <- htmltools::tags
    as.character(tags$figure(
      tags$img(src = x, alt = cap),
      tags$figcaption(cap)
    ))
  }
)
```

以下のコードチャンクから生成されたプロットは `<figure>` タグ内に配置されます．

```{r, fig.cap='cars データの散布図'}
par(mar = c(4.5, 4.5, .2, .2))
plot(cars, pch = 19, col = 'red')
```
````

````
`<figure>` と `<figcaption>` タグの見栄えのために CSS スタイルを追加します．`figure` には
↪ 破線の枠を，キャプションには明桃色の背景を設定します．

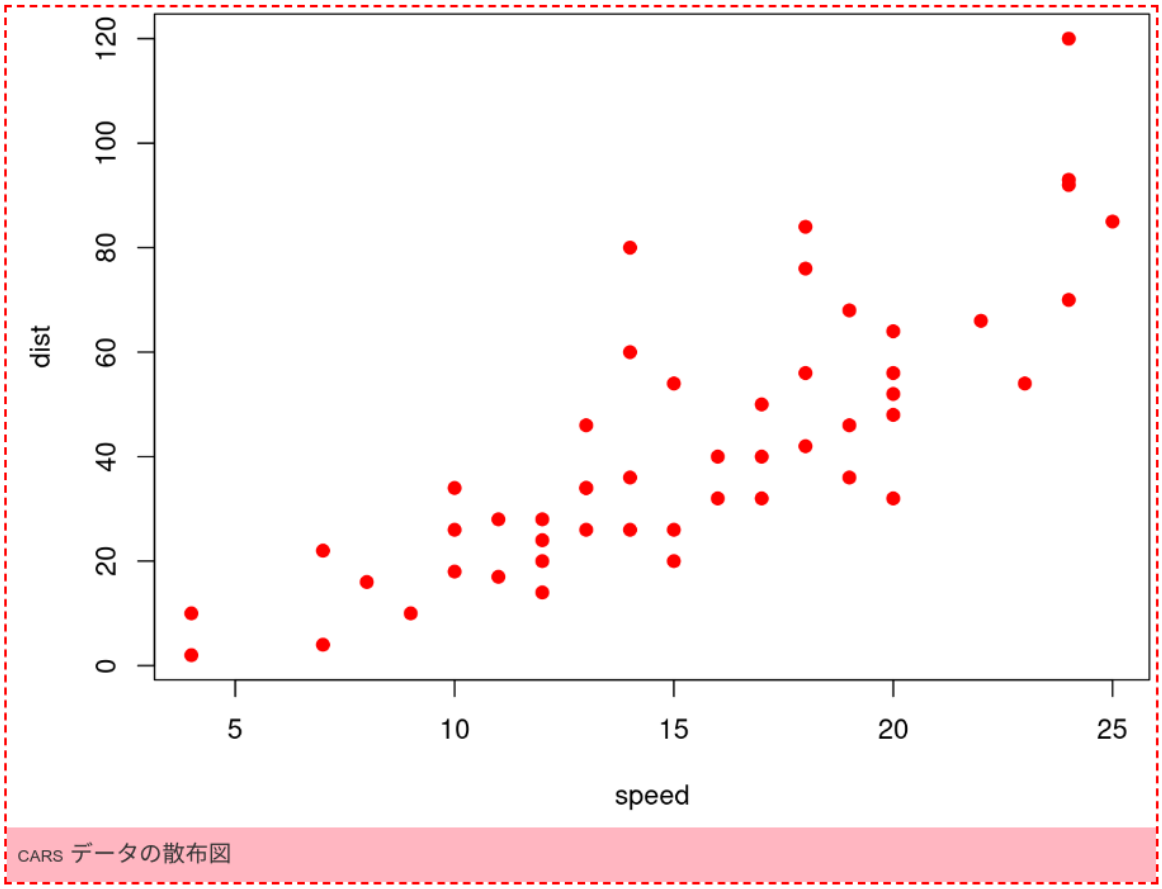
```{css, echo=FALSE}
figure {
  border: 2px dashed red;
  margin: 1em 0;
}
figcaption {
  padding: .5em;
  background: lightpink;
  font-size: 1.3em;
  font-variant: small-caps;
}
```
````

図 12.2 がその出力です．この例では実際には plot フックを上書きしましたが，この章の他のほとんどの例ではデフォルトのフックの冒頭にカスタムフックを構築していることに注意してください．デフォルトのフックを完全に上書きするのは，組み込まれている機能を無視しても構わない時に限るべきです．例えば次の plot フックは out.width ='100%' や fig.show = 'animate' といったチャンクオプションがあるかもしれないことを考慮していません．

この例はファイルパス x と plot フックが活用できそうなことを示すものです．図のスタイルのカスタマイズだけが必要なら，HTML5 タグを使うことはありません．通常であれば，デフォルトの plot フックは以下のような HTML コードに画像を出力します．

```
<div class="figure">
  <img src="ファイルパス" />
  <p class="caption">キャプション</p>
</div>
```

よって div.figure と p.caption に対して CSS ルールを定義するだけで可能となります．



<figure> と <figcaption> タグの見栄えのために CSS スタイルを追加します. figure には破線の枠を, キャプションには明桃色の背景を設定します.

図 12.2: HTML5 figure タグ内の図

第13章

チャンクフック (*)

チャンクフックはあるチャンクオプションの値が NULL ではないときに駆動する関数です. チャンクフックを使うと, チャンク内でコードを実行する以上の追加のタスクを実行することができます. 例えばグラフに後処理をしたり (例えば 13.1 節, 13.2 節) , コードチャンクの実行時間を記録したいときなどです. このようなタスクはレポート内の計算や分析に必須でなくても, 例えばグラフを改良したり最も時間のかかるチャンクを特定したりといった, 他の目的に対しては役に立つででしょう.

例えばコンソールになんらかの情報をただ表示するだけなど, チャンクフックをまったく別の作用のために使うことができますし, あるいは返り値を使うなら, それが文字列であれば出力文書にその値を書き出すこともできます.

出力フック (12 章参照) のように, チャンクフックは knitr::knit_hooks オブジェクトにて登録されます. 出力フックの名前は **knitr** によって予約されているので, カスタムチャンクフックに使ってはならないことに注意してください.

```
01 names(knitr:::.default.hooks)
```

```
##  [1] "source"          "output"
##  [3] "warning"         "message"
##  [5] "error"           "plot"
##  [7] "inline"          "chunk"
##  [9] "text"            "evaluate.inline"
## [11] "evaluate"        "document"
```

チャンクフックは同じ名前のチャンクオプションと関連付けられています. 例えば greet という名

前のチャンクフックを登録できます.

```
knitr::knit_hooks$set(greet = function(before) {
  if (before)
    "Hello!" else "Bye!"
})
```

この後すぐにフック関数の引数について説明します. まずは以下のチャンクでチャンクオプション greet = TRUE を設定してみます.

````
```{r, greet=TRUE}
1 + 1
```
````

するとチャンクの前に “Hello!” という文字が現れ, 以下のチャンクの出力部の後に “Bye!” という文字が現れます. これは両者が文字列だからです.

Hello!

```
1 + 1
```

```
## [1] 2
```

Bye!

チャンクフック関数は before・options・envir・name の 4 つの引数を取ることができます. 言い換えるならこのような形式にすることができます.

```
function(before, options, envir, name) {

}
```

4 つの引数はすべてあってもなくてもかまいません．4 つ, 3 つ, 2 つ, 1 つ, あるいは引数がなくとも可能です．上記の例では before 引数 1 つだけを使っています．これらの引数は以下のような意味があります．

- before: このチャンクが現在, 実行される直前か直後かです．チャンクフックはコードチャンクごとに 2 度実行される, つまり直前に 1 度 hook(before = TRUE) が, 直後に hook(before = FALSE) が実行されることに注意してください．
- options: 現在のコードチャンクのチャンクオプションのリストです．例えば list(fig.width = 5, echo = FALSE, ...) のような値です．
- envir: チャンクフックが評価される環境です．
- name: チャンクフックのトリガーとなるチャンクオプションの名前です．

この章の冒頭で言及したように, チャンクフックの返す値が文字列でなければ無視されなにも起こりませんが, 文字列のときは出力文書に書き出されます．

13.1 グラフをクロップする

チャンクフック knitr::hook_pdfcrop() は PDF やその他の種類の画像ファイルをクロップ, つまりグラフから余分な余白を削除するのに使えます．これを有効にするには, コードチャンク内で knit_hooks$set() を使って対応するチャンクオプションをオンに設定してください．これが例です．

```
knitr::knit_hooks$set(crop = knitr::hook_pdfcrop)
```

そうすると, チャンクオプション crop = TRUE を使ってグラフをクロップできます．

フック関数 hook_pdfcrop() は内部プログラム pdfcrop を呼び出して PDF ファイルをクロップします．このプログラムは通常 LaTeX の配布パッケージ (例えば TeX Live や MikTeX) に同梱されています．システムでこれが使用可能かどうかは次のようにして確認できます．

```
# 返り値が空でないなら使用可能
Sys.which("pdfcrop")
```

```
##                 pdfcrop
## "/home/ks/bin/pdfcrop"
```

LaTeX 配布パッケージの TinyTeX (1.2 節参照) を使っていて, pdfcrop があなたのシステムで利用できないときは, tinytex::tlmgr_install('pdfcrop') でインストールできます.

訳注

knitr::hook_pdfcrop の使用には ghostscript も必要になります. 環境によっては別途, 手動でインストールする必要があるかもしれません.

PNG や JPEG といった PDF でないグラフ画像ファイルに対しては, このフック関数は R パッケージの **magick** (Ooms, 2021b) を呼び出してクロップします. この R パッケージがインストールされているか確かめておきましょう. 図 13.1 はクロップされていないグラフで, 図 13.2 はクロップされた同じグラフです.

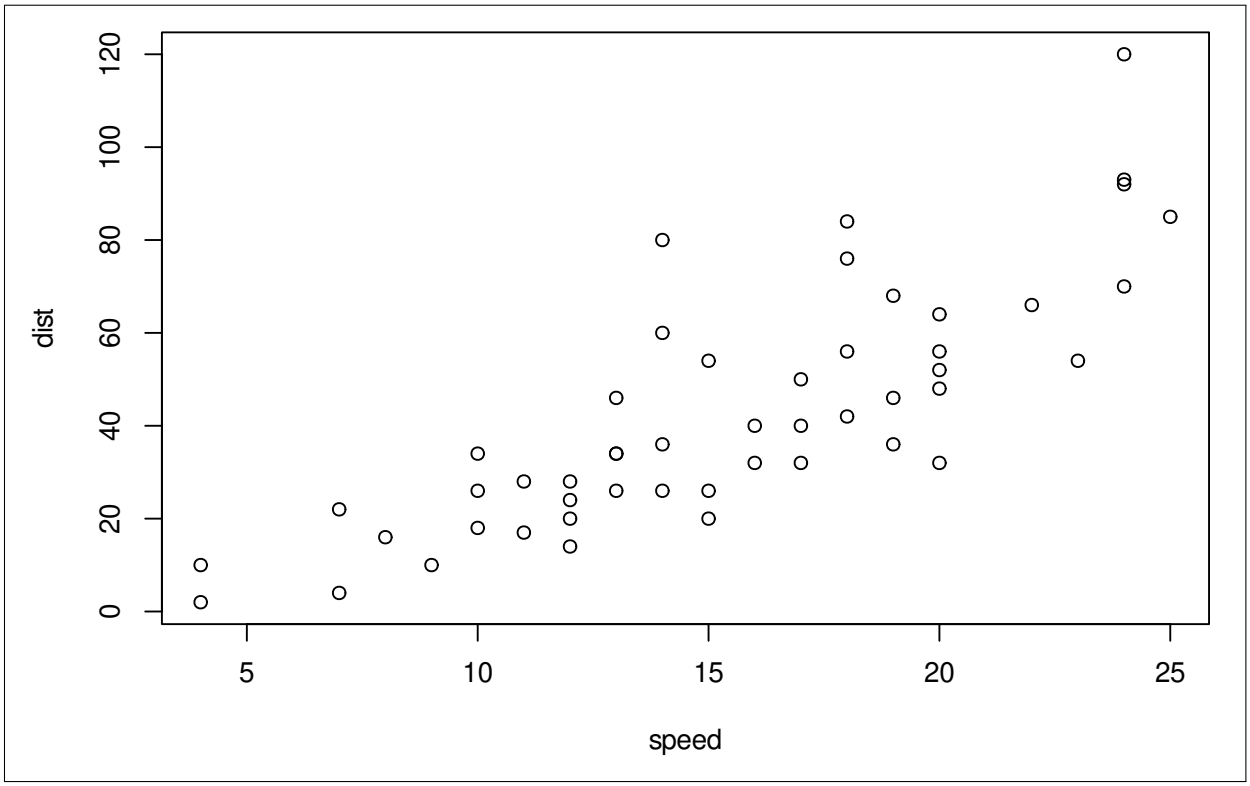


図 13.1: クロップされていないグラフ

13.2 PNG のグラフを最適化する

OptiPNG (http://optipng.sourceforge.net) プログラムをインストールしていれば, knitr::hook_optipng() フックを使って PNG 形式のグラフ画像ファイルの画質を劣化させ

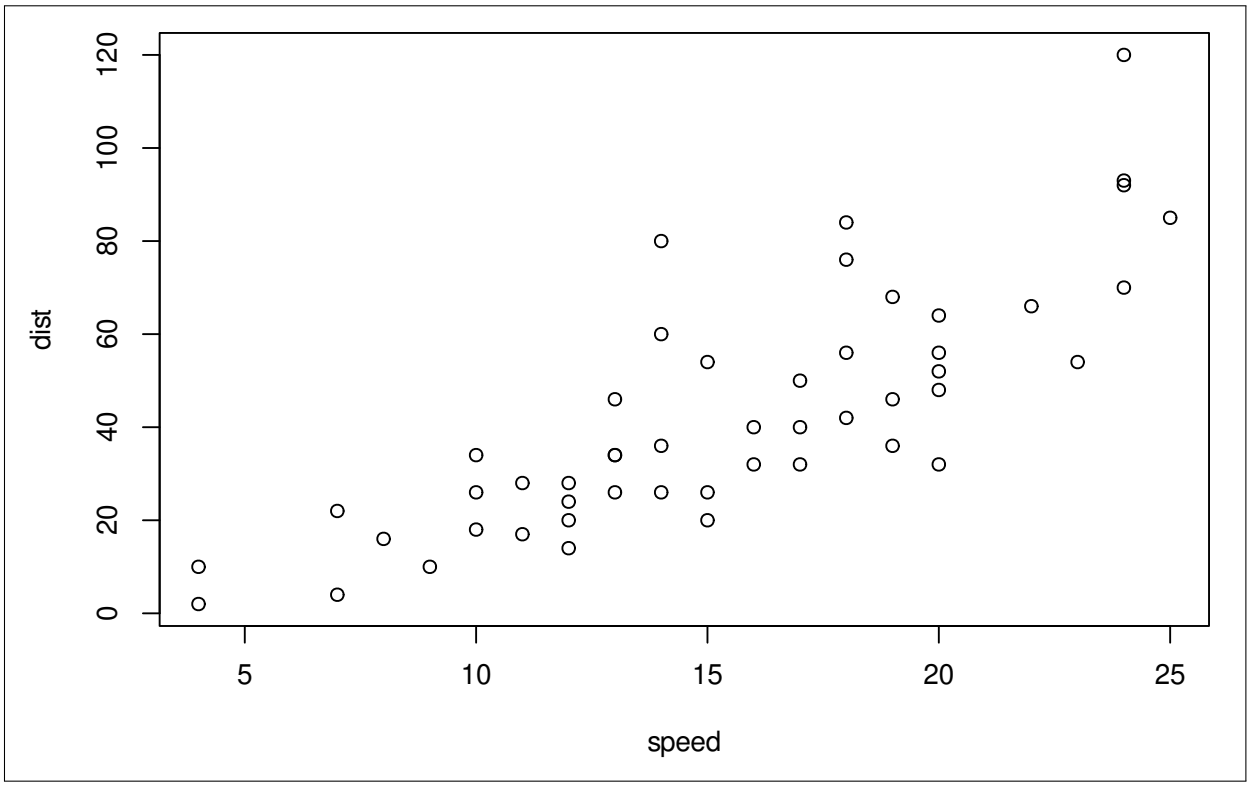


図 13.2: クロップされたグラフ

ることなく縮小して最適化できます.

```
01 knitr::knit_hooks$set(optipng = knitr::hook_optipng)
```

このフックを設定したら, チャンクオプション `optipng` を使い, OptiPNG へのコマンドライン引数を渡すことができます (例えば `optipng = '-o7'`). コマンドライン引数はオプションなので, フックを有効にするためだけに `optipng = ''` と書くことも可能です. 使用可能な引数を知るには OptiPNG のウェブサイト上にあるユーザーマニュアルを見てください.

macOS ユーザーは Homebrew (`https://brew.sh`) で簡単に OptiPNG をインストールできます (`brew install optipng`).

13.3 チャンクの実行時間をレポートする

knitr はデフォルトでは knit 処理中にテキストベースの進捗バーを提供します. より正確なチャンクの時間の情報がほしいなら, カスタムチャンクフックを登録して各チャンクの時間を記録する

こともできます．これはそのようなフックの例です．

```
knitr::knit_hooks$set(time_it = local({
  now <- NULL
  function(before, options) {
    if (before) {
      # 各チャンクの直前の時刻を記録する
      now <<- Sys.time()
    } else {
      # チャンク直後の時刻との差を計算する
      res <- difftime(Sys.time(), now)
      # 時間を表示するための文字列を返す
      paste("Time for this code chunk to run:", res)
    }
  }
}))
```

するとこれ以降のチャンクでは，チャンクオプション `time_it` を使って時間を測定できます．これが例です．

````
```{r, time_it = TRUE}
Sys.sleep(2)
```
````

全てのコードチャンクで時間を表示したいなら，もちろん `knitr::opts_chunk$set(time_it = TRUE)` でグローバルに設定することができます．

上記のフック関数では，さらに詳細な情報をチャンクオプションから出力することもできます．つまりフック関数の `options` 引数を使います．例えば，返り値のチャンクラベルを表示することもできます．

```
paste("Time for the chunk", options$label, "to run:", res)
```

あるいはフック関数で時間を表示させずに記録するだけという手もあります．

```
all_times <- list()  # 全てのチャンクの時間を保存する
knitr::knit_hooks$set(time_it = local({
  now <- NULL
  function(before, options) {
    if (before) {
      now <<- Sys.time()
    } else {
      res <- difftime(Sys.time(), now)
      all_times[[options$label]] <<- res
    }
  }
}))
```

こうすると all_times オブジェクトで全ての実行時間情報にアクセスすることができます. このオブジェクトはチャンクラベルを名前にもつ名前つきリストで, 各要素の値はそれぞれのチャンクの実行時間です.

最後に技術的な注意事項として, 先ほどのフックで使われた local() 関数に詳しくない人もいるかもしれませんので, これについて説明したいとおもいます. この関数でコードを「ローカルな」環境で実行することができます. その主な恩恵は, コード内で作られた変数はこの環境内のローカルなものになるので, 外部の環境, たいていの場合はグローバル環境を汚染することがないということです. 例えばここでは local() 内で now 変数を作成し, これを time_it 内で使用しています. フック関数内では通常の代入演算子 <- の代わりに二重アロー演算子 <<- で now の値を更新しています. <<- は 親環境（ここではあくまでも, local() 環境の内部にある）の変数に代入し, <- は単に現在の環境にのみ値を代入するからというのが理由です. 各コードチャンクが評価される直前に, ローカル変数 now は現在の時刻を記録します. 各コードチャンクが評価されたら現在時刻と now との差を計算します. local() はコード内に渡された最後の値を返しますが, ここではそれがフック関数であることに注意してください. 簡潔に言うなら, local() は, ローカルだけで使われグローバル環境で使われない変数を露出しないことで, ワークスペースをきれいに保つということです. グローバル環境に変数 now が作られても構わなければ, local() を使わないという選択もできます.

13.4 出力にチャンクヘッダを表示する

読者に元のチャンクヘッダのコードを表示したい時もあるかもしれません. 例えば R Markdown のチュートリアルを書いていて, チャンクの出力とその出力を生成するのに使用したチャンクオプ

ションの両方を表示すれば, 読者が自分で同じことをする方法を学ぶことができるというわけです.

元のチャンクオプションは実際にはチャンクオプションの params.scr 内に文字列として保存されています. これを知ったあなたは params.src を出力するチャンクフックを書くこともできます. 以下はその完全な例です.

```
---
title: 出力にチャンクヘッダを表示する
---

本来のチャンクヘッダとフッタの内側にチャンクを出力する
`wrapper` という名前のチャンクフックを用意します.

```{r, setup, include=FALSE}
knitr::knit_hooks$set(wrapper = function(before, options) {
  # 本来のチャンクはインデントされる
  if (is.null(indent <- options$indent)) indent <- ''

  # wrapper=TRUE オプションを隠す
  opts <- gsub(', wrapper=TRUE', '', options$params.src)

  if (before) {
    # ヘッダを追加する
    sprintf('\n\n%s````\n```{r,%s}\n````\n', indent, opts)
  } else {
    # フッタを追加する
    sprintf('\n\n%s````\n```\n````\n', indent)
  }
})
```

ここでチャンクオプション `wrapper=TRUE` でフックを適用します. `wrapper=TRUE` をヘッダの最後に置くことと, 正確に `wrapper=TRUE` でなければならず, 上記で呼び出されている `gsub()` を修正しない限り, 例えば `wrapper=T` はダメで, コンマとスペースの後に続けなければならないことも忘れないでください.

```{r, test-label, collapse=TRUE, wrapper=TRUE}
```
```

````
1 + 1
plot(cars)
```
````

本来のチャンクヘッダが出力に現れるはずです．フックはチャンクがインデントされていても動作するはずです．これが例です．

- **箇条書きその 1**

  ````
  ```{r, eval=TRUE, wrapper=TRUE}
  2 + 2
  ```
  ````

- **もう 1 つ箇条書き**

基本的には, `options$params.src` から取り出したチャンクヘッダを ```{r, } の中に入れることで元のヘッダを再現しています．そしてこの行を 1 組の 4 連続バッククオートで囲んでいるので, 出力時にはそのまま表示されます．本来のコードチャンクはインデントされているかもしれない (例: 箇条書き内にネストされている場合) ので, 適切にインデントを追加することも必要になります．これはチャンクオプション `options$indent` に保存されています．

上記の例の最後の, 箇条書き内の出力はこのようになります．

> - 箇条書きその 1
>
>   ````
>   ```{r, eval=TRUE}
>   ````
>
>   ```
>   2 + 2
>   ```
>
>   ````
>   ## [1] 4
>
>   ```
>   ````
>
> - もう 1 つ箇条書き

コードチャンクが評価され, チャンクヘッダも追加されていることが分かったかと思います．
  ````

13.5　rgl によるインタラクティブな 3 次元グラフを埋め込む

rgl パッケージ (Adler and Murdoch, 2021) を使うとインタラクティブな 3 次元グラフを生成できます．WebGL 形式で保存すれば，これらのグラフは（保存後も）インタラクティブのままです．これはフック関数 `rgl::hook_webgl()` を使えば可能です．以下の例は **rgl** と **knitr** で 3 次元グラフをインタラクティブ性を保ったまま保存できるようにする方法を示しています．

````
---
title: rgl で 3 次元グラフを埋め込む
output: html_document
---

**rgl** を保存するフック関数を用意する．

```{r, setup}
library(rgl)
knitr::knit_hooks$set(webgl = hook_webgl)
```

フックを有効にした後で，チャンクオプション `webgl = TRUE` で
この 3 次元グラフが動作するかを確認してください．

```{r, test-rgl, webgl=TRUE}
x <- sort(rnorm(1000))
y <- rnorm(1000)
z <- rnorm(1000) + atan2(x,y)
plot3d(x, y, z, col = rainbow(1000))
```
````

この例をコンパイルすると図 13.3 のようなインタラクティブな 3 次元散布図が得られるはずです．インタラクティブなグラフは出力フォーマットが HTML の時だけ動作することに注意してください．

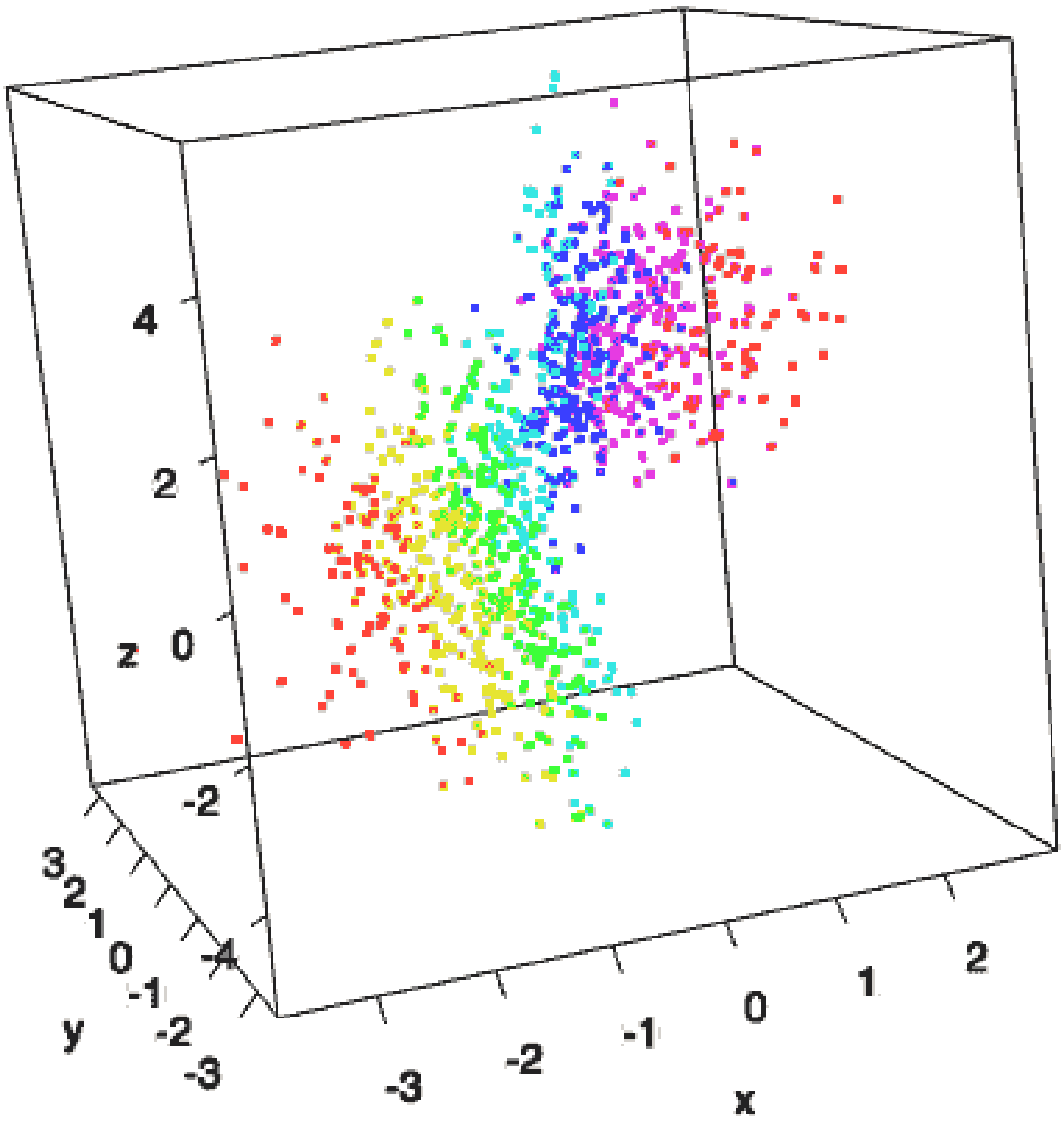


図 13.3: rgl パッケージから生成した 3 次元散布図

第 14 章

その他の knitr の小ワザ

knitr には, チャンクオプション (11 章)・出力フック (12 章)・チャンクフック (13 章) にとどまらず, 他にも役に立つ関数や小ワザがあります. この章では, コードチャンクの再利用, knit を早めに打ち切る方法, グラフの配置場所のカスタマイズの方法などといった小ワザを紹介します.

14.1 コードチャンクを再利用する

コードチャンクは, コピーアンドペーストなしで文書のどの場所でも自由に再利用できます. ポイントはコードチャンクにラベルを付けることで, そうすると他の場所でラベルによって参照することができます. コードチャンクの再利用には 3 種類の方法があります.

14.1.1 チャンクを別の場所にも埋め込む (*)

あるコードチャンクは, チャンクのラベル名を <<>> で囲んで別のコードチャンクに埋め込むことができます. すると **knitr** は自動的に <<**ラベル**>> を実際のコードへと展開してくれます. 例えば, この方法で R 関数を作ることができます.

華氏温度を摂氏温度に変換する関数を定義する

```
```{r, f2c}
F2C <- function(x) {
  <<check-arg>>
  <<convert>>
}
```
```

```
```

最初に入力値が数値か確認する

```{r, check-arg, eval=FALSE}
  if (!is.numeric(x)) stop("入力は数値でなければなりません!")
```

それから実際に変換します

```{r, convert, eval=FALSE}
  (x - 32) * 5/ 9
```

これはドナルド=クヌースの提案する文芸プログラミング[*1] の主要なアイディアの 1 つに基づいたものです. この技術の利点は (複雑な) コードを小さな部品に分割し, 別々のコードチャンクに書き, 文脈の中で説明することができる点です. 全ての部品は実行される主要なコードチャンクで構成することができます.

上記の例に対して, `f2c` というラベルのある最初のコードチャンクはこうなります.

```{r, f2c}
F2C <- function(x) {
  if (!is.numeric(x)) stop("The input must be numeric!")
  (x - 32) * 5/ 9
}
```

1 つのコードチャンクには好きな数のコードチャンクを埋め込むことが可能です. 埋め込みは再帰的にすることも可能です. 例えば, チャンク A をチャンク B に埋め込み, さらにチャンク B をチャンク C に埋め込むこともできます. チャンク C はチャンク B から読み込まれたチャンク A を含むことになります.

マーカー <<**ラベル**>> は独立した行に置く必要はありません. コードチャンクのどこにでも埋め込

[*1] https://en.wikipedia.org/wiki/Literate_programming

むことができます.

14.1.2 別のチャンクで同一のチャンクラベルを使う

完全に同じコードチャンクを 2 回以上使いたいならば, ラベル付きのチャンクを定義し, さらに同じラベルで中身が空のチャンクを作ることもできます. 例えばこのように.

これは評価されないコードチャンクです

```
```{r, chunk-one, eval=FALSE}
1 + 1
2 + 2
```
```

実際に評価されるのはこちらです

```
```{r, chunk-one, eval=TRUE}
```
```

上記の例でチャンクラベル “chunk-one” を 2 度使っており, 2 度目のチャンクは最初のチャンクの単なる再利用です.

グラフないしは他のファイルを生成するのに, この方法で複数回コードチャンクを実行するのはお薦めしません. 後のチャンクで作成された画像ファイルがそれ以前のものを上書きするかもしれないからです. これらのチャンクのうち 1 つだけにチャンクオプション `eval = TRUE` を使い, それ以外では `eval =FALSE` を使うのならば大丈夫です.

14.1.3 参照ラベルを使う (*)

チャンクオプション `ref.label` はチャンクラベルのベクトルを取り, そのチャンクの中身を取得できます. 例えば以下の `chunk-a` というラベルのコードチャンクは `chunk-c` と `chunk-b` を結合したものです.

```
```{r chunk-a, ref.label=c('chunk-c', 'chunk-b')}
```


```{r chunk-b}
# これはチャンク b
1 + 1
```


```{r chunk-c}
# これはチャンク c
2 + 2
```
```

言い換えるなら, chunk-a は本質的にこうなります.

```
```{r chunk-a}
# これはチャンク c
2 + 2
# これはチャンク b
1 + 1
```
```

チャンクオプション ref.label のあるおかげで, コピーアンドペーストを使うことなくコードチャンクをとても柔軟に再構成することができます. 参照先のコードチャンクが ref.label が使われたチャンクの前にあるか, 後にあるかは問題になりません. 先に書かれたコードチャンクは後のコードチャンクを参照できます.

4.19 節にはこのチャンクオプションの応用例があります.

14.2 オブジェクトが作られる前に使用する (*)

knitr 文書内の全てのコードは, コードチャンクとインライン R コードも含めて, 最初から最後まで順番に実行されます. 理論上は, 値が代入される前の変数を使うことができません. しかし場合により, 文書内で変数の値により早く言及したいことがあるでしょう. 例えば論文の中で結果を概

要欄に掲載したいというのはよくある状況ですが, 実際には結果は文書のもっと後で計算されます. 以下の例はそのアイディアを具体化したものですが, 実行はできません.

```
---
title: 重要なレポート
概要: >
  この分析では `x` の平均値が
  `r mx` であった.
---

我々は次のチャンクで `mx` を作成した.

```{r}
x <- 1:100
mx <- mean(x)
```
```

この問題を解決するには, オブジェクトの値がどこかに保存され, 文書が次回コンパイルされる時に読み込まれなければなりません. これは, 文書が最低でも 2 回コンパイルされなければならないという意味であることに注意してください. 以下は `saveRDS()` 関数を使った, 実行可能な解決策の 1 つです.

```
```{r, include=FALSE}
mx <- if (file.exists('mean.rds')) {
  readRDS('mean.rds')
} else {
  "`mx` の値はまだ利用できない"
}
```


---
title: 重要なレポート
概要: >
  この分析では `x` の平均値が
```

```
  `r mx` であった.
---

我々は次のチャンクで `mx` を作成した.

```{r}
x <- 1:100
mx <- mean(x)
saveRDS(mx, 'mean.rds')
```
```

最初のコンパイルでは, 概要に「mx の値はまだ利用できない」という文言が現れます. その後, もう 1 度コンパイルすると mx の値が現れます.

knitr::load_cache() 関数はもう 1 つの解決策で, 特定のコードチャンクでキャッシュ済みのオブジェクトから値を読み込むことできます. このアイディアは上記の例と似ていますが, オブジェクトが自動でキャッシュデータベースに保存されるため, オブジェクトを手動で保存して読み込む手間を省くことになります. あなたは load_cache() で読み込むだけでいいのです. 以下は単純化した例です.

```
---
title: An important report
abstract: >
  この分析では `x` の平均値が
  `r knitr::load_cache('mean-x', 'mx')` であった.
---

我々は次のチャンクで `mx` を作成した.

```{r mean-x, cache=TRUE}
x <- 1:100
mx <- mean(x)
```
```

この例では, チャンクラベル mean-x をコードチャンクに追加し, それを load_cache() 関数に渡します. そしてチャンクオプション cache = TRUE でチャンクをキャッシュしています. このコードチャンクの全てのオブジェクトはキャッシュデータベースに保存されます. 繰り返しになりますが, オブジェクト mx はキャッシュデータベースから正しく読み込まれるには, この文書を最低でも 2 回コンパイルしなければなりません. mx の値が将来も変更される予定がないなら, 文書をこれ以上コンパイルする必要はありません.

もし load_cache() の第 2 引数でオブジェクト名を指定しないなら, キャッシュデータベース全体が現在の環境に読み込まれます. そうすると, 文書の後方でオブジェクトが作成される前でも, キャッシュデータベースにあるどのオブジェクトも使えます. これが例です.

```
knitr::load_cache("mean-x")
x  # the object `x`
mx  # the object `mx`
```

14.3 knit 処理を打ち切る

時には knit 処理を文書の末尾よりも早い時点で終了したいこともあります. 例えば何かを分析する作業をしていて, 結果の前半だけを共有したいとか, まだ最後のコードが書ききれていないというときです. このような状況ではコードチャンクで knit_exit() 関数を使ってみましょう. この関数はそのチャンクの直後で knit 処理を終わらせることができます.

以下は単純な例です. ここではとても単純なチャンクと, その後にもっと時間のかかるチャンクを配置しています.

````
```{r}
1 + 1
knitr::knit_exit()
```
````

あなたは出力のうち上記のコンテンツだけを見たい.

````
```{r}
Sys.sleep(100)
```
````

```
```

通常ならば 100 秒待つところですが, knit_exit() を呼び出しているので文書の残りの部分は無視されます.

14.4 どこにでもグラフを生成し, 表示させる

グラフは通常コードチャンク内で生成され, その直下に表示されますが, 以下の例のように表示場所を好きなところに指定したり, コードチャンクに隠すことも選べます.

このコードチャンクでグラフを生成しますが, 表示はしません.

```
```{r cars-plot, dev='png', fig.show='hide'}
plot(cars)
```
```

別の段落でグラフを導入します

```
![A nice plot.](`r knitr::fig_chunk('cars-plot', 'png')`)
```

一時的にグラフを隠すためにコードチャンクでチャンクオプション fig.show='hide' を使用しました. それから別の段落で knitr::fig_chunk() 関数を呼び出して, このグラフ画像のファイルパスを取得しました. このパスは普通は test_files/figure-html/cars-plot-1.png のようになっています. このファイルパスを導出するためには, fig_chunk() 関数にチャンクラベルとグラフィックデバイス名を渡す必要があります.

blogdown で作成したウェブサイトへの fig_chunk() の応用を https://stackoverflow.com/a/46305297/559676 で見ることもできます. この関数はどの R Markdown 出力フォーマットでも動作します. 特にスライド上では, スクリーンの広さが限られているため, 画像を表示するのに便利でしょう. 1 つのスライドでコードを提示し, さらに別のスライドで画像を表示させることもできます.

14.5　以前のコードチャンクのグラフを修正する

knitr はデフォルトでは, コードチャンクごとに新規にグラフィックデバイスを開いてグラフを記録しています. これは 1 つ問題を起こしています. グラフィックデバイスが既に閉じられているため, 以前のコードチャンクで作成されたグラフを簡単には修正できないという問題です. base R のグラフィックではたいていの場合で問題となります. なお **ggplot2** (Wickham　Chang, et al., 2021) のような grid ベースのグラフィックは, グラフを R オブジェクトとして保存できるので当てはまりません. 例えばあるコードチャンクでグラフを描き, 後のチャンクでグラフに線を描き足そうとしても, R は高水準グラフがまだ作られていないというエラーを示すので, 線を描き足すことができません.

全てのコードチャンクでグラフィックデバイスを開いたままにしたいなら, 文書の冒頭で **knitr** パッケージのオプションである　を設定します.

```
01 knitr::opts_knit$set(global.device = TRUE)
```

より頻繁に使われる `opts_chunk` ではなく `opts_knit` が使われていることに注意してください. 例は Stack Overflow の https://stackoverflow.com/q/17502050 という投稿で見ることもできます.

グローバルなグラフィックデバイスを必要としなくなった時は, オプションを `FALSE` に設定できます. これは完全な例です.

````
---
title: "グラフの保存にグローバルグラフィックデバイスを使用する"
---

まず，グローバルグラフィックデバイスを有効にします.

```{r, include=FALSE}
knitr::opts_knit$set(global.device = TRUE)
```

グラフを描画します.
````

````
```{r}
par(mar = c(4, 4, 0.1, 0.1))
plot(cars)
```
````

以前のコードチャンクのグラフに線を追加します．

````
```{r}
fit <- lm(dist ~ speed, data = cars)
abline(fit)
```
````

グローバルデバイスを切ります．

````
```{r, include=FALSE}
knitr::opts_knit$set(global.device = FALSE)
```
````

別のグラフを描画します．

````
```{r}
plot(pressure, type = 'b')
```
````

14.6　グループ化したチャンクオプションを保存し再利用する (*)

いくつかのチャンクオプションを頻繁に使うのなら，それらを 1 つのグループとして保存し，以降はグループ名を書くだけで再利用できるようにするとよいかもしれません．これは `knitr::opts_template$set(name = list(options))` で実行できます．それからチャンクオプション `opts.label` を用いてこのグループ名を参照できます．例えばこのように．

````
```{r, setup, include=FALSE}
knitr::opts_template$set(fullwidth = list(
```
````

````
  fig.width = 10, fig.height = 6,
  fig.retina = 2, out.width = '100%'
))
```


```{r, opts.label='fullwidth'}
plot(cars)
```
````

opts.label = 'fullwidth' とすると, **knitr** は knitr::opts_template から一連のチャンクオプションを読み込み, 現在のチャンクに適用します. これはタイピングの労力を削減できます. チャンクオプションを文書全体で使用しなければならないならば, グローバルに設定すべきでしょう (11 章参照).

opts.label から読み込んだオプションを上書きすることもできます. 例えば以下のチャンクで fig.height = 7 を設定したなら, 実際の値は 6 でなく 7 になります.

````
```{r, opts.label='fullwidth', fig.height=7}
plot(cars)
```
````

オプションのグループは好きな数だけ保存できます. 例えば knitr::opts_template$set(group1 = list(...), group2 = list(...)) のように.

14.7 Rmd ソースの生成に knitr::knit_expand() を使う

knitr::knit_expand() 関数は, デフォルトで {{ }} 内の表現を値に展開 (expand) します. これが例です.

```
knitr::knit_expand(text = "`pi` の値は {{pi}} である.")
## [1] "`pi` の値は 3.14159265358979 である."
knitr::knit_expand(
  text = "`a` の値は {{a}} なので, `a + 1` は {{a+1}} である.",
```
```

```
  a = round(rnorm(1), 4)
)
## [1] "`a` の値は 0.2198 なので, `a + 1` は 1.2198 である."
```

{{ }} 内に動的なものが含まれている Rmd 文書であれば, knit_expand() を適用して knit() を呼び出してコンパイルすることができるということを, この例は意味しています. 例えばここに template.Rmd という文書があったとします.

```
# {{i}} に対する回帰

```{r lm-{{i}}}
lm(mpg ~ {{i}}, data = mtcars)
```
```

mtcars データセット内の mpg に対する他の変数全てを一つ一つ使った線型回帰モデルを構築できます.

```
```{r, echo=FALSE, results='asis'}
src = lapply(setdiff(names(mtcars), 'mpg'), function(i) {
  knitr::knit_expand('template.Rmd')
})
res = knitr::knit_child(text = unlist(src), quiet = TRUE)
cat(res, sep = '\n')
```
```

この例が難しくて理解できないと感じたら, チャンクオプション results = 'asis' の意味を知るのに 11.11 節を, knitr::knit_child() の使用法を知るのに 16.4 節を見てください.

14.8 コードチャンクにラベルの重複を許可する (*)

knitr はデフォルトでは文書内でチャンクラベルが重複することを許可しません. 重複するラベルは文書を knit する際にエラーを引き起こします. これは文書内でコードチャンクをコピーアンド

ペーストするときに最もよく起こります. あなたもこのようなエラーメッセージにでくわしたことがあるかもしれません.

```
processing file: myfile.Rmd
Error in parse_block(g[-1], g[1], params.src, markdown_mode) :
  Duplicate chunk label 'cars'
Calls: <Anonymous> ... process_file -> split_file -> lapply ->
  FUN -> parse_block
Execution halted
```

しかし, 重複するラベルを許可したいこともあるというものです. 例えば親文書 `parent.Rmd` があり, その中で子文書を複数回 knit すれば, 失敗するでしょう.

```
# 設定
settings <- list(...)

# 1 度目の実行
knit_child("useful_analysis.Rmd")

# 新しい設定
settings <- list(...)

# 再実行
knit_child("useful_analysis.Rmd")
```

この筋書きでは, 子文書が knit される**前に** R のグローバルオプションを設定することでラベルの重複を許可できます.

```
options(knitr.duplicate.label = "allow")
```

子文書ではなくメインの文書でラベルの重複を許可したいなら, `knitr::knit()` が呼び出される**前に**設定しなければなりません. それを実現する可能性の 1 つとして, `~/.Rprofile` ファイル内で設定するという方法があります (詳細は `?Rprofile` のヘルプを見てください).

このオプションの設定は注意深くすべきです．ほとんどのエラーメッセージは，それなりの理由があってこそ存在します．重複するチャンクを許可することは図や相互参照に関して気が付かないうちに問題を生み出す可能性があります．例えば，グラフ画像のファイル名はチャンクラベルによって決まるので，2 つのコードチャンクが同じラベルを持ち，かつ両方のチャンクが図を生成しているなら，理論上はこれらの画像ファイルは互いに上書きすることになります (そしてエラーも警告も発しません)．**knitr** は `knitr.duplicate.label = "allow"` オプションがあると，重複するラベルに暗黙に数字の接頭語を追加して変更しています．例えば，2 つのコードチャンクに対してはこうなります．

````
```{r, test}
plot(1:10)
```

```{r, test}
plot(10:1)
```
````

2 つ目のラベルは暗黙のうちに `test-1` に変更されます．これはラベル `test` のチャンクからのグラフ画像を上書きすることを回避するかもしれませんが，同時にチャンクラベルが予想できなくなります．ということは，図の相互参照 (4.7 節参照) も相互参照がチャンクラベルに基づいているので難しくなるでしょう．

14.9 より透明性のあるキャッシュの仕組み

11.4 節で紹介した **knitr** のキャッシュの仕組みが複雑すぎると思ったら (実際そうです!)，`xfun::cache_rds()` 関数に基づいた，より簡単なキャッシュの仕組みを検討してください．これが例です．

```
xfun::cache_rds({
  # ここに時間のかかるコードを書く
})
```

knitr のキャッシュは，キャッシュの無効化のタイミングがどう決定されるかという点が難解なのです．`xfun::cache_rds()` においては，これはずっと明確です．最初に R コードをこの関数に与え

たときは, コードが評価され結果が .rds ファイルに保存されます. 次に cache_rds() を再実行すると, .rds ファイルを読み込み, コードを再び評価することなく直ちに結果を返します. キャッシュを無効化する最も明確な方法は, .rds ファイルを削除することです. 手動で削除したくないなら, xfun::cache_rds() に rerun = TRUE 引数を付けて呼び出します.

knitr のソース文書上のコードチャンクで xfun::cache_rds() が呼び出された時, .rds ファイルのパスはチャンクオプション cache.path とチャンクラベルによって決定します. 例えば input.Rmd という Rmd 文書に foo というチャンクラベルのあるコードチャンクがあるとします.

````
```{r, foo}
res <- xfun::cache_rds({
  Sys.sleep(3)
  1:10
})
```
````

.rds ファイルのパスは input_cache/FORMAT/foo_HASH.rds という形式になります. ここで FORMAT は Pandoc の出力フォーマット名 (例えば html あるいは latex) であり, HASH は a-z および 0-9 からなる 32 桁の 16 進 MD5 ハッシュ値です. 例えば input_cache/html/foo_7a3f22c4309d400eff95de0e8bddac71.rds のようになります.

?xfun::cache_rds のヘルプで言及されているように, キャッシュを無効化したいであろう 2 つのよくあるケースがあります. (1) 評価式が変更された時, (2) 評価式の外部の変数が使用され, その変数の値が変更された時です. 次に, この 2 つのキャッシュ無効化の方法がどう動作するのかと, 異なるコードのバージョンに対応する複数のキャッシュのコピーをどう保持するかを説明します.

14.9.1 コードの変更によってキャッシュを無効化する

例えば cache_rds({x + 1}) から cache_rds({x + 2}) へと, cache_rds() 内のコードを変更したとき, キャッシュは自動で無効化され, コードは再評価されます. しかし, 空白やコメントの変更は問われないことに注意してください. あるいは一般論として, パースされた表現に影響のない範囲の変更ではキャッシュは無効化されません. 例えば cache_rds() にパースされた以下 2 つのコードは本質的に同等です.

```
res <- xfun::cache_rds({
  Sys.sleep(3  );
  x<-1:10;  # セミコロンは問題ではない
  x+1;
})

res <- xfun::cache_rds({
  Sys.sleep(3)
  x <- 1:10  # これはコメント
  x +
    1  # 空白の変更は完全に自由
})
```

つまり, 最初のコードを cache_rds() で実行したなら, 2 度目のコードはキャッシュの利便性を得られます. この仕様のおかげでキャッシュを無効化することなくコードの見た目を整える変更ができます.

2 つのバージョンのコードが同等であるか自信がないなら, 以下の parse_code() を試してください.

```
parse_code <- function(expr) {
  deparse(substitute(expr))
}
# 空白とセミコロンは影響しない
parse_code({x+1})
```

```
## [1] "{"         "    x + 1" "}"
```

```
parse_code({ x   +    1; })
```

```
## [1] "{"         "    x + 1" "}"
```

```
# 左アロー演算子と右アロー演算子は同等
identical(parse_code({x <- 1}), parse_code({1 -> x}))
```

```
## [1] TRUE
```

14.9.2 グローバル変数の変更によってキャッシュを無効化する

変数にはグローバルとローカル変数の 2 種類があります．グローバル変数は評価式の外部で作られ，ローカル変数は評価式の内部で作られます．キャッシュされた結果は，評価式内のグローバル変数の値が変われば，もはや再度実行して得られるはずの結果を反映していません．例えば以下の評価式で，y が変化したなら，あなたが一番やりたいのはきっと，キャッシュを無効化して評価をやり直すことでしょう．さもなければ古い y の値を維持したままになってしまいます．

```
y <- 2

res <- xfun::cache_rds({
  x <- 1:10
  x + y
})
```

y が変化した時にキャッシュを無効化するには，キャッシュを無効化すべきかを決定する際に y も考慮する必要があることを，hash 引数を通して cache_rds() に教えてあげることもできます．

```
res <- xfun::cache_rds({
  x <- 1:10
  x + y
}, hash = list(y))
```

hash 引数の値が変化した時，前述のキャッシュファイル名に含まれる 32 桁のハッシュ値も対応して変化するため，キャッシュは無効化されます．これで他の R オブジェクトとキャッシュの依存関係を指定する手段を得ました．例えば R のバージョンに依存してキャッシュを取りたいなら，このようにして依存関係を指定することもできます．

```
res <- xfun::cache_rds({
  x <- 1:10
  x + y
}, hash = list(y, getRversion()))
```

あるいはデータファイルが最後に修正されたタイミングに依存させたいなら, こうします.

```
res <- xfun::cache_rds({
  x <- read.csv("data.csv")
  x[[1]] + y
}, hash = list(y, file.mtime("data.csv")))
```

hash 引数にこのグローバル変数のリストを与えたくなければ, 代わりに hash = "auto" を試しましょう. これは cache_rds() に全てのグローバル変数を自動的に把握するよう指示し, 変数の値のリストを hash 引数の値として使わせます.

```
res <- xfun::cache_rds({
  x <- 1:10
  x + y + z  # y と z はグローバル変数
}, hash = "auto")
```

これは以下と同等です.

```
res <- xfun::cache_rds({
  x <- 1:10
  x + y + z  # y と z はグローバル変数
}, hash = list(y = y, z = z))
```

hash = "auto" とした時, グローバル変数は codetools::findGlobals() によって識別されます. これは完全に信頼できるものではありません. あなたのコードを一番良く知っているのはあなた自身ですので, hash 引数には明示的に値のリストを指定して, どの変数がキャッシュを無効化できるか

を万全にすることをお薦めします.

14.9.3 キャッシュの複数のコピーを保持する

キャッシュは典型的には時間のかかるコードに対して使用されるので, きっとあなたは無効化することに対して躊躇するでしょう. キャッシュを無効化するのが早すぎたり, 積極的すぎたりしたことを後悔するかもしれません. もし古いバージョンのキャッシュが再び必要になったら, 再現のために長い計算時間を待たなければなりませんから.

`cache_rds()` の `clean` 引数を `FALSE` に設定すれば, キャッシュの古いコピーを保持できます. R のグローバルオプション `options(xfun.cache_rds.clean = FALSE)` の設定で, この挙動を R セッション全体を通したデフォルトにもできます. デフォルトでは, `clean = TRUE` と `cache_rds()` は毎回, 古いキャッシュを削除しようと試みます. `clean = FALSE` の設定は, まだコードを試行錯誤しているうちは有用になりえます. 例えば, 2 つのバージョンの線形モデルのキャッシュを取ることができます.

```
model <- xfun::cache_rds({
  lm(dist ~ speed, data = cars)
}, clean = FALSE)

model <- xfun::cache_rds({
  lm(dist ~ speed + I(speed^2), data = cars)
}, clean = FALSE)
```

どちらのモデルを使うかを決めたら, `clean = TRUE` を再度設定するか, この引数を消すことでデフォルトの `TRUE` に戻すことができます.

14.9.4 knitr のキャッシュ機能との比較

knitr キャッシュ, つまりチャンクオプション `cache = TRUE` と, `xfun::cache_rds()` をそれぞれいつ使えばよいのか迷っているかもしれません. `xfun::cache_rds()` の最大の欠点は, 評価式の値のみをキャッシュしそれ以外の結果をキャッシュしないことです. その一方で **knitr** は評価式以外の値についてもキャッシュを取ります. 出力やグラフを表示するといった評価式以外の結果には有用なものもあります. 例えば以下のコードでは, `cache_rds()` が次にキャッシュを読み込んだ時には, テキスト出力とグラフが失われてしまい, `1:10` という値だけが戻ってきます.

```
xfun::cache_rds({
  print("Hello world!")
  plot(cars)
  1:10
})
```

これと比較してオプション cache = TRUE のあるコードチャンクでは, 全てがキャッシュされます.

````
```{r, cache=TRUE}
print("Hello world!")
plot(cars)
1:10
```
````

knitr のキャッシュ機能の大きな欠点であると同時にユーザーが最もよく不満の対象とするのは, キャッシュがとても多くの要因で決まるため, 知らないうちに無効化してしまうことがある点です. 例えば, チャンクオプションのいかなる変更もキャッシュを無効化する可能性がありますが,[*2] 演算に影響しないであろうチャンクオプションもあります. 以下のコードチャンクでチャンクオプション fig.width = 6 を fig.width = 10 へと変更してもキャッシュを無効化すべきではありませんが, 実際は無効化してしまいます.

````
```{r, cache=TRUE, fig.width=6}
# there are no plots in this chunk
x <- rnorm(1000)
mean(x)
```
````

実際に **knitr** のキャッシュはかなり強力で柔軟であり, 多くの方法で挙動を調整できます. あなたはキャシュがどう動作するのかを学び理解するのに, 最終的に計算するタスクの所要時間よりもはるかに多くの時間を費やしてしまうかもしれません. ですので私はパッケージの作者として, これ

[*2] これはデフォルトの挙動であり, 変更することができます. より細かい粒度でキャッシュを生成し, 全てのチャンクオプションがキャッシュに影響しないようにするには, https://gedevan-aleksizde.github.io/knitr-doc-ja/cache.html をご覧ください.

らのあまり知られていない機能は紹介するに値するのかと, 疑問に思うことがよくあります.

まだはっきりわからない人は, `xfun::cache_rds()` は演算をキャッシュする一般的な方法でありどこでも動作しますが, 一方の **knitr** のキャッシュは **knitr** 文書でのみ動作すると覚えてください.

第15章

その他の言語

R Markdown は **knitr** を通して R 言語以外の多くのプログラミング言語をもサポートしています．言語の名前は 3 連続のバッククオートの後のカーリーブレースの最初の単語で表現されます．例えば ```{r} の小文字の r はコードチャンクに R のコードが含まれていることを意味し，```{python} は Python のコードチャンクであることを表しています．この章ではあなたがあまり詳しくないであろういくつかの言語をお見せします．

knitr では，どの言語も言語エンジンを通してサポートされています．言語エンジンは本質的にはソースコードとコードチャンクを入力として，出力として文字列を返す関数です．これらは knitr::knit_engines オブジェクトで管理されています．既存のエンジンはこのようにして確認することもできます．

```
01 names(knitr::knit_engines$get())
```

```
##  [1] "awk"       "bash"      "coffee"    "gawk"
##  [5] "groovy"    "haskell"   "lein"      "mysql"
##  [9] "node"      "octave"    "perl"      "psql"
## [13] "Rscript"   "ruby"      "sas"       "scala"
## [17] "sed"       "sh"        "stata"     "zsh"
## [21] "highlight" "Rcpp"      "tikz"      "dot"
## [25] "c"         "cc"        "fortran"   "fortran95"
## [29] "asy"       "cat"       "asis"      "stan"
## [33] "block"     "block2"    "js"        "css"
## [37] "sql"       "go"        "python"    "julia"
## [41] "sass"      "scss"      "R"         "bslib"
```

```
## [45] "targets"
```

現時点では, R 言語でないほとんどの言語はコードチャンクごとに独立して実行されます. 例えば, 同じ文書内の bash コードチャンクは全てそれぞれ別々のセッションで実行されるため, 後の bash コードチャンクはそれ以前の bash チャンクで作成された変数を使うことができませんし, cd による作業ディレクトリの変更も異なる bash チャンク間で維持できません. R, Python, そして Julia のコードチャンクのみが同一セッションで実行されます. 全ての R コードチャンクは同一の R セッションで実行され, 全ての Python コードチャンクは同一の Python セッションされ……, ということに注意してください. R セッションと Python セッションは 2 つの異なるセッションですが, 一方のセッションからもう一方のセッションのオブジェクトにアクセスしたり操作したりすることは可能です (15.2 節参照).

R Markdown Definitive Guide (Xie J.J. Allaire, and Grolemund, 2018) の Section 2.7[*1] では Python, シェル, SQL, Rcpp, Stan, JavaScript, CSS, Julia, C そして Fortran のコードを使用する例が紹介されています. この章ではさらなる言語エンジンを紹介します. そしてさらなる例はリポジトリ https://github.com/yihui/knitr-examples で見られます. “engine” という単語を含むファイルを探してください.

初めに, カスタム言語エンジンの登録によってこれがどのように動作するかを解明しましょう.

15.1 カスタム言語エンジンを登録する (*)

knitr::knit_engines$set() でカスタム言語エンジンを登録できます. これは関数を入力として受け容れます. これが例です.

```
knitr::knit_engines$set(foo = function(options) {
  # ソースコードは options$code にある
  # それを使ってやりたいことは何でもやろう
})
```

これは foo エンジンを登録し, ```{foo} で始まるコードチャンクを使えるようになります.

エンジン関数は 1 つの引数 options を取り, これはコードチャンクのオプションのリストです. options$code にある文字列ベクトルとして, チャンクのソースコードにアクセスできます. 例えば, このコードチャンクに対して考えます.

[*1] https://bookdown.org/yihui/rmarkdown/language-engines.html

````
```{foo}
1 + 1
2 + 2
```
````

options の code 要素は文字列ベクトル c('1 + 1', '2 + 2') になります.

言語エンジンは実はプログラミング言語として動作しなくてもよいですが, コードチャンクの任意のテキストを処理できます. まずは, コードチャンクの本文を大文字に変換するエンジンの例をお見せします.

```
knitr::knit_engines$set(upper = function(options) {
  code <- paste(options$code, collapse = "\n")
  if (options$eval)
    toupper(code) else code
})
```

ポイントは toupper 関数を「コード」に適用して, \n でコードの全ての行を連結し, 単一の文字列として結果を返すことです. toupper() はチャンクオプション eval = TRUE の時にのみ適用され, そうでなければ元の文字列が返されることに注意してください. このことは eval のようなチャンクオプションをエンジン関数内で利用する方法を示唆しています. 同様に, results = 'hide' の時に出力を隠すため, 関数内に if (options$results == 'hide') return() を加えることも検討することもできます. 以下は upper エンジンをオプションとともに使用するチャンクの例です.

````
```{upper}
Hello, **knitr** engines!
```
````

HELLO, **KNITR** ENGINES!

次に, py という名前のもう 1 つの Python エンジン[*2]の例を紹介します. このエンジンは単純に R

[*2] 実用的には組み込みの python エンジンを使うべきです. これは **reticulate** パッケージに基づいており, より良

の system2() 関数から python コマンドを呼び出すことで実装しています.

```
knitr::knit_engines$set(py = function(options) {
  code <- paste(options$code, collapse = '\n')
  out  <- system2(
    'python3', c('-c', shQuote(code)), stdout = TRUE
  )
  knitr::engine_output(options, code, out)
})
```

上記のエンジン関数を完全に理解するために, 以下を知っておく必要があります.

1. Python コードは文字列として与えられ (上記関数の code), コードはコマンドラインの呼び出し python -c 'code' によって実行できます. これが system2() のしていることです. system2() stdout = TRUE を指定することでテキスト出力を収集しています.
2. 最終的な出力を生成するため, チャンクオプション・ソースコード・テキスト出力を knitr::engine_output() 関数に与えることができます. この関数は echo = FALSE と results = 'hide' のようなよく使うオプションを処理します. よってあなたはこれらの場合に注意する必要はありません.

knitr の多くの言語エンジンはこのようにして定義されています. つまり system2() を使って言語に対応するコマンドを実行してます. もし技術的に詳しい話に興味があるなら, R ソースコードにはほとんどの言語エンジンが書かれているここ https://github.com/yihui/knitr/blob/master/R/engine.R を確認することもできます.

そして今や, 新しいエンジン py を使うことができます. 例えばこのように.

````
```{py}
print(1 + 1)
```


## 2
````

く Python コードチャンクをサポートしてくれます (15.2 節参照).

あなたのバージョンの言語エンジンが **knitr** の既存の言語エンジンよりも必要性がるか, より良いものだと確信しているなら, `knitr::knit_engines$set()` によって既存のものを上書きすることすらできます. たいていの場合は既存のエンジンに慣れたユーザーが驚いてしまうかもしれないので, そうすることはお薦めしませんが, どちらにせよこの可能性は頭の片隅に置いてほしいです.

15.2 Python コードの実行と双方向処理

あなたが Python を好んでいることは知っていますので, とてもはっきりと言ってしまいましょう. R Markdown と **knitr** はなんと Python をサポートしています.

Python のコードチャンクを R Markdown 文書に加えるには チャンクヘッダ ```` ```{python} ```` を使うことができます. 例えばこのように.

````
```{python}
print("Hello Python!")
```
````

いつもどおりにチャンクヘッダに `echo = FALSE` or `eval = FALSE` といったチャンクオプションを追加することができます. Python の **matplotlib** パッケージで描かれたグラフもサポートしています.

R Markdown と **knitr** の Python サポートは **reticulate** パッケージ (Ushey JJ Allaire, and Tang, 2021) に基づいており, このパッケージの重要な機能の 1 つは Python と R の双方向的なコミュニケーションを可能にすることです. 例えば **reticulate** の `py` オブジェクトを介して R セッションから Python の変数にアクセスしたり作成したりすることもできます.

````
```{r, setup}
library(reticulate)
```


Python セッションで変数 `x` を作成する

```{python}
x = [1, 2, 3]
```
````

R コードチャンクで Python 変数 `x` にアクセスする

```
```{r}
py$x
```
```

R を使って Python セッションで新しい変数 `y` を作成し，
`y` にデータフレームを与える

```
```{r}
py$y <- head(cars)
```
```

Python で変数 `y` を表示する

```
```{python}
print(y)
```
```

reticulate パッケージに関する詳細については，https://rstudio.github.io/reticulate/ のドキュメントを見ることもできます．

15.3 asis エンジンでコンテンツを条件付きで実行する

その名が示すとおり，asis エンジンはチャンクの内容をそのまま書き出します．このエンジンを使う利点は条件に応じてコンテンツを読み込めることです．つまりチャンクオプション echo によりチャンクの内容の表示を決定します．echo = FALSE の時はチャンクは隠されます．以下は簡単な例です．

```
```{r}
getRandomNumber <- function() {
  sample(1:6, 1)
```
```

````
}
```


```{asis, echo = getRandomNumber() == 4}
https://xkcd.com/221/ によれば，**真の**乱数を生成しました!
```
````

asis チャンク内のテキストは条件式 getRandomNumber() == 4 が (ランダムに) 真であるならば表示されます.

15.4 シェルスクリプトを実行する

あなたが好んでいるシェルに応じて, bash・sh・zsh エンジンでシェルスクリプトを実行できます. 以下はチャンクヘッダ ```{bash} を使った bash の例です.

```
01 ls *.Rmd | head -n 5
```

```
## index.Rmd
## rmarkdown-cookbook.Rmd
```

bash は R の system2() 関数で呼び出されていることに注意してください. ~/.bash_profile や ~/.bash_login のようなプロファイルにある, あなたの定義したコマンドのエイリアスや PATH などの環境変数は無視されます. ターミナル上でシェルを使っている時のようにこれらのプロファイルがほしいなら, engine.opts を介して -l 引数を与えることもできます. これが例です.

````
```{bash, engine.opts='-l'}
echo $PATH
```
````

-l 引数を全ての bash チャンクで有効にしたいなら, 文書の冒頭でグローバルチャンクオプションに設定することもできます.
````

```
knitr::opts_chunk$set(engine.opts = list(bash = "-l"))
```

チャンクオプション engine.opts に文字列ベクトルとして他の引数を bash に与えることもできます.

15.5 D3 で可視化する

R のパッケージ **r2d3** (Strayer Luraschi, and JJ Allaire, 2020) は D3 可視化のインターフェースです. このパッケージは例えば Shiny のような他のアプリケーションと同様に R Markdown 文書内で使うことができます. R Markdown 内で使うにはコードチャンクで r2d3() 関数を呼び出すか, d3 エンジンを使用することができます. 後者は D3 ライブラリと Javascript の理解が要求されますが, それは本書で扱う範囲を超えますので, 読者自身による学習に任せます. 以下は d3 エンジンで棒グラフを描く例です.

````
---
title: "D3 でグラフを生成する"
output: html_document
---

最初に，**r2d3** パッケージを読み込み **knitr** が自動で
`d3` エンジンをセットアップしてくれるようにします

```{r setup}
library(r2d3)
```

ここで R でデータを生成して D3 に渡してグラフを描画できます.

```{d3, data=runif(30), options=list(color='steelblue')}
svg.selectAll('rect')
  .data(data)
  .enter()
    .append('rect')
````
````

```
      .attr('width', function(d) { return d * 672; })
      .attr('height', '10px')
      .attr('y', function(d, i) { return i * 16; })
      .attr('fill', options.color);
```
```

## 15.6 cat エンジンでチャンクをファイルに書き出す

コードチャンクの内容を外部ファイルに書き出し, 以降の他のコードチャンクで使用するのは時には有用である可能性があります. もちろん, `writeLines()` のような R の関数で行っても良いですが, 内容が比較的長かったり, 特殊な文字が含まれていたり, `writeLines()` に渡したい文字列がごちゃごちゃしたりしているかもしれません. 以下は 長い文字列を `my-file.txt` に書き出す例です.

```
writeLines("これは長い文字列です.
複数行にわたります. ダブルクオート \"\" は
忘れずにエスケープしてください.
ですが ' シングルクオート' は大丈夫です.
バックスラッシュがいくつ必要か考えるときにあなたが
正気を失わないでいられることを願います.
例えば, '\t' なのか `\\t` なのか '\\\\t' なのか?",
  con = "my-file.txt")
```

R 4.0.0 以降では `r"()"` 内での生の文字列 (`?Quotes` のヘルプ参照) がサポートされ始めたので, 特殊文字のルールを全て覚える必要はなくなり, この問題は大いに緩和されました. 生の文字列があってもなお, チャンク内で長い文字列を明示的にファイルに書き出すことは読者の注意力を少しばかり削ぐ可能性があります.

knitr の `cat` エンジンは, 例えばバックスラッシュのリテラルが必要な時は, 二重バックスラッシュが必要といった, R の文字列ルールを一切考えることなく, コードチャンクの内容の表示かつ / または外部ファイルへの書き出しの方法を提供してくれます.

チャンクの内容をファイルに書き出すには, チャンクオプション `engine.opts` にファイルパスを指定してください. 例えば `engine.opts = list(file = 'path/to/file')` のように. この内部では, `engine.opts` で指定された値のリストが `base::cat()` に渡されます. そして `file` は `base::cat()` の
```

引数の 1 つです.

次に, cat エンジンの使い方の詳しい説明のため 3 つの例を提示します.

15.6.1 CSS ファイルへ書き込む

7.3 節でお見せしたように, 要素を CSS でスタイル設定するために css コードチャンクを Rmd 文書に埋め込むことができます. 別の方法として, カスタム CSS ファイルを, html_document のようないくつかの R Markdown 出力フォーマットで有効な css オプションを介して Pandoc に渡す方法もあります. cat エンジンはこの CSS ファイルを Rmd から書き込むのに使用できます.

以下の例は文書のチャンクから custom.css ファイルを生成し, そのファイルパスを html_document フォーマットの ccs オプションに渡す方法を示しています.

````
---
title: "コードチャンクから CSS ファイルを作成する"
output:
  html_document:
    css: custom.css
---

以下のチャンクは `custom.css` へ書き込まれ，ファイルは
Pandoc の変換時に使われます.

```{cat, engine.opts = list(file = "my_custom.css")}
h2 {
  color: blue;
}
```

## そしてこの見出しは青くなります
````

css コードチャンクのアプローチとこのアプローチの唯一の違いは, 前者が CSS コードをその場に書き込む, つまりコードチャンクのあるまさにその場所に書き込み, そしてそこは出力文書の <body> タグの内側ですが, 後者は CSS を出力文書の <head> の領域に書き込みます. 出力文書の見た目に実用上の違いは一切生じません.

15.6.2 LaTeX コードをプリアンブルに含める

6.1 節では, LaTeX コードをプリアンブルに追加する方法を紹介しました．これには 外部の .tex ファイルが必要でした．このファイルもまた Rmd から生成することができます．これがその例です．

````
---
title: "チャンクから .tex ファイルを作成する"
author: "Jane Doe"
documentclass: ltjsarticle
classoption: twoside
output:
  pdf_document:
    latex_engine: lualatex
    includes:
      in_header: preamble.tex
---

# どのように動作するか

出力する PDF のヘッダとフッタを定義するために
コードチャンクを `preamble.tex` に書き出しましょう．

```{cat, engine.opts=list(file = 'preamble.tex')}
\usepackage{fancyhdr}
\usepackage{lipsum}
\pagestyle{fancy}
\fancyhead[CO,CE]{これは fancy header}
\fancyfoot[CO,CE]{そしてこれは fancy footer}
\fancyfoot[LE,RO]{\thepage}
\fancypagestyle{plain}{\pagestyle{fancy}}
```

\lipsum[1-15]
````

```
# さらに適当なコンテンツ

\lipsum[16-30]
```

上記の cat コードチャンク内の LaTeX コードで, PDF 文書のヘッダとフッタを定義しました. フッタに著者名も表示したいなら, 別の cat コードチャンクにオプション engine.opts = list(file = 'preamble.tex', append = TRUE) と code = sprintf('\\fancyfoot[LO,RE]{%s}', rmarkdown::metadata$author) を付けることで preamble.tex に著者情報を追加することができます. このチャンクの動作を理解するには, この節の最初の方で紹介した engine.opts が base::cat() に渡されるということを思い出してください. つまり append = TRUE は cat() に渡されます. そして チャンクオプション code はこの後の 16.2 節を読めば理解できるでしょう.

15.6.3 YAML データをファイルに書き込みつつ表示する

cat コードチャンクの中身はデフォルトでは出力文書に表示されません. 中身を書き出した後で表示もしたいならば, チャンクオプション class.source に言語名を指定してください. 言語名はシンタックスハイライトに使われます. 以下の例では, 言語名を yaml に指定しています.

```
```{cat, engine.opts=list(file='demo.yml'), class.source='yaml'}
a:
  aa: "something"
  bb: 1
b:
  aa: "something else"
  bb: 2
```
```

その出力を以下に表示し, そしてファイル demo.yml としても生成します.

```
01 a:
02   aa: "something"
```

```
  bb: 1
b:
  aa: "something else"
  bb: 2
```

ファイル `demo.yml` が実際に生成されたことを示すには, **yaml** パッケージ (J. Stephens Simonov, et al., 2020) で読み込んでみることができます.

```
xfun::tree(yaml::read_yaml("demo.yml"))
```

```
## List of 2
##  ¦-a:List of 2
##  ¦  ¦-aa: chr "something"
##  ¦  ¦-bb: int 1
##  ¦-b:List of 2
##     ¦-aa: chr "something else"
##     ¦-bb: int 2
```

15.7 SAS コードを実行する

あなたは `sas` エンジン で SAS (https://www.sas.com) を実行するかもしれません. あなたの環境変数 `PATH` に SAS の実行ファイルがあることを確認するか, (`PATH` の意味を知らないなら) チャンクオプション `engine.path` に実行ファイルのフルパスを与える必要があります. 例えば `engine.path = "C:\\Program Files\\SASHome\\x86\\SASFoundation\\9.3\\sas.exe"` のように. 以下は “Hello World” を表示する例です.

````
```{sas}
data _null_;
put 'Hello, world!';
run;
```
````

15.8 Stata コードを実行する

Stata をインストールしているなら, stata エンジンで Stata のコードを実行できます. stata 実行ファイルが環境変数 PATH から見つけられないかぎり, チャンクオプション engine.path を介して実行ファイルのフルパスを指定する必要があります. 例えば engine.path = "C:/Program Files (x86)/Stata15/StataSE-64.exe" のように. 以下は簡単な例です.

````
```{stata}
sysuse auto
summarize
```
````

knitr の stata エンジンの機能はかなり限定的です. Doug Hemken が **Statamarkdown** パッケージでこれを実質的に拡張しており, GitHub の https://github.com/Hemken/Statamarkdown で利用可能です. "Stata R Markdown" でオンライン検索することでパッケージのチュートリアルを見つけられるでしょう.

15.9 Asymptote でグラフィックを作成する

Asymptote (https://asymptote.sourceforge.io) はベクタグラフィックのための強力な言語です. Asymptote をインストール済みなら (インストールの説明はウェブサイトを見てください) asy エンジンを使い R Markdown に Asymptote のコードを書き実行することもできます. 以下はそのリポジトリ https://github.com/vectorgraphics/asymptote からコピーした例で, 出力を図 15.1 に示します.

```
import graph3;
import grid3;
import palette;
settings.prc = false;

currentprojection=orthographic(0.8,1,2);
size(500,400,IgnoreAspect);
```

```
08
09 real f(pair z) {return cos(2*pi*z.x)*sin(2*pi*z.y);}
10
11 surface s=surface(f,(-1/2,-1/2),(1/2,1/2),50,Spline);
12
13 surface S=planeproject(unitsquare3)*s;
14 S.colors(palette(s.map(zpart),Rainbow()));
15 draw(S,nolight);
16 draw(s,lightgray+opacity(0.7));
17
18 grid3(XYZgrid);
```

PDF 出力に対しては追加の LaTeX パッケージが必要であることに注意してください．そうでないとこのようなエラーが出ることでしょう．

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

このようなエラーが発生したなら，欠けている LaTeX パッケージのインストール方法を 1.3 節で確認してください．

上記の `asy` チャンクでは，`settings.prc = false` という設定を使いました．この設定がないと Asymptote は PDF 出力時にインタラクティブな 3D グラフィックを表示してしまいます．しかしインタラクティブなグラフィックは Acrobat Reader でのみ見ることができます．Acrobat Reader を使用しているなら，グラフを操作できます．例えば図 15.1 ではマウス操作で 3D 平面を回転できます．

15.9.1 R でデータを生成し Asymptote に読み込ませる

ここでは，最初に以下の R コードチャンクのように，R で生成したデータを CSV ファイルに保存します．

```
01 x <- seq(0, 5, l = 100)
02 y <- sin(x)
03 writeLines(paste(x, y, sep = ","), "sine.csv")
```

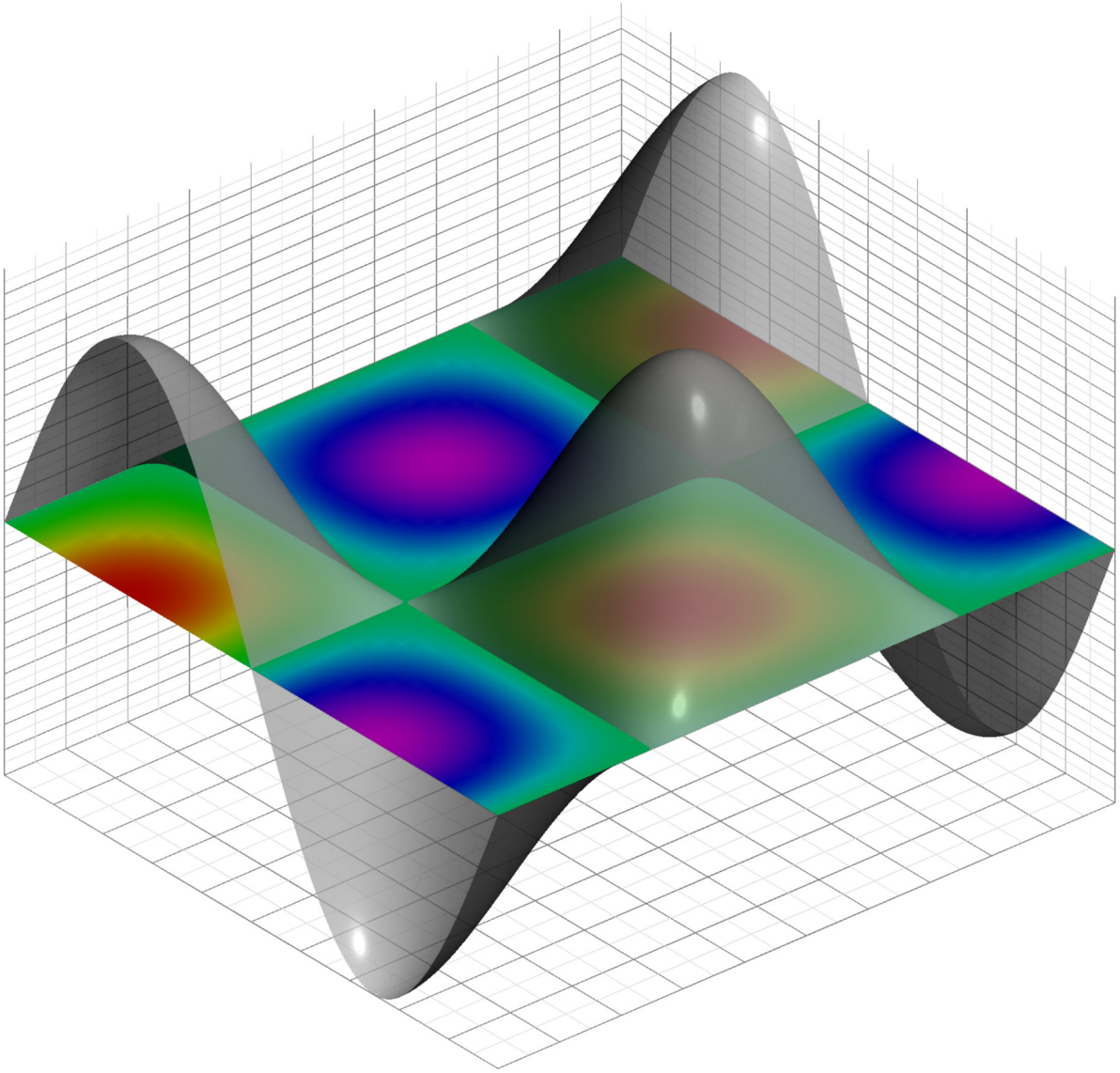


図 15.1: Asymptote で作成した 3D グラフィック

それから Asymptote でこれを読み込み, データに基づいたグラフを描画し図 15.2 に示します. 以下が asy コードチャンクです.

```
01 import graph;
02 size(400,300,IgnoreAspect);
03 settings.prc = false;
04
05 // import data from csv file
```

```
file in=input("sine.csv").line().csv();
real[][] a=in.dimension(0,0);
a=transpose(a);

// generate a path
path rpath = graph(a[0],a[1]);
path lpath = (1,0)--(5,1);

// find intersection
pair pA=intersectionpoint(rpath,lpath);

// draw all
draw(rpath,red);
draw(lpath,dashed + blue);
dot("$\delta$",pA,NE);
xaxis("$x$",BottomTop,LeftTicks);
yaxis("$y$",LeftRight,RightTicks);
```

15.10 Sass/SCSS で HTML ページをスタイリングする

Sass (https://sass-lang.com) は CSS を拡張した言語で, 基本的な CSS で行っていた のよりはるかに柔軟な方法でルールを作成できます. これを学ぶことに関心があるなら, 公式ドキュメントを見てください.

R パッケージの **sass** (Cheng Mastny, et al., 2021) は SaSS を CSS にコンパイルするのに使用できます. **sass** パッケージに基づいて, **knitr** はコードチャンクを CSS にコンパイルするため 2 つの言語エンジン, sass and scss を読み込みます. Sass と SCSS の構文は互いに対応しているためです. 以下はチャンクヘッダが ```{scss} である scss コードチャンクです.

```
$font-stack: "HGS創英角ﾎﾟｯﾌﾟ体", "Comic Sans MS", cursive, sans-serif;
$primary-color: #00FF00;

.book.font-family-1 {
```

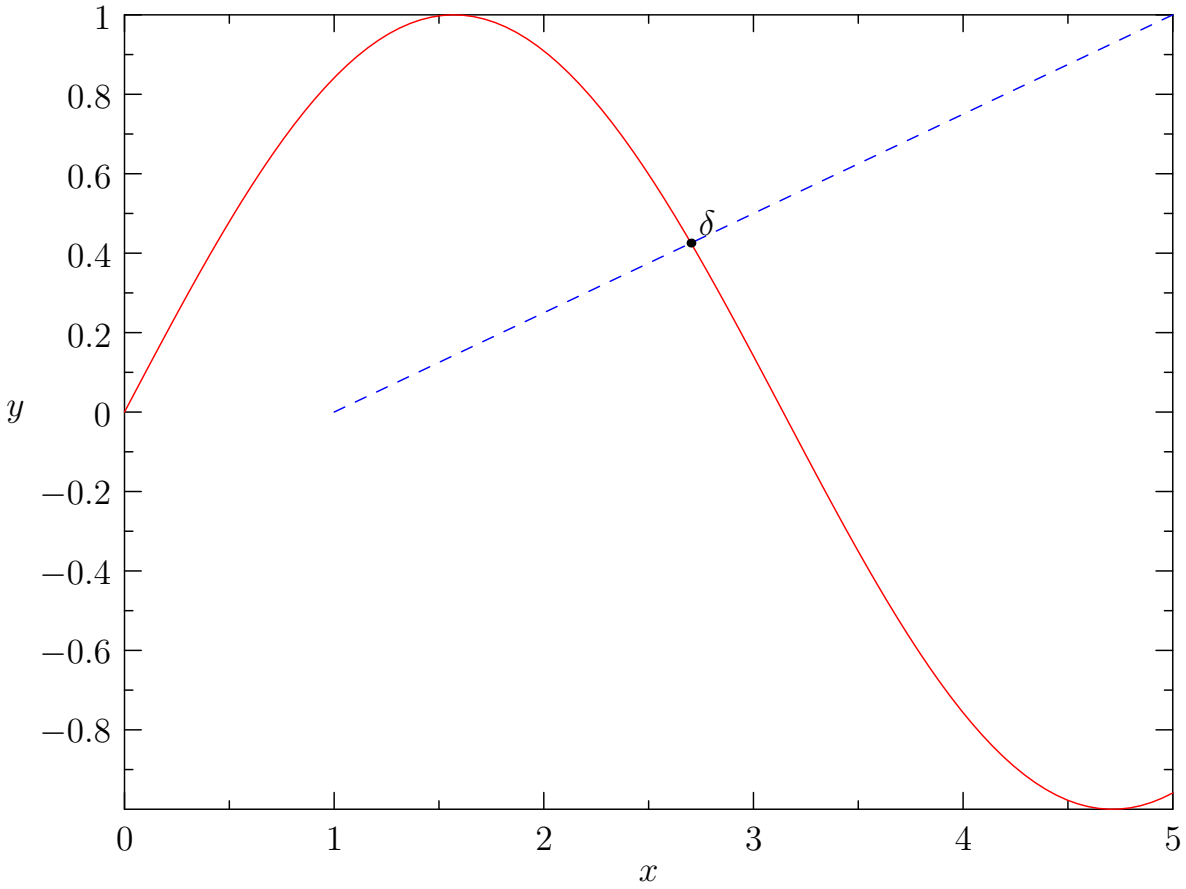


図 15.2: R からデータを渡し Asymptote でグラフを描く

```
  font: 100% $font-stack;
  color: $primary-color;
}
```

sass エンジンも使うことができます．Sass 構文は SCSS 構文とわずかに異なります．例えばこのように．

```
```{sass}
$font-stack: "HGS創英角ﾎﾟｯﾌﾟ体", "Comic Sans MS", cursive, sans-serif
$primary-color: #00FF00

.book.font-family-1
```
```

```
  font: 100% $font-stack
  color: $primary-color
```
```

あなたがこのセクションの HTML 版[*3]を読んでいるなら, このページのフォントが Comic Sans に変化したことに気付くでしょう. これには驚いたかもしれませんが, パニックにならないでください, あなたは脳卒中になどなっていません[*4].

sass/scss コードチャンクは sass::sass() 関数によってコンパイルされます. 現在はチャンクオプション engine.opts で CSS コードの出力スタイルをカスタマイズできます. 例えば engine.opts = list(style = "expanded") のように. デフォルトのスタイルは “compressed” です. これが何を意味するのか自信がないなら, ?sass::sass_options のヘルプを参照し, output_style 引数の項目を探してください.

[*3] https://bookdown.org/yihui/rmarkdown-cookbook/eng-sass.html
[*4] https://twitter.com/andrewheiss/status/1250438044542361600
```

第16章

プロジェクトを管理する

大きなプロジェクトやレポートの作業をしている時には, 1 つの R Markdown 文書の中に全てのテキストとコードを置かず, 代わりに小さな単位に分けたものをうまくまとめたいでしょう. この章では, R Markdown と関係する複数のファイルをまとめる方法を紹介します.

16.1 外部の R スクリプトを実行する

もし R Markdown 文書に大量のコードがあるなら, 以下の例のように, コードの一部を外部 R スクリプトに配置し, `source()` か `sys.source()` 経由で実行するよう検討してください.

````
```{r, include=FALSE}
source("your-script.R", local = knitr::knit_global())
# または sys.source("your-script.R", envir = knitr::knit_global())
```
````

お薦めするやり方は, `sys.source()` の `envir` 引数または `source()` の `local` 引数を明示的に使い, コードが確実に適正な環境で評価されるようにすることです. これらのデフォルト値は適切な環境でないかもしれません. 間違った環境で変数を作成してしまい, その後のチャンクでオブジェクトが見つからず驚くということになりかねません.

それから, R Markdown 文書の中で, これらのスクリプトで作成された, データや関数といったオブジェクトを使えるのです. このやり方は R Markdown 文書が簡潔になるだけでなく, R コードの開発がもっと便利になるという効果もあります. 例えば R コードのデバッグは, R Markdown より, ピュアな R スクリプトでやるほうがたいてい簡単です.

上記の例では include = FALSE を使っていることに注目してください．出力を一切表示させずにスクリプトの実行するだけにしたいからです．出力が欲しければこのチャンクオプションを削除するか，11.7 節で紹介したオプションを使って，隠したり表示したりを出力の種類の違いによって選択することもできます．

16.2 外部スクリプトをチャンク内で読み込む

16.1 節で紹介した source() の方法には欠点があります．それはデフォルトではソースコードを見ることができないという点です．source(..., echo = TRUE) を使うことはできますが，ソースコードのシンタックスがきちんとハイライトされません．加えて 16.1 節で言及したように，source() の local 引数について注意を払う必要があります．この節ではこういった問題のない代わりになる方法を紹介します．

1 つでも外部スクリプトがあれば，基本的にはそれを読み込んで中身を，チャンクの code オプションに渡すことができます．code オプションは文字列ベクトルをとるので，それをコードチャンクの本文として扱えます．以下に少しだけ例をお見せします．

- code オプションはソースコードを文字列ベクトルとして取ることができます．これが例です．

````
```{r, code=c('1 + 1', 'if (TRUE) plot(cars)')}
```
````

- 外部ファイルを読み込むこともできます．

````
```{r, code=xfun::read_utf8('your-script.R')}
```
````

- ファイルを好きな数だけ読み込むこともできます．

````
```{r, include=FALSE}
read_files <- function(files) {
  unlist(lapply(files, xfun::read_utf8))
}
````
````

```
```

```{r, code=read_files(c('one.R', 'two.R'))}
```

他の言語のスクリプトも読み込めます. R Markdown で他の言語を使う方法は 15 章を確認してください. 以下に, もう少しだけ R 以外のコードの例をお見せします.

- Python スクリプトを読み込む.

```{python, code=xfun::read_utf8('script.py')}
```

- C++ ファイルを読み込む:

```{Rcpp, code=xfun::read_utf8('file.cpp')}
```

code オプションがあれば, お気に入りのエディタ使って複雑なコードを開発した上で, それを R Markdown 文書のコードチャンクに読み込ませるということができます.

16.3 外部スクリプトから複数のコードチャンクを読み込む (*)

16.2 節では, コードを単一のチャンクに読み込む方法を紹介しました. この節では外部スクリプトから複数のチャンクを読み取る方法を 1 つ紹介します. ポイントは, スクリプト内のコードにラベルを付ける必要がありますが, 同じラベルを R Markdown 文書のコードチャンクにも使用できるという点です. つまり外部スクリプトのコードを `knitr::read_chunk()` 関数を介して各コードチャンクに展開できるのです. スクリプトのブロックにラベルを付けるには, `## ----` の後にラベルを書きます (行の終わりにも好きな数のダッシュ記号を続けることができます). 例えば次のように, 1 つのスクリプトにはラベル付けされたブロックを複数含めることができます.

```
## ---- test-a --------
1 + 1

## ---- test-b --------
if (TRUE) {
  plot(cars)
}
```

上記のスクリプトのファイル名が `test.R` であるとします．R Markdown 文書ではこれを `knitr::read_chunk()` 関数で読み込み，コードチャンク内ではそのコードをラベルで使えます．これが例です．

外部スクリプトを読み込む

```
```{r, include=FALSE, cache=FALSE}
knitr::read_chunk('test.R')
```
```

これで，例えばこのようにコードを使用できる

```
```{r, test-a, echo=FALSE}
```

```{r, test-b, fig.height=4}
```
```

コードチャンクの副産物にも影響するというのが主な理由ですが `knitr::read_chunk()` を使っていることに注意してください．つまりこの関数を読み込んだコードチャンクがキャッシュされていないことを確認してください (この説明は 11.4 節参照)．

16.1, 16.2 節で紹介したように，この方法は別の環境でコード開発できるという柔軟性をもたらしてくれます．

16.4 子文書 (*)

R Markdown 文書が長過ぎると思った時は, 短い文書に分割することも考えると. そして, チャンクオプション child を使ってメイン文書に子文書として読み込ませましょう. child オプションは子文書のファイルパスを文字列ベクトルとして取ります. 例えばこのように.

```
```{r, child=c('one.Rmd', 'two.Rmd')}
```
```

knitr のチャンクオプションは任意の R コードから値を取ることができるので, child オプションの応用として条件付で文書を読み込ませる方法があります. 例えばあなたのレポートの中に, 上司が関心を持たないような技術的に詳細な補足文書があるなら, この変数を使えばその補足文書をレポートに含むかどうかを制御できます.

あなたのボスにレポートを読ませるなら `BOSS_MODE` を `TRUE` に変える

```
```{r, include=FALSE}
BOSS_MODE <- FALSE
```
```

条件付きで補足文書を読み込む

```
```{r, child=if (!BOSS_MODE) 'appendix.Rmd'}
```
```

あるいはまだ始まってないフットボールの試合の速報レポートを書いているなら, 試合結果に応じて異なる子文書を読み込むようにすることもできます. 例えば child = if (winner == 'ブラジル') 'ブラジル.Rmd' else 'ドイツ.Rmd' のように. これで試合 (ここではドイツ対ブラジル) が終わり次第すぐに, レポートを提出できます.

子文書をコンパイルする別の方法として, knitr::knit_child() 関数があります. この関数は R コードチャンク内部またはインライン R コードで呼び出せます. 例えばこのように.

```
```{r, echo=FALSE, results='asis'}
res <- knitr::knit_child('child.Rmd', quiet = TRUE)
cat(res, sep = '\n')
```
```

knit_child() 関数は knit された子文書の文字列ベクトルを返します．これは cat() とチャンクオプション results = "asis" を使ってメインの文書に還元することができます．

テンプレートとして子文書を使うこともできますし，毎回異なるパラメータを与えつつ何度でも knit_child() を呼び出すこともできます．以下の例では mtcars データの mpg を従属変数，そして残りの変数を説明変数として使った回帰分析を実行しています．

```
```{r, echo=FALSE, results='asis'}
res <- lapply(setdiff(names(mtcars), 'mpg'), function(x) {
  knitr::knit_child(text = c(
    '## "`r x`" への回帰',
    '',
    '```{r}',
    'lm(mpg ~ ., data = mtcars[, c("mpg", x)])',
    '```',
    ''
  ), envir = environment(), quiet = TRUE)
})
cat(unlist(res), sep = '\n')
```
```

上記の例を自己完結的なものにするために，knit_child() にファイルを入力するのではなく text 引数に R Markdown コンテンツを渡しました．もちろんファイルにコンテンツを書き出し，knit_child() にファイルパスを渡すこともできます．例えば以下のコンテンツを template.Rmd という名前のファイルに保存します．

```
## "`r x`" への回帰
```

````
```{r}
lm(mpg ~ ., data = mtcars[, c("mpg", x)])
```
````

そして代わりにファイルを knit します.

```
res <- lapply(setdiff(names(mtcars), 'mpg'), function(x) {
  knitr::knit_child(
    'template.Rmd', envir = environment(), quiet = TRUE
  )
})
cat(unlist(res), sep = '\n')
```

16.5 グラフ画像ファイルを残す

ほとんどの R Markdown 出力フォーマットはデフォルトで `self_contained = TRUE` オプションを使用しています. これは出力文書に R グラフを埋め込むので, 出力文書を閲覧する時の中間ファイルは必要ありません. その結果, グラフ画像のフォルダ (典型的には `_files` という接尾語があります) は Rmd 文書がレンダリングされた後に削除されます.

ときにはグラフ画像ファイルを残したいことがあります. 例えば学術誌の中には, 画像ファイルを別個に提出するよう著者に求めるものもあります. R Markdown ではこれらのファイルの自動削除を回避する 3 通りの方法があります.

1. 出力フォーマットがサポートしていれば, 以下のように `self_contained = FALSE` オプションを使う.

   ```
   output:
     html_document:
       self_contained: false
   ```

   しかし, この方法ではグラフ画像ファイルが出力文書に埋め込まれません. それを望まなければ, 次の 2 つの方法を検討しましょう.

2. 最低いずれか 1 つのコードチャンクでキャッシュ (11.4 節参照) を有効にする．キャッシュが有効な時は R Markdown は画像フォルダを削除しません．
3. 出力フォーマットがサポートしていれば, 以下のように `keep_md = TRUE` オプションを使用する．

```
output:
  word_document:
    keep_md: true
```

R Markdown に対し Markdown 中間出力ファイルを保存するよう指示した時, 同時に画像フォルダも保存されます．

16.6 R コードチャンク用の作業ディレクトリ

R コードチャンクの作業ディレクトリは, デフォルトでは Rmd 文書のあるディレクトリです．例えば Rmd ファイルのパスが `~/Downloads/foo.Rmd` であるなら, R コードチャンクが評価される作業ディレクトリは `~/Downloads/` になります．ということは, チャンク内で外部ファイルを相対パスで参照するとき, そのパスは Rmd ファイルのあるディレクトリからの相対パスであることを知っておくべきことを意味します．前述の Rmd ファイルの例では, コードチャンク内での `read.csv("data/iris.csv")` は `~/Downloads/data/iris.csv` から CSV ファイルを読み込むことを意味しています．

よく分からない時は, `getwd()` をコードチャンクに追加して文書をコンパイルし, `getwd()` の出力を確認できます．

時には他のディレクトリを作業ディレクトリとして使いたいこともあります．一般的な作業ディレクトリの変更方法は `setwd()` ですが, `setwd()` は R Markdown あるいは他の **knitr** ソース文書で一貫して使えるわけではないことに注意してください．これは `setwd()` が現在のコードチャンクに限って動作し, 作業ディレクトリはこのコードチャンクが評価された後に元に戻ることを意味します．

全てのコードチャンクに対して作業ディレクトリを変更したい場合, 文書の冒頭で `setup` コードチャンクを設定できます．

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

```
knitr::opts_knit$set(root.dir = '/tmp')
```

これは以降の全てのコードチャンクの作業ディレクトリを変更します.

RStudio を使用しているなら, 作業ディレクトリをメニュの `Tools -> Global Options -> R Markdown` からも選択できます (図 16.1 参照). デフォルトの作業ディレクトリは Rmd ファイルのディレクトリで, 他に 2 つの選択肢があります. “Current” オプションで R コンソールの現在の作業ディレクトリを使うか, “Project” オプションで Rmd ファイルが入っているプロジェクトのルートディレクトリを作業ディレクトリとして使うこともできます.

RStudio では, 図 16.2 で見せるように, 個別の Rmd 文書をそれぞれ固有の作業ディレクトリで knit することもできます. “Knit Directory” を変更し “Knit” ボタンをクリックした後で, **knitr** は新しい作業ディレクトリを使ってコードチャンクを評価します. これらの全ての設定は既に言及した `knitr::opts_knit$set(root.dir = ...)` に集約されています. よってあなたがこれまでの選択肢のいずれにも満足しないのなら, `knitr::opts_knit$set()` を使いご自分でディレクトリを指定できます.

作業ディレクトリに関して完全に正しい選択というものはありません. それぞれに長所と短所があります.

- (**knitr** のデフォルト) Rmd 文書のディレクトリをコードチャンクの作業ディレクトリとして使うなら, ファイルパスは Rmd 文書からの相対パスだと想定していることになります. これは ウェブブラウザで相対パスを扱うのと似ています. 例えば `https://www.example.org/path/to/page.html` という HTML ページでの画像 `<img src="foo/bar.png" />` に対して, ウェブブラウザが `https://www.example.org/path/to/foo/bar.png` から画像を取得するのと似ています. 言い換えるなら, 相対パス `foo/bar.png` は HTML ファイルのあるディレクトリ `https://www.example.org/path/to/` からの相対位置です.
  このアプローチの利点は Rmd ファイルを Rmd ファイルが参照しているファイルと**一緒に**, 相対的な位置関係を保っている限りどこへでも自由に移動できることです. 上記の HTML ページと画像の例では, `page.html` と `foo/bar.png` を `https://www.example.org/another/path/` へ一緒に移動させることができます. そしてあなたは `<img />` の `src` 属性の相対パスを更新する必要はありません.
  Rmd 文書の相対パスを「Rmd ファイルからの相対位置」とは対照的に「R コンソールの作業ディレクトリからの相対位置」と考えるのを好むユーザもいます. よって **knitr** のデフォルトディレクトリは混乱を招きます. 私が **knitr** を設計する際に R コンソールの作業ディレクトリをデフォルトで使わないようにした理由は, ユーザがいつでも `setwd()` で作業ディ

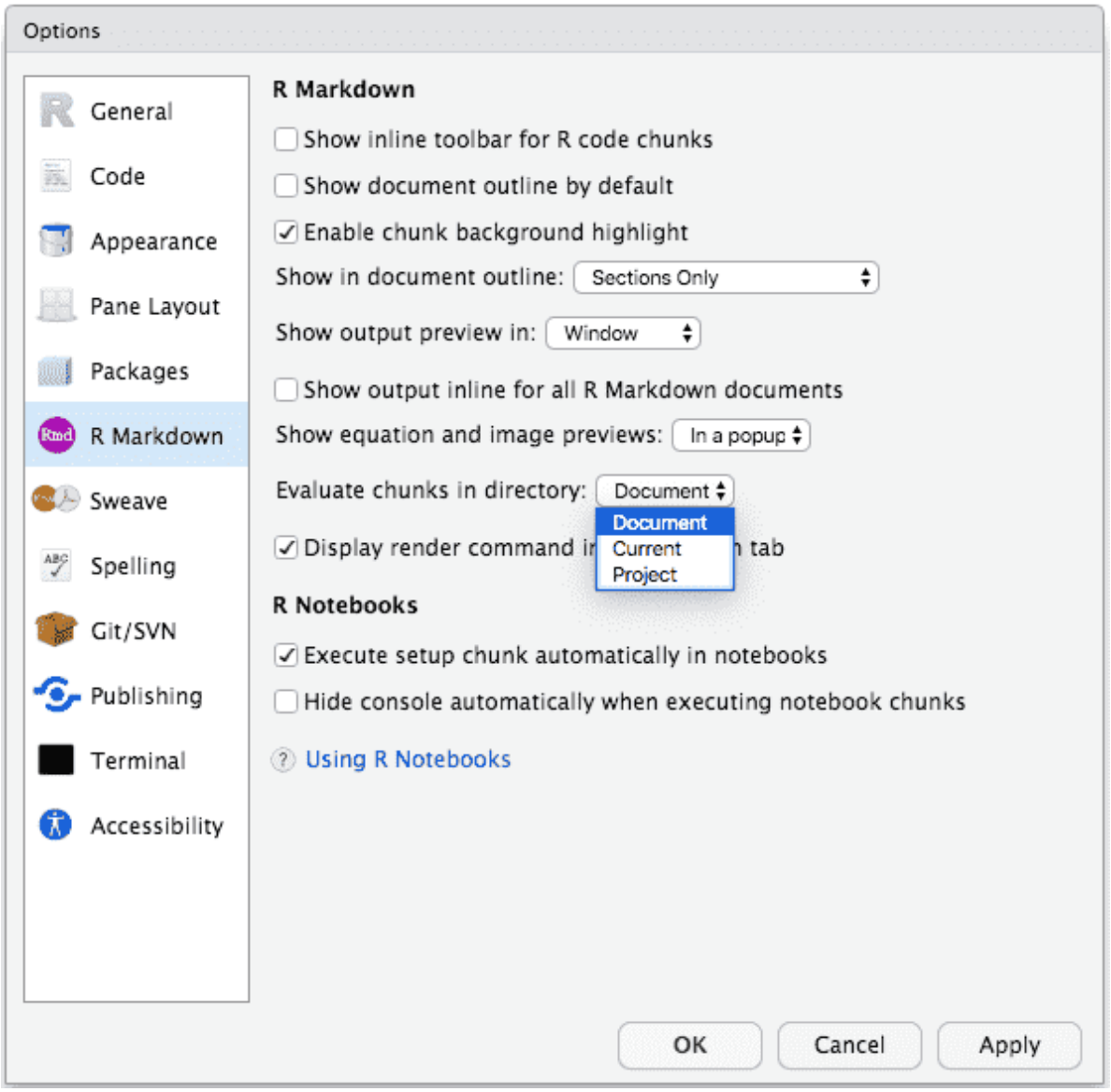


図 16.1: R Studio で R Markdown 文書用のデフォルトの作業ディレクトリを変更する

レクトリを変更したければできてしまうからでした．この作業ディレクトリが安定している保証はありません．毎度のようにユーザが `setwd()` をコンソールで呼び出すと，Rmd 文書内のファイルパスが無効になるリスクがあります．ファイルパスが Rmd ファイルの制御の手から離れて `setwd()` という外部要因に依存しているというのは不自然なことでしょう．相対パスを考慮する際に，Rmd ファイルを「宇宙の中心」として扱えば，Rmd ファイル内にあるパスはもっと安定するでしょう．

その上，あなたが相対パスを考慮するのが難しすぎて嫌だと言うなら，図 16.3 のように RStudio 上で自動補完機能を使ってファイルパスを入力することもできます．RStudio は Rmd ファイルからの相対パスを補完しようと試みます．

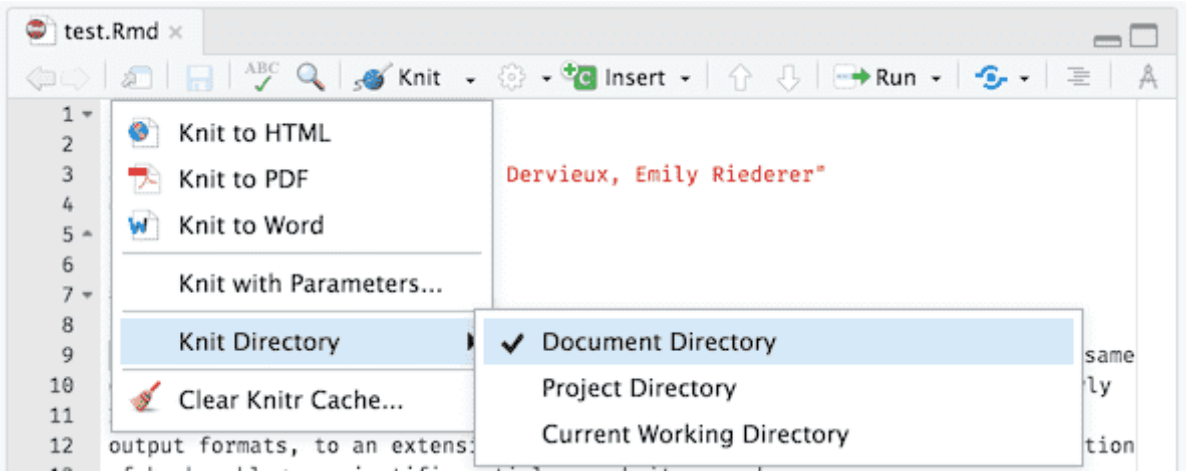


図 16.2: RStudio の他の使用可能な作業ディレクトリで Rmd 文書を knit する

- R コンソールの作業ディレクトリはプログラミング的あるいは対話的に文書を knit するのに良い選択になりうるでしょう. 例えばループ中に文書を複数回 knit し, そこで毎回で異なる作業ディレクトリを使い, 各々のディレクトリ内の異なるデータファイル (ファイル名は同じとします) を読み込むこともできます. この種の作業ディレクトリは **ezknitr** パッケージ (Attali, 2016) で推奨されており, 実は `knitr::opts_knit$set(root.dir)` を使って **knitr** のコードチャンクの作業ディレクトリを変更しています.
- プロジェクトディレクトリを作業ディレクトリとして使うことには明確な前提が要求されます. そもそもプロジェクト (例えば RStudio のプロジェクトか, バージョン管理プロジェクト) を使わなければならないということです. この点はアプローチにとっての欠点となりえます. この種の作業ディレクトリを使う利点はあらゆる Rmd 文書内の全ての相対パスがプロジェクトのルートディレクトリからの相対パスになることです. よってプロジェクト内で Rmd ファイルがどこにあるかを考えたり, 他のファイルの場所に対応して調整したりする必要はありません. この種の作業ディレクトリは **here** パッケージ (Müller, 2020) で推奨されており, このパッケージでは渡された相対パスを解決し絶対パスを返す `here::here()` 関数を提供しています (相対パスはプロジェクトのルートからの相対であることを忘れないでください). 欠点となるのは, 参照されているファイルを Rmd ファイルとともにプロジェクト内の他の場所に移動させた時に, Rmd 文書内の参照パスを更新する必要があることです. Rmd ファイルを他の人と共有する時は, プロジェクト全体も共有しなければなりません.
  これらの種類のパスは HTML でのプロトコルやドメインのない絶対パスと似ています. 例えば `https://www.example.org/path/to/page.html` というページの画像 `<img src="/foo/bar.png" />` はウェブサイトのルートディレクトリ以下の画像を参照しています. つまり `https://www.example.org/foo/bar.png` です. 画像の `src` 属性の先頭の `/` はウェブサイトのルートディレクトリを表しています. HTML の絶対パスと相対パスについてもっと

学びたい (あるいはもっと混乱したい) なら, **blogdown** 本の Appendix B.1[*1] (Xie Hill, and Thomas, 2017) を見てください.

作業ディレクトリのうんざりする問題は, ほとんどの場合, 相対パスに対処している時に抱く「**何に対して相対的なの?**」という疑問に端を発します. 既に言及したように, いろいろな人がいろいろな好みを持っており, 完全に正しい回答はありません.

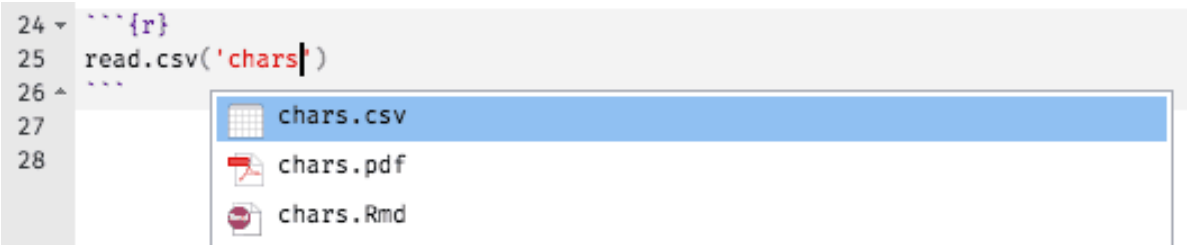


図 16.3: RStudio 上で Rmd 文書のファイルパスを自動補完する

16.7 R パッケージのビネット

R パッケージの開発を経験したか, プロジェクトで自作関数の明瞭なドキュメントや厳格なテストが要求されたなら, プロジェクトを R パッケージと結びつけてみてはどうでしょうか. R パッケージの作り方が分からないなら, RStudio IDE でメニューバーの `File -> New Project` をクリックし, プロジェクトの種類に R パッケージを選ぶことで簡単に始めることができます.

プロジェクトの管理に R パッケージを使うことには多くの利益があります. 例えば `data/` フォルダにデータを置き, `R/` に R コードを書き, 例えば **roxygen2** パッケージ (Wickham Danenberg, et al., 2021) を使用して, ドキュメントを `man/` に生成し, `test/` には単体テストを追加できます. R Markdown のレポートなら `vignette/` にパッケージのビネットとして書くことができます. ビネット内ではデータセットを読み込みパッケージ内の関数を呼び出せます. (`R CMD build` コマンドか RStudio で) パッケージをビルドする時に, ビネットは自動でコンパイルされます.

R Markdown でパッケージのビネットを作成するのに最も簡単な方法は, RStudio のメニュー `File -> New File -> R Markdown -> From Template` を経由するものです (図 16.4 参照). それから **rmarkdown** パッケージから "Package Vignette" を選択すると, ビネットのテンプレートが得られます. テンプレートの, タイトル・著者・その他のメタデータを変更したら, レポートの本文を書き始めましょう.

他の方法としては, **usethis** (Wickham and Bryan, 2021) をインストールし `usethis::use_vignette()` 関数を使ってビネットのスケルトンを作成できます. 以下はパッケージのビネットの YAML フロ

[*1] https://bookdown.org/yihui/blogdown/html.html

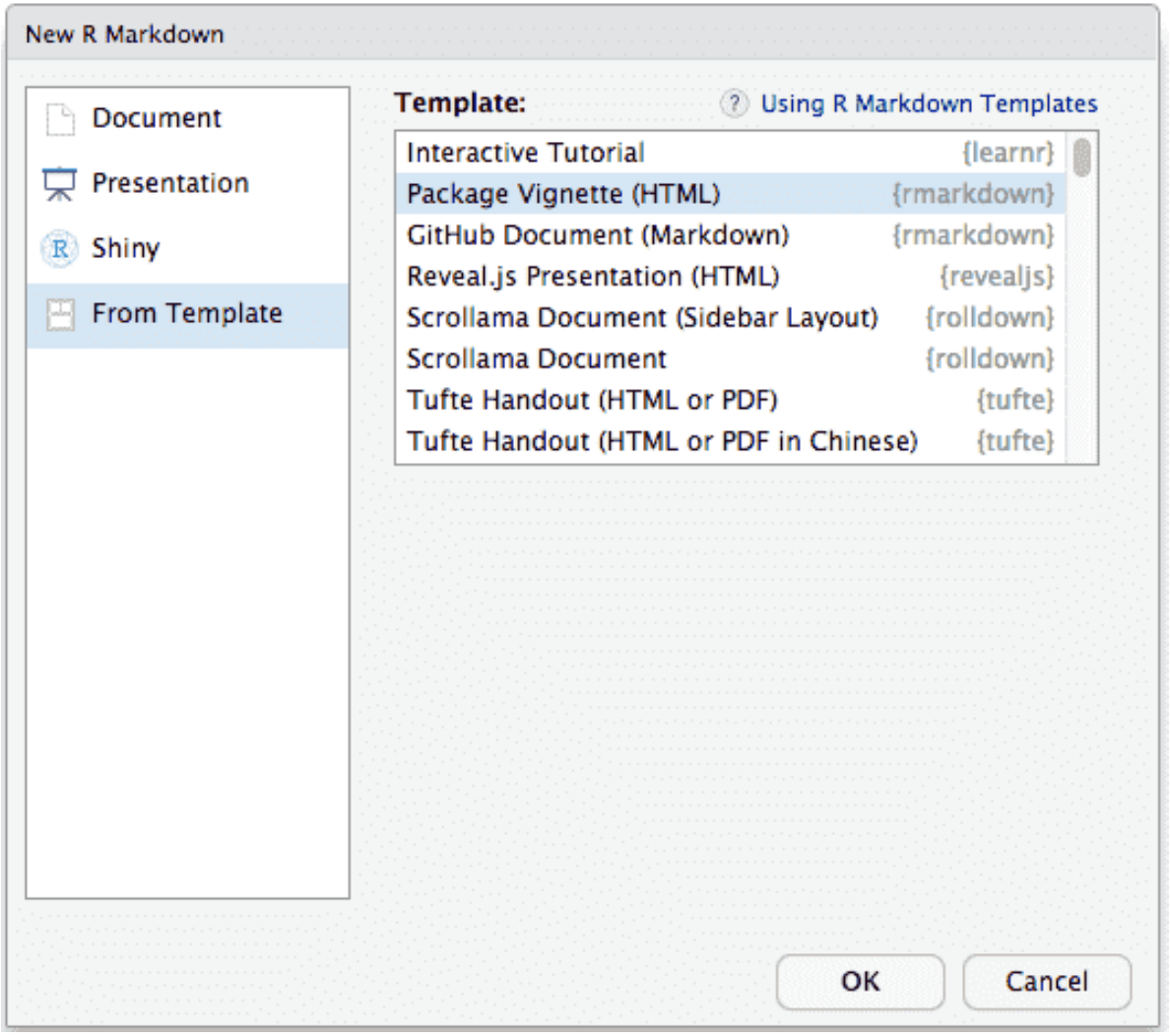


図 16.4: RStudio でパッケージのビネットを作成する

ントマターの典型的な姿です.

```
---
title: "ビネットのタイトル"
author: "ビネットの著者"
output: rmarkdown::html_vignette
vignette: >
  %\VignetteIndexEntry{ビネットのタイトル}
  %\VignetteEngine{knitr::rmarkdown}
  %\VignetteEncoding{UTF-8}
```

```
---
```

`title` フィールドと `\VignetteIndexEntry{}` コマンドの両方で, ビネットのタイトルを変更しなければならないことに注意してください. 上記のビネット情報の他にも, パッケージの `DESCRIPTION` ファイルにさらに 2 つすべきことがあります.

1. `DESCRIPTION` ファイルに `VignetteBuilder: knitr` を指定する.
2. `DESCRIPTION` ファイルに `Suggests: knitr, rmarkdown` を追加する.

ビネット出力フォーマットは HTML でなくてもかまいません. PDF でも可能なので, `output: pdf_document` も使えます. 他の出力フォーマットでも `beamer_presentation` や `tufte::tufte_html` のような, HTML か PDF を作成するものであればどれも大丈夫です. ただし, 現時点では R は HTML と PDF のビネットのみを認識します.

16.8 R パッケージの R Markdown テンプレート

16.7 節の図 16.4 では, 編集可能なパッケージビネットの HTML テンプレートを **rmarkdown** パッケージから取得する手順を表しています. この R Markdown ファイルには R パッケージのビネットを作るに当たっての適切なメタデータが詰め込まれています.

同様に, どのような R パッケージであっても, R Markdown テンプレートを同梱して, (この図で示しているように) ユーザが RStudio IDE を通してアクセスしたり, あるいはどのプラットフォーム上でも `rmarkdown::draft()` 関数でアクセスできるようにするとよいでしょう.

16.8.1 テンプレートのユースケース Template use-cases

テンプレートはカスタマイズされた文書構造・スタイル・コンテンツを共有するのに便利な方法です. 多くのすばらしい例が世に出回っています.

多くのテンプレートは入力済みのメタデータによって文書構造とスタイルを追加しています. すでに **rmarkdown** パッケージの (HTML の) ビネットテンプレートを例としてお見せしました. 同様に, **rmdformats** パッケージ (Barnier, 2021) では様々なカスタムスタイル関数を `output` オプションに渡すテンプレートがいくつも提供されています.

その他のテンプレートではパッケージで必要になる文書の構文を例示しているものがあります. 例えば **pagedown** パッケージ (Xie Lesur, et al., 2021) はポスター・履歴書・その他のページレイアウト用に無数のテンプレートを同梱しています. 同様に **xaringan** パッケージ (Xie, 2021f) の

忍者風のプレゼンテーションテンプレートは様々なスライドフォーマットのオプションに対する構文を例示しています.

テンプレートによってはパッケージの機能と構文を例示していることもあります. 例えば **flexdashboard** パッケージ (Richard Iannone JJ Allaire, and Borges, 2020) と **learnr** (Schloerke JJ Allaire, and Borges, 2020) パッケージには, サンプルのダッシュボートとチュートリアルをそれぞれ作成するために, パッケージから関数を呼び出すコードチャンク付きのテンプレートを同梱しています.

同様に, テンプレートには定型的なコンテンツの雛形を含んでいるものもあります. 例えば **rticles** パッケージ (JJ Allaire Xie Dervieux, et al., 2021) にはたくさんテンプレートがあって, R Markdown 出力を様々な学術誌で要求されるスタイルとガイドラインに沿って調整できます. コンテンツの雛形は, 四半期レポートを作成するチームのようなところで組織的に設定する際にも便利です.

16.8.2 テンプレートの準備

usethis パッケージ (Wickham and Bryan, 2021) にはテンプレートの作成に役に立つ関数があります. `usethis::use_rmarkdown_template("テンプレート名")` を実行すると, 必要なディレクトリ構造とファイルが自動で作成されます. テンプレート名は自分で付けましょう.

代わりに自分のテンプレートを手動で準備したいなら, `inst/rmarkdown/templates` のサブディレクトリを作成してください. このディレクトリ内に, 少なくとも 2 つのファイルを保存する必要があります.

1. `template.yaml` という名前のファイル. これは RStudio IDE に対して, 人間が判読できるテンプレートの名称などの基本的なメタデータを与えます. 例えば以下のように 最低でも, このファイルは `name` と `description` フィールドを持っているべきです.

```
name: テンプレートの例
description: このテンプレートが何をするか
```

   テンプレートが選択された時に新しいディレクトリを作成してほしいなら, `create_dir: true` を含めることもできます. 例えば **learnr** パッケージのテンプレート[*2]は `create_dir: true` を設定しており, 一方で **flexdashboard** パッケージのテンプレート[*3] はデフォルト

[*2] https://github.com/rstudio/learnr/blob/master/inst/rmarkdown/templates/tutorial/template.yaml
[*3] https://github.com/rstudio/flexdashboard/blob/master/inst/rmarkdown/templates/flex_dashboard/template.

の `create_dir: false` を使用しています．これらのテンプレートを RStudio で開いてみると，様々なユーザの意図に気付くはずです．

2. `skeleton/skeleton.Rmd` 内に保存された R Markdown 文書ファイル．これは R Markdown 文書に挿入したいどのようなコンテンツでも含めることができます．

オプションとして，`skeleton` フォルダにはスタイルシートや画像といった，作ったテンプレートで使われる追加のリソースを含めることができます．これらのファイルはテンプレートとともにユーザのコンピュータに読み込まれます．

R Markdown のカスタムテンプレートを作るためのさらに詳細な情報は，RStudio Extensions[*4] と *R Markdown Definitive Guide* (Xie J.J. Allaire, and Grolemund, 2018) の Document Templates の章[*5] を参照してください．

16.9 bookdown で本や長いレポートを書く

bookdown パッケージ (Xie, 2021a) は複数の R Markdown 文書で構成される長い文書を作成できるように設計されています．例えば本を執筆したいなら，章ごとに別々の Rmd ファイルに書き，**bookdown** を使ってこれらのファイルを本にコンパイルできます．

RStudio ユーザーにとって最も簡単な始め方は，図 16.5 にあるように，IDE 上で `File -> New Project -> New Directory -> Book Project using bookdown` を選んで **bookdown** プロジェクトを作成することです．

RStudio を使っていないか，コンソールから作業するのが好きなら，`bookdown:::bookdown_skeleton('本のディレクトリ')` 関数を呼べば同じものが作れます．

使用法を実演するために，同じディレクトリに 3 つのファイルを含めた最低限の例を用意しました．

```
directory
  ¦- index.Rmd
  ¦- 01-導入.Rmd
  ¦- 02-分析.Rmd
```

以下に各ファイルの中身とそれぞれの役目を示します．

---

yaml

[*4] https://rstudio.github.io/rstudio-extensions/rmarkdown_templates.html

[*5] https://bookdown.org/yihui/rmarkdown/document-templates.html

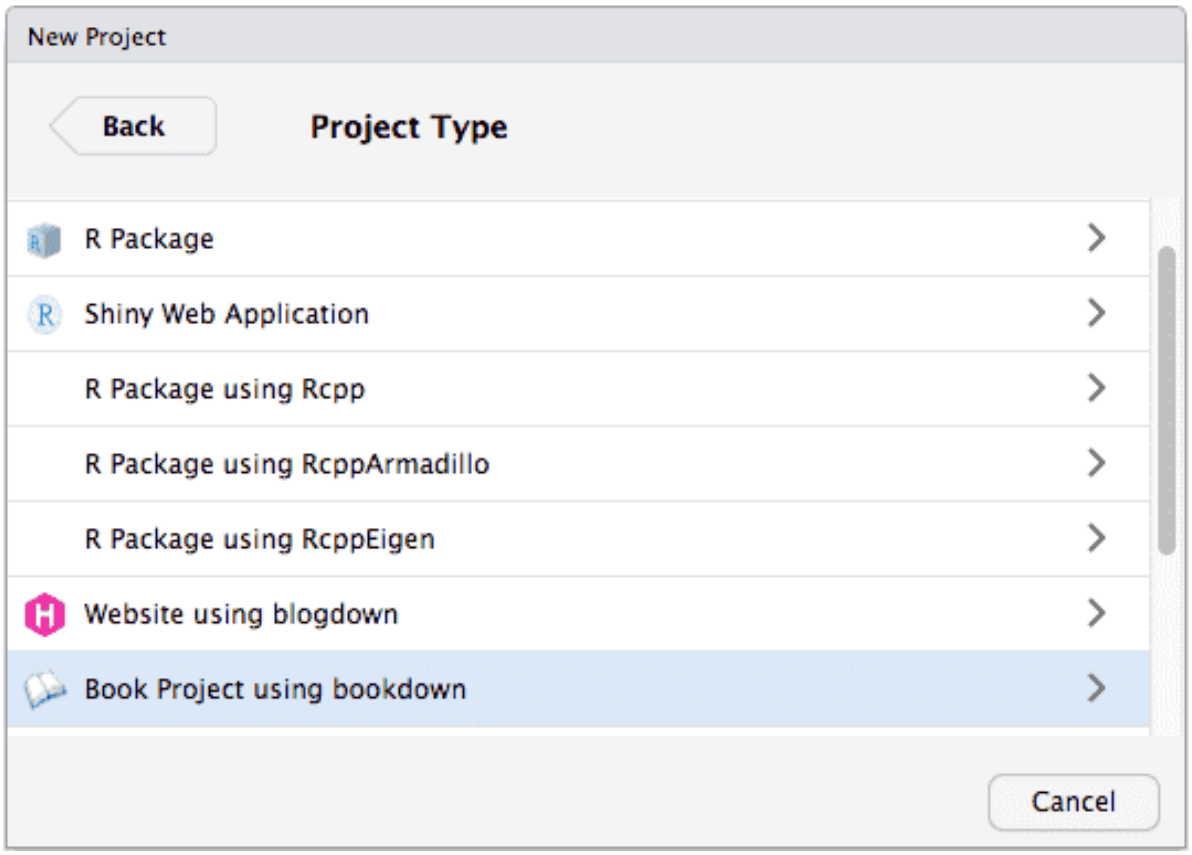


図 16.5: RStudio で bookdown プロジェクトを作成する

- **index.Rmd**:

```
---
title: "最低限の bookdown プロジェクト"
site: bookdown::bookdown_site
output: bookdown::gitbook
---

# はじめに {-}

なにか書く
```

最初のファイルは典型的には `index.Rmd` と呼ばれます．YAML フロントマターを与える唯一の Rmd ファイルとなるべきです．また, 特殊な YAML フィールド, `site: bookdown::bookdown_site`

を含めて, **bookdown** を使うべきことを **rmarkdown** に知らせることで, 単一の Rmd ファイルをレンダリングするのではなく, 全ての Rmd ファイルをビルドさせます. `bookdown::gitbook` ・ `bookdown::pdf_book` ・ `bookdown::word_document2` ・ `bookdown::epub_book` といったどのような **bookdown** 出力フォーマットでも使えます.

次の 2 つの Rmd ファイルは 2 つの章になります.

- **01-導入.Rmd:**

```
# 第 1 章

これは第 1 章です.
```

- **02-分析.Rmd**:

```
# 第 2 章

これは第 2 章です.
```

これらの Rmd ファイルをレンダリングするためには, `rmarkdown::render()` の代わりに `bookdown::render_book('index.Rmd')` を呼ぶべきです. その内部では, デフォルトで **bookdown** が全ての Rmd ファイルを 1 つの Rmd に結合し, コンパイルします. ファイルは名前順に結合されます. 上記の例でファイル名の頭に数字を付けたのはそれが理由です.

bookdown プロジェクトをカスタマイズできる設定は多くあります. **bookdown** のより包括的な概要として, **rmarkdown** 本 (Xie J.J. Allaire, and Grolemund, 2018) の Chapter 18 を読んでください. 完全なドキュメントは **bookdown** 本 (Xie, 2016) になります.

16.10 blogdown でウェブサイトを構築する

R Markdown に基づいたウェブサイトを構築したいなら, **blogdown** パッケージ (Xie Dervieux, and Presmanes Hill, 2021) の使用を検討しするとよいでしょう. 最も簡単な始め方は図 16.5 にあるように RStudio メニューから `File -> New Project -> New Directory -> Website using blogdown` を選ぶことです. これまで **blogdown** を使ったことがないのなら, ダイアログボックスのデフォルト設定を使うとよいでしょう. そうでないなら, ウェブサイトのテーマのよ

うな項目をカスタマイズできます．RStudio を使用していないのなら，空のディレクトリで `blogdown::new_site()` 関数を呼び出せば，新しいウェブサイトが作れます．

ウェブサイトのプロジェクトには Rmd 文書をいくつ含めてもかまいません．これらは通常のページか，ブログの記事にできます．あなたのウェブサイトに表示されるものは自動的かつ動的に生成されるので，R Markdown があれば簡単に自分のウェブサイトを管理できるようになります．

ウェブサイトの管理の基本的なワークフローとこのパッケージの概要については，**blogdown** 本 (Xie Hill, and Thomas, 2017) の Chapter 1[*6] を読むことをお薦めします．

[*6] https://bookdown.org/yihui/blogdown/get-started.html

第17章

ワークフロー

この章では R Markdown プロジェクトの運用のみならず個別の R Markdown 文書で作業する際の豆知識を紹介します．*R for Data Science*[*1] (Wickham and Grolemund, 2016a) の Chapter 30[*2] も確認するとよいでしょう．ここには (R Markdown 文書を含む) 分析ノートの使用に関する豆知識が簡単に紹介されています．Nicholas Tierney も *R Markdown for Scientists.*[*3] でワークフローについて議論しています．

17.1 RStudio のキーボード・ショートカットを使う

R・**rmarkdown** パッケージ・Pandoc がインストールされているかぎり，R Markdown のフォーマットはあなたの選ぶどんなテキストエディタでも使用できます．しかし，RStudio は R Markdown と深く統合されているので，円滑に R Markdown を使って作業できます．

あらゆる IDE (統合開発環境) と同じく，RStudio にはキーボード・ショートカットがあります．完全な一覧はメニューの `Tools -> Keyboard Shortcuts Help` で見られます．R Markdown に関連する最も便利なショートカットを表 17.1 にまとめました．

加えて，`F7` キーを押してあなたの文書のスペルチェックがができます．`Ctrl + Alt + F10` (macOS では `Command + Option + F10`) で R セッションを再起動することもできます．新しい R セッションから演算するほうが結果はより再現性が高いため，定期的に再起動することは再現性の確保に役立ちます．これはドロップダウンメニューの Run ボタンの後ろに隠れている，“Restart R and Run All Chunks” を使ってもできます．

[*1] 邦題『R で学ぶデータサイエンス』

[*2] https://r4ds.had.co.nz/r-markdown-workflow.html

[*3] https://rmd4sci.njtierney.com/workflow.html

表 17.1: R Markdown に関連する RStudio のキーボード・ショートカット

Task	Windows & Linux	macOS
R チャンクを挿入	Ctrl+Alt+I	Command+Option+I
HTML をプレビュー	Ctrl+Shift+K	Command+Shift+K
文書を knit する (knitr)	Ctrl+Shift+K	Command+Shift+K
Notebook をコンパイル	Ctrl+Shift+K	Command+Shift+K
PDF をコンパイル	Ctrl+Shift+K	Command+Shift+K
ここから上のチャンクをすべて実行	Ctrl+Alt+P	Command+Option+P
このチャンクを実行	Ctrl+Alt+C	Command+Option+C
このチャンクを実行	Ctrl+Shift+Enter	Command+Shift+Enter
次のチャンクを実行	Ctrl+Alt+N	Command+Option+N
全てのチャンクを実行	Ctrl+Alt+R	Command+Option+R
次のチャンクかタイトルへ移動	Ctrl+PgDown	Command+PgDown
前のチャンクかタイトルへ移動	Ctrl+PgUp	Command+PgUp
文書のアウトラインを表示 / 隠す	Ctrl+Shift+O	Command+Shift+O
本, ウェブサイトその他のビルド	Ctrl+Shift+B	Command+Shift+B

17.2 R Markdown のスペルチェック

RStudio IDE を使っているなら, `F7` キーを押すかメニューの `Edit -> Check Spelling` をクリックして Rmd 文書のスペルチェックができます. リアルタイムなスペルチェックは RStudio v1.3 で使えるようになったので, これ以降のバージョンならば手動でスペルチェックを動作させる必要はなくなりました.

RStudio を使っていないなら, **spelling** パッケージ (Ooms and Hester, 2020) に `spell_check_files()` 関数があります. これは R Markdown を含む一般的な文書フォーマットのスペルチェックができます. Rmd 文書のスペルチェック時は, コードチャンクはスキップされテキストのみチェックされます.

17.3 `rmarkdown::render()` で R Markdown をレンダリングする

RStudio あるいは他の IDE を使用していないなら, 次の事実を知っておくべきでしょう. R Markdown 文書は `rmarkdown::render()` 関数によってレンダリングされているのです. つまり, あ

らゆる R スクリプト内でプログラミングによって R Markdown 文書をレンダリングできることを意味します. 例えば, for ループで連続した調査レポートを州ごとにレンダリングできます.

```
for (state in state.name) {
  rmarkdown::render(
    'input.Rmd', output_file = paste0(state, '.html')
  )
}
```

出力ファイル名は州ごとに異なります. 州を state 変数にして input.Rmd 文書の中で使うこともできます. これが例です.

```
---
title: "`r state` に関するレポート"
output: html_document
---

`r state` の面積は `r state.area[state.name == state]` 平方マイルである.
```

?rmarkdown::render のヘルプを読むと他にも使える引数を知ることができます. ここではそれらのうち clean と envir 引数の 2 つだけを紹介しようと思います.

前者の clean は Pandoc の変換がうまくいかない時のデバッグに特に役立ちます. rmarkdown::render(..., clean = FALSE) を呼び出すと, .md ファイルを含め, .Rmd ファイルから knit された全ての中間ファイルが維持されます. Pandoc がエラーを発していたらこの .md ファイルからデバッグを始めることもできます..

後者の envir は rmarkdown::render(..., envir = new.env()) を呼び出した時に, 確実に空の新しい環境で文書をレンダリングする方法を提供してくれます. つまりコードチャンク内で作成されたオブジェクトはこの環境内にとどまり, あなたの現在のグローバル環境を汚すことがありません. 一方で, Rmd 文書を新しい R セッションでレンダリングして, いま開いている R セッションのオブジェクトがあなたの Rmd 文書を汚さないようにしたいのであれば, この例のように rmarkdown::render in xfun::Rscript_call() を呼び出せばよいでしょう.

```
xfun::Rscript_call(
  rmarkdown::render,
  list(input = 'my-file.Rmd', output_format = 'pdf_document')
)
```

この方法は RStudio で Knit ボタンをクリックする方法と似ています．これも同様に新しい R セッションで Rmd 文書をレンダリングします．Rmd 文書を他の Rmd 文書内でレンダリングする必要がある場合は，コードチャンクで直接 rmarkdown::render() を呼び出すのではなく，代わりにこちらの方法を使うことを強く勧めます．なぜなら rmarkdown::render() は内部で多くの副産物をもたらし，さらにそれらに依存関係があることから，同じ R セッションで他の Rmd 文書をレンダリングするのに影響を及ぼすことがあるからです．

xfun::Rscript_call() の第 2 引数は rmarkdown::render() に渡す引数のリストを取ります．実は xfun::Rscript_call は汎用的な関数で，新しい R セッションで任意の R 関数（引数はオプション）を呼び出します．関心があるならヘルプページをご覧になってください．

17.4 パラメータ化されたレポート

17.3 節では for ループ内で一連のレポートをレンダリングする方法を 1 つ紹介しました．実際には rmarkdown::render() はこのタスクのために設計された params という名前の引数を持っています．この引数を通じてレポートをパラメータ化することができます．レポート用のパラメータを指定した時は，レポート内で params 変数が使えます．例えば，以下を呼び出したとします．

```
for (state in state.name) {
  rmarkdown::render('input.Rmd', params = list(state = state))
}
```

それから input.Rmd 内部では，オブジェクト params が state 変数を持つリストになります．

```
---
title: "`r params$state` に関するレポート"
output: html_document
---
```

```
`r params$state` の面積は
`r state.area[state.name == params$state]`
平方マイルである.
```

レポートに対してパラメータを指定する別の方法として, YAML フィールドで params を使うという手もあります. 例えばこのように.

```
---
title: パラメータ化されたレポート
output: html_document
params:
  state: ネブラスカ州
  year: 2019
  midwest: true
---
```

YAML の params フィールドまたは rmarkdown::render() の params 引数には, いくつでもパラメータを含められることに注目してください. YAML のフィールドと rmarkdown::render() の引数とが両方あるときには, render() の引数の値が対応する YAML フィールドの値を上書きしてしまいます. 例えば先ほどの params フィールドを使った例で rmarkdown::render(..., params = list(state = 'アイオワ州', year = 2018) を呼び出した場合は, R Markdown 文書上の params$state は **ネブラスカ州** の代わりに **アイオワ州** に, params$year は 2019 の代わりに 2018 になります.

同じ R Markdown 文書を一連のレポート群へとレンダリングする時は, 各レポートのファイル名が一意になるように rmarkdown::render() の output_file 引数を調整する必要があります. そうでないと, うっかりレポートファイルを上書きしてしまいます. 例えば, 各州の各年ごとにレポートを生成できる関数を書きます.

```
render_one <- function(state, year) {
  # input.Rmd の出力フォーマットが PDF と仮定
  rmarkdown::render(
    'input.Rmd',
```

```
    output_file = paste0(state, '-', year, '.pdf'),
    params = list(state = state, year = year),
    envir = parent.frame()
  )
}
```

そして for ループをネストして全てのレポートを生成します.

```
for (state in state.name) {
  for (year in 2000:2020) {
    render_one(state, year)
  }
}
```

最終的に, **アラバマ州**-2000.pdf, **アラバマ州**-2001.pdf, ..., **ワイオミング州**-2019.pdf, and **ワイオミング州**-2020.pdf のように一連のレポートを得られます.

パラメータ化されたレポートであれば, Shiny で作成されたグラフィカルユーザーインターフェイス (GUI) を通して対話的にパラメータを入力することも可能です. これは YAML に params フィールドを与えることが必要ですが, 各パラメータに対応する適切な入力ウィジェットを用いた GUI を **rmarkdown** が自動的に作成してくれます. 例えばチェックボックスはブーリアン型のパラメータに対して用意されます.

RStudio を使用していないなら, rmarkdown::render() 呼び出して params = 'ask' を渡せば GUI を開始できます.

```
rmarkdown::render("input.Rmd", params = "ask")
```

RStudio を使用しているなら, メニューの Knit ボタンの中にある Knit with Parameters をクリックすることが可能です. 図 17.1 はパラメータに対する GUI の例を示しています.

パラメータ化されたレポートの詳細については, *R Markdown Definitive Guide* (Xie J.J. Allaire, and Grolemund, 2018) の Chapter 15[*4] を読むとよいでしょう.

[*4] https://bookdown.org/yihui/rmarkdown/parameterized-reports.html

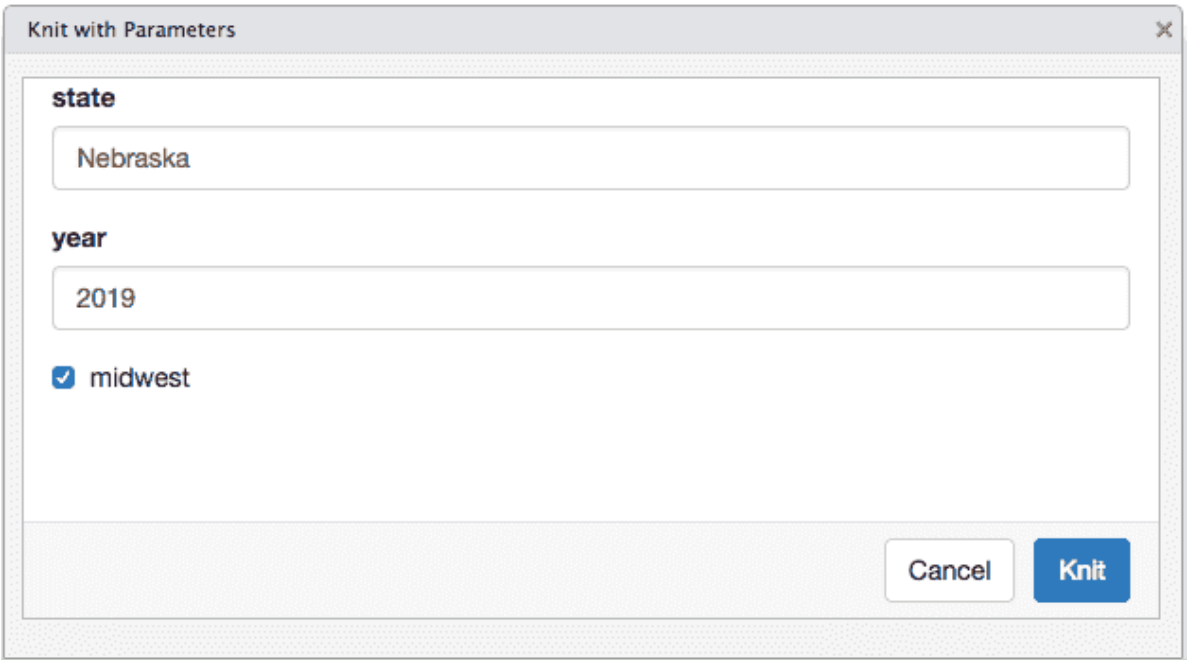


図 17.1: GUI から入力できるパラメータで R Markdown を knit する

17.5 `Knit` ボタンをカスタマイズする (*)

RStudio の `Knit` ボタンをクリックすると, 新規の R セッション内で `rmarkdown::render()` が呼び出され, 同じディレクトリに入力ファイルと同じ基底名の出力ファイルが出力されます. 例えば出力フォーマット `html_document` で `example.Rmd` を knit すると, `example.html` というファイルが作られます.

文書がどうレンダリングされるかをカスタマイズしたいという状況もあるでしょう. 例えば今日の日付を文書に含めたり, コンパイルした文書を別のディレクトリに出力したいというときです. このようなことは適切な `output_file` 引数を付けて `rmarkdown::render()` を呼び出すことで実現できるのですが (17.3 節参照) , レポートをコンパイルするのに `rmarkdown::render()` をいちいち呼び出すことに頼るのは不便かもしれません.

文書の YAML フロントマターで `knit` フィールドを与えれば `Knit` ボタンの挙動を制御できます. このフィールドは, 主要な引数 `input`（ 入力 Rmd 文書のパス）を伴って関数を取ってくれますが, 現時点では他の引数は無視されます. 関数のソースコードを直接 `knit` コードに書くことも, R パッケージなどどこか別の場所に関数を置いてそれを呼び出すことも可能です. カスタム `knit` 関数が日常的に必要ならば, 毎度のように R Markdown 文書に関数のソースコードを繰り返し書くのではなく, パッケージに関数を置くことをお薦めします.

YAML に直接ソースコードを置くなら, 関数全体をパーレン () で囲まなければなりません. ソースコードが複数行になるなら, 最初の行以外の全ての行にスペース 2 つ分のインデントをしなければなりません. 例えば出力ファイル名にレンダリングした日付を含めたい場合, 次のような YAML コードが使用可能です.

```
---
knit: (function(input, ...) {
    rmarkdown::render(
      input,
      output_file = paste0(
        xfun::sans_ext(input), '-', Sys.Date(), '.html'
      ),
      envir = globalenv()
    )
  })
---
```

例えば 2019/07/29 に `example.Rmd` を knit したなら, 出力ファイル名は `example-2019-07-29.html` となります.

上記のアプローチは単純で直截的ですが, 関数が R Markdown 文書で使われるのが 1 度限りでないと, YAML に直接関数を埋め込むのは管理が難しくなります. そこで例えばパッケージ内に `knit_with_date()` という関数を作成するとよいでしょう.

```
#' RStudio 用のカスタム knit 関数
#'
#' @export
knit_with_date <- function(input, ...) {
  rmarkdown::render(
    input,
    output_file = paste0(
        xfun::sans_ext(input), '-', Sys.Date(), '.',
        xfun::file_ext(input)
    ),
    envir = globalenv()
```

```
  )
}
```

上記のコードを **myPackage** という名前のパッケージに追加すれば, 次のような YAML 設定を使いカスタム `knit` 関数を参照することが可能になります.

```
---
knit: myPackage::knit_with_date
---
```

`?rmarkdown::render` のヘルプページを見て, `Knit` ボタンの背後にある `knit` 関数のカスタマイズについて, さらなるアイディアを見つけるのもよいでしょう.

17.6 trackdown で Google ドライブの Rmd 文書を共同編集する

trackdown パッケージ (Kothe Zandonella Callegher, et al., 2021) は R Markdown (または Sweave) 文書の共同執筆・編集に対するシンプルなソリューションを提案してくれます. **trackdown** は **googledrive** パッケージ (D'Agostino McGowan and Bryan, 2021) を基にして, ローカルの `.Rmd` (または `.Rnw`) ファイルをプレーンテキスト形式として Google ドライブにアップロードします. Markdown (あるいは LaTeX) の構文 の可読性のよさ[*5]と Google ドキュメントの提案する広く普及しているオンラインのインターフェースという利点を活かすことで, 共同編集者たちは容易に執筆編集作業に貢献することができます. 全ての著者の貢献を統合したのち, 最終的な文書がローカルにダウンロードされレンダリングされます.

trackdown は CRAN から, あるいは開発版を GitHub からインストールしてもいいでしょう. (https://github.com/claudiozandonella/trackdown):

```
# install from CRAN
install.packages("trackdown")

# install the development version
```

[*5] **訳注**: LaTeX の可読性?? バイナリファイルと比較して, ということでしょうか?

```
05 remotes::install_github("claudiozandonella/trackdown", build_vignettes = TRUE)
```

17.6.1 trackdown の作業工程

.Rmd (あるいは .Rnw) の共同での読み書き作業では, 異なるコンピュータのコードやナラティブを持つテキストを利用することが重要となります:

- **コード** - 共同作業でのコード執筆は伝統的な **Git** を下地にしたオンラインリポジトリ (GitHub や GitLab) を使用する工程に沿って, 最も効率的になされます.
- **ナラティブ** - ナラティブとしてのテキスト共同執筆は, 同じ文書を複数のユーザで同時編集できる, よく知られシンプルなオンラインインターフェースである **Google ドキュメント** の使用によって最も効率的になされます.

したがって, この作業工程のイメージは単純です. .Rmd (または .Rnw) 文書をナラティブの共同編集のため Google ドキュメントとして Google ドライブにアップロードし, Git のバージョン管理と共同作業機能の力を利用しつつコードが機能し続けるようローカルに文書をダウンロードします. この Google ドライブへのアップロード / からのダウンロードの繰り返し作業は納得のいく成果物ができるまで続けられます (図 17.2 参照). この作業工程はこのように要約できます.

> **Git** による **コード** の共同執筆と **Google ドキュメント** による **ナラティブ**の共同執筆

作業工程の詳細な例は https://ekothe.github.io/trackdown/articles/trackdown-workflow.html を見てください.

関数と特別な機能

trackdown は作業工程の管理のために様々な関数を提供します.

- `upload_file()` は Google ドライブへ最初にファイルをアップロードする際に使います.
- `update_file()` は Google ドライブにアップロード済みのファイルをローカルファイルで更新します.
- `download_file()` はローカルファイルを, ダウンロードした Google ドライブのファイルの編集済みバージョンで更新します.
- `render_file()` は Google ドライブのファイルをダウンロードしローカルでレンダリングします.

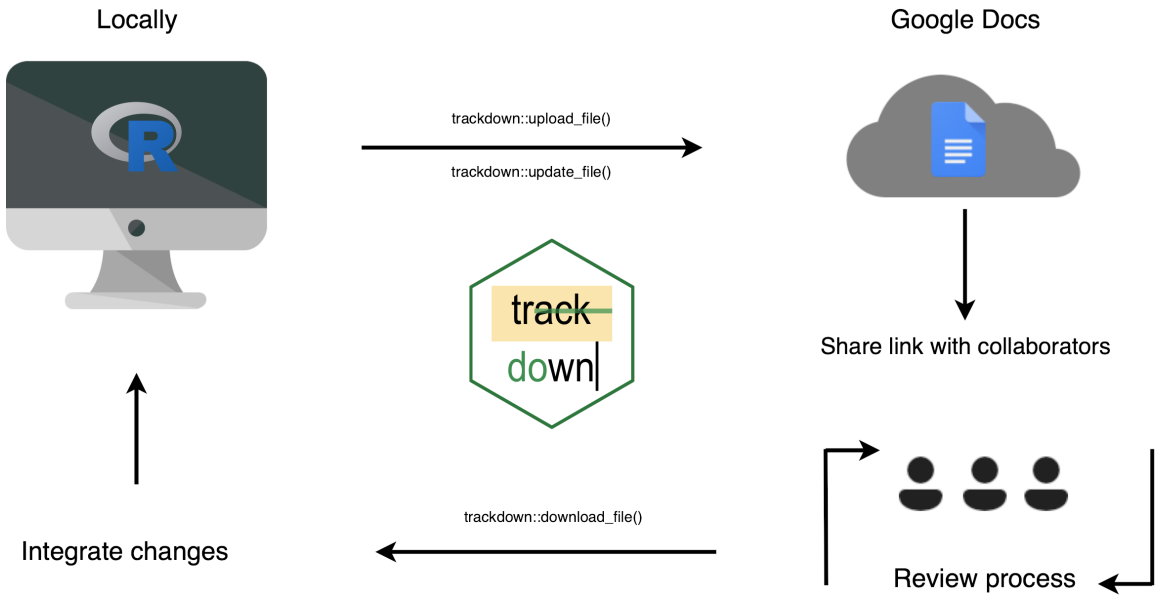


図 17.2: trackdown の作業工程, コードの共同執筆は Git を使いローカルで行い, ナラティブの共同執筆は Google ドキュメントを使いオンラインで行う

さらにそれ以外にも **trackdown** は Google ドキュメントの文書の共同執筆・編集を円滑にする機能を提案します.

- **コードを隠す:** 文書のヘッダにあるコード (YAML ヘッダや LaTeX プリアンブル) やコードチャンクは Google ドライブへのアップロード時に除去され, ダウンロード時まで自動保存されます. これによって共同編集者がコードをうっかり変更してファイルを破損してしまうのを防ぎ, ナラティブに意識を集中させることができます.
- **出力のアップロード:** 実際の出力文書, つまりレンダリングされたファイルは Goodle ドライブに `.Rmd` (または `.Rnw`) ファイルとまとめてアップロードされます. これは共同編集者が図や表を含む全体のレイアウトを見通すのに役立ち, コメントの追加や変更の提案・議論を可能にします.
- **Google ドライブの共有ドライブを使用する:** 文書を個人用の Google ドライブにも, 共同作業促進のために共有ドライブにもアップロードできます.

Google ドキュメントの利点

Google ドキュメントはユーザに親しみやすく直感的な, 無償の web ベースのインターフェースを提案しており, 複数ユーザが同時に同じ文書を編集することが可能です. Google ドキュメントでは以下が可能です.

- 変更箇所の追跡 (提案の採用・却下履歴を含む)
- 変更の提案と議論についてのコメントの追加
- スペル・文法ミスのチェック (Grammarly のようなサードパーティのサービスとの統合も可能)

さらに Google ドキュメントは誰でも文書の共同編集に貢献することが可能です. プログラミング経験も要求せず, ユーザは単にナラティブの編集にだけ集中することができます.

全ての共同編集者が Google アカウントを持つ必要がないことも知っておいてください (ただし Google ドキュメントの全ての機能を活用するために持っておくことをお勧めします). **trackdown** の作業工程を管理する人物だけが, ファイル Google ドライブにアップロードするためにアカウントを持つ必要があります. 共有リンクを使用して他の共同編集者を文書の編集に招待することができます.

17.7 workflowr で R Markdown プロジェクトを研究用サイトでまとめる

workflowr パッケージ (J. Blischak Carbonetto, and M. Stephens, 2020; J. D. Blischak Carbonetto, and M. Stephens, 2019) は (データ分析の) プロジェクトをテンプレートとバージョン管理ツールである GIT を使って体系的に編成することが可能です. プロジェクトに変更を加えるたびに, 変更の記録を残すことができるので, **workflowr** はプロジェクトの特定のバージョンと対応するウェブサイトを構築できます. これはあなたの分析結果の履歴をすべて閲覧できることを意味します. このパッケージはバージョン管理のためバックエンドで GIT を使用していますが, 特に GIT に詳しくなる必要はありません. このパッケージは, 内部で GIT の操作を行う R の関数を提供し, あなたはこれらの関数を呼び出せばいいだけです. そのうえ, **workflowr** は自動的に再現可能なコードへのベストプラクティスを自動化します. R Markdown 文書がレンダリングされるたびに, **workflowr** は `set.seed()` でシード値を設定, `sessionInfo()` でセッション情報を記録, そして絶対ファイルパスをスキャンする, などなど, といったことを自動的に行います. このパッケージの導入方法と詳細はパッケージのドキュメント[*6]をご覧ください.

workflowr の主著者である John Blischak は, R プロジェクトのワークフローと関連のあるパッケージとガイドを網羅的ではないですがリストにまとめています. これは GitHub レポジトリ https://github.com/jdblischak/r-project-workflows で見ることができます.

[*6] https://jdblischak.github.io/workflowr/

17.8 R Markdown から E メールを送信する Send emails based on R Markdown

blastula パッケージ (Richard Iannone and Cheng, 2020) があれば Rmd 文書を E メールの本文にして送信できます. Rmd 文書を E メールへレンダリングするには, 文書に出力フォーマット `blastula::blastula_email` を使用する必要があります.

```
---
title: 週次レポート
output: blastula::blastula_email
---

ボスへ

お疲れ様です.

以下が `iris` データの分析になります.

```{r}
summary(iris)
plot(iris[, -5])
```

もううんざりだというのなら知らせていただきたく.

よろしくお願いします
ジョン
```

この Rmd 文書は `blastula::render_email()` 関数でレンダリングされ, 出力は `blastula::smtp_send()` に渡されます. これは E メールを送信する関数です. `smtp_send()` には E メールサーバとあなたの認証が必要であることに注意してください.

RStudio Connect を使用しているなら, https://solutions.rstudio.com/r/blastula/ で, 自動化したもの, 条件付けたもの, パラメータ化した E メールを含め, さらなる例が見つかります.

付録 A

knitr のチャンク及びパッケージオプション

この付録は https://gedevan-aleksizde.github.io/knitr-doc-ja/options.html で公開されているものと同一です.

knitr パッケージはソースコード, テキスト, グラフ, チャンクで使用するプログラミング言語といった, コードチャンクのコンポネントのほとんど全部をカスタマイズするための多くのオプションを提供します. `knit` 処理のカスタマイズをパッケージレベルでカスタマイズするオプションもあります. この章では **knitr** で使用できる全てのチャンクオプションとパッケージオプションを解説します. 以下のリスト中で, オプションのデフォルト値になっているものはカッコ内に表記しています.

A.1 チャンクオプション一覧

チャンクオプションはチャンクのヘッダに書きます. チャンクヘッダの構文は文書フォーマットがなんであるかに依存します. 例えば `.Rnw` ファイル (R + LaTeX) であれば, `<< >>=` という記号の中に書きます. `.Rmd` ならば, チャンクヘッダは ```` ```{r} ```` 内に書きます. 以下の例は主に `.Rmd` (R Markdown) の場合ですが, ほとんどのチャンクオプションはどのフォーマットでも使用可能です.

チャンクオプションは以下のように **タグ名=値** という形式で書きます.

````
```{r, my-chunk, echo=FALSE, fig.height=4, dev='jpeg'}
```
````

チャンクラベルは特殊なチャンクオプションです (例: 先ほどの例の `my-chunk` がそれにあたりま

す). これは唯一のタグが不要なチャンクオプションです (つまり, 値のみ書くことになります). もし **タグ名=値** の形式で書きたいのならば, チャンクオプション名の `label` を明示的に使うこともできます.

```
```{r label="my-chunk"}
```
```

各チャンクのラベルは文書内において一意であることが前提です. 特にキャッシュとグラフのファイル名はチャンクラベルで紐付けているため重要です. ラベルのないチャンクは `unnamed-chunk-i` という形式でラベル名が割り当てられます. `i` は順に整数が割り当てられます.

文書全体のチャンクオプションのデフォルト値を変更するために `knitr::opts_chunk$set()` を使うことができます. 例えば以下のようなチャンクを文書の冒頭に書きます.

```
```{r, setup, include=FALSE}
knitr::opts_chunk$set(
  comment = '', fig.width = 6, fig.height = 6
)
```
```

チャンクオプションの豆知識をいくつか掲載します.

1. チャンクヘッダは 1 行で書かねばなりません. 改行してはいけません.
2. チャンクラベルとファイルパスにスペース, ピリオド `.`, アンダースコア `_` を使用するのは避けましょう. セパレータが必要ならば, ハイフン `-` の使用を推奨します. 例えば `setup-options` はラベル名として望ましいですが `setup.options` や `chunk 1` は良くありません. `fig.path = 'figures/mcmc-'` はパス名として良いですが, `fig.path = 'markov chain/monte carlo'` は良くありません.
3. 全てのオプションの値は **R の構文として適切でなければなりません**. チャンクオプションを関数の引数のように考えると良いでしょう.

- 例えば **character** 型をとるオプションは引用符で囲まなければなりません. 例: `results = 'asis'` や `out.width = '\\textwidth'`. ただしリテラルのバックスラッシュは二重のバックスラッシュが必要なことを忘れないでください.
- 理論上はチャンクラベルもまた引用符で囲む必要がありますが, 利便性のため書かなくとも自動で引用符が追加されます (例: ```` ```{r, 2a}``` ```` は ```` ```{r, label='2a'}``` ```` として扱われます).
- R のコードとして有効なものである限り, いくらでも複雑な構文を書くことができます.

以下では **オプション**: (**デフォルト値**; 値の型) という形式で, **knitr** で使えるチャンクオプションのリストを掲載します.

A.1.1 コード評価関連

- **eval**: (TRUE; logical または numeric).: コードチャンクを評価するかどうか. どの R の評価式を評価するかを選ぶために numeric のベクトルを使用することもできます. 例: eval=c(1, 3, 4) ならば 1 つ目, 3 つ目, そして 4 つ目の評価式を評価し, eval = -(4:5) は 4, 5 つ目の式以外の全てを評価します.

A.1.2 テキストの出力関連

- **echo**: (TRUE; logical または numeric).: 出力される文書にソースコードを表示するかどうか. 「表示」「隠す」に対応する TRUE/FALSE に加え, どの R の評価式を表示するかを選ぶために numeric のベクトルを使用することもできます. 例: echo=2:3 は 2, 3 番めの評価式を表示し, echo = -4 は 4 番目だけを非表示にします.
- results: ('markup'; character) 実行結果のテキストの部分をどう表示するかを制御します. このオプションは通常のテキスト出力にのみ影響することに注意してください (警告・メッセージ・エラーは適用範囲外です). 取りうる値は次のとおりです.
  - markup: 出力の文書フォーマットに応じて適切な環境でテキスト出力をマークアップします. 例えば R Markdown ならば "[1] 1 2 3" が **knitr** によって以下のように加工されます. この場合, results='markup' は囲み (```) 付きのコードブロックとして出力されることを意味します.

```
[1] 1 2 3
```

  - asis: テキスト出力を「そのまま」書き出します. つまり, 生の結果テキストをマークアップ一切なしでそのまま文書に書き出します.

```{r, results='asis'}
cat("I'm raw **Markdown** content.\n")
```

  - **hold**: チャンクと flush の全てのテキスト出力をチャンクの末尾に固定します.
  - **hide** (または FALSE): テキスト出力を表示しません.
- collapse: (FALSE; logical) 可能であれば, ソースと出力をつなげて 1 つのブロックにするか

どうかです (デフォルトではソースと出力はそれぞれ独立したブロックです). このオプションは Markdown 文書でのみ有効です.

- **warning**: (TRUE; logical).: 警告文 (warning() で出力されるテキスト) を保存するかどうかです. FALSE の場合, 全ての警告文は文書に出力されず, 代わりにコンソールに書き出されます. 警告文の一部を選ぶインデックスとして, numeric 型のベクトルを指定することもできます. この場合のインデックスの数値は「何番目の警告文を表示するか」を参照する (例: 3 はこのチャンクから投げられた 3 番目の警告文を意味します) ものであって,「何番目の R コードの警告文の出力を許可するか」ではないことに注意してください.
- **error**: (TRUE; logical).: エラー文 (stop() で出力される文です) を保持するかどうかです. デフォルトの TRUE では, **コード評価はエラーが出ても停止しません!** エラー時点で停止させたい場合はこのオプションを FALSE に指定してください. **R Markdown ではこのデフォルト値は FALSE に変更されていることに注意**してください. チャンクオプションに include=FALSE がある場合, 起こりうるエラーを見落とさないように, **knitr** はエラー時点で停止するようになります.
- **message**: (TRUE; logical).: message() が出力するメッセージ文を (warning オプションと同様に) 表示するかどうかです.
- **include**: (TRUE; logical).: 出力する文書にチャンクの出力を含めるかどうかです. FALSE ならばなにも書き出されませんが, コードの評価はなされ, チャンク内にプロット命令があるのならグラフのファイルも生成されます. よってこの図をそれ以降で任意に挿入することもできます.
- **strip.white**: (TRUE; logical).: 出力時にソースコードの冒頭と末尾から空白行を除去するかどうかです.
- **class.output**: (NULL; character).: テキストの出力ブロックに追加するクラス名のベクトル. このオプションは R Markdown で HTML を出力する際にのみ機能します. 例えば class.output = c('foo', 'bar') はテキスト出力が <pre class="foo bar"></pre> で囲まれたブロックとして書き出されます.
- **class.message**/**class.warning**/**class.error**: (NULL; character)・: class.output と同様に, R Markdown においてそれぞれ メッセージ文, 警告文, エラー文のブロックに対してクラス名を与えます. class.source もまた同様にソースコードのブロックに対して適用されます. **ref**(#code-decoration) 節を参照してください.
- **attr.output**/**attr.message**/**attr.warning**/**attr.error**: (NULL; character).: 上記の class.* オプション群と同様に, Pandoc に対してコードブロックの属性を指定します. つまり class.* は attr.* の特殊ケースです. 例: class.source = 'numberLines' は attr.source = '.numberLines' と等価ですが, attr.source は任意の値を取ることができます. 例えば, attr.source = c('.numberLines', 'startFrom="11"').
- **render**: (knitr::knit_print; function(x, options, ...)).: チャンクで表示される値に対して適用する関数です. 関数の第 1 引数には (x) はチャンクの各評価式が評価された結果が

与えられます．このチャンクのチャンクオプションがリストとして第二引数 opstions に与えられます．この関数は文字列を返すことを想定しています．詳細は package vignette (vignette('knit_print', package = 'knitr')) を参照してください．

- **split**: (FALSE; logical).: 出力ブロックを分割したファイルに書き出すかどうか．LaTeX ならば \input{} で読み込み，HTML ならば <iframe></iframe> タグで読み込まれます．このオプションは .Rnw, .Rtex そして .Rhtml でのみ機能します．

A.1.3 コードの装飾関連

- tidy: (FALSE) R コードを整形するかどうかです．他の有効な値は次のとおりです．
  - TRUE (tidy = 'formatR' と等価です): 整形のために formatR::tidy_source() を呼び出します．
  - 'styler': コード整形のために styler::style_text() を呼び出します．
  - 整形されたコードを返す，function(code, ...) {} という形式の任意の関数．
  - 整形が失敗した場合，元の R コードは変更されません (警告は表示されます)．
- tidy.opts: (NULL; list) tidy オプションで指定した関数へのオプション引数のリストです．例えば tidy.opts = list(blank = FALSE, width.cutoff = 60) は tidy = 'formatR' に対して空白行を削除し各行が 60 文字におさまるように改行しようとします．
- prompt: (FALSE; logical) R コードにプロンプト記号 (> など) を付けるかどうかです．?base::options ヘルプページの prompt と continue を参照してください．プロンプト記号の追加は，読者がコードをコピーするのを難しくさせるため，prompt=FALSE のほうが良い選択であることに留意してください．エンジンが R 以外の場合，このオプションはうまく機能しないことがあります (issue #1274[*1])．
- **comment**: ('##'; character).: テキスト出力の各行の先頭文字です．デフォルトでは，コメントアウトできるよう ## となっているので，読者は文書から任意の範囲をそのままコピーしても出力部分は無視されるのでそのまま実行することができます．comment = '' を指定することで，デフォルトの ## は除去されます．
- **highlight**: (TRUE; logical).: ソースコードをシンタックスハイライトするかどうかです[*2]．
- **class.source**: (NULL; character).: 出力された文書のソースコードブロックのクラス名です．出力ブロックに対して機能する class.output をはじめとする class.* シリーズと同様です．
- **attr.source**: (NULL; character).: ソースコードブロックの属性です．attr.output などの attr.* シリーズと同様です．
- size: ('normalsize'; character) .Rnw 使用時のチャンクサイズのフォントサイズです．指定

[*1] https://github.com/yihui/knitr/issues/1274
[*2] 訳注: R Markdown ではさらに，YAML フロントマターで適用するハイライトのテーマ名を指定できます

可能なサイズは overleaf のヘルプページ (英語)[*3] を参照してください[*4].

- **background**: ('#F7F7F7'; character).: .Rnw 使用時のチャンクブロックの背景色です[*5].
- **indent**: (character).: チャンクの出力で各行に追加する文字です. 典型的には空白と同義です. このオプションは読み込み専用を想定しており, 値は **knitr** が文書を読み込む際に設定されます. 例えば以下のチャンクでは, indent は空白文字 2 個です[*6].

```
```{r indent-example, echo=2}
set.seed(42)
rnorm(10)
```
```

A.1.4 キャッシュ関連

- **cache**: (FALSE; logical).: コードチャンクのキャッシュを取るかどうかです. 初回の実行またはキャッシュが存在しない場合は通常通り実行され, 結果がデータセットが保存され (.rdb, .rdx ファイルなど), それ以降でコードチャンクが評価されることがあれば, 以前保存されたこれらのファイルからこのチャンクの結果を読み出します. ファイル名がチャンクラベルと R コードの MD5 ハッシュ値で一致する必要があることに注意してください. つまりチャンクになんらかの変更がある度に異なる MD5 ハッシュ値が生成されるため, キャッシュはその度に無効になります. 詳細は キャッシュの解説[*7] を参考にしてください.
- **cache.path**: ('cache/'; character).: 生成したキャッシュファイルの保存場所を指定します. R Markdown ではデフォルトでは入力ファイルの名前に基づきます. 例えば INPUT.Rmd の FOO というラベルのチャンクのキャッシュは INPUT_cache/FOO_*.* というファイルパスに保存されます.
- **cache.vars**: (NULL; character).: キャッシュデータベースに保存される変数名のベクトルを指定します. デフォルトではチャンクで作られた全ての変数が識別され保存されますが, 変数名の自動検出はロバストではないかもしれませんし, 保存したい変数を選別したい場合もあるかもしれないので, 保存したい変数を手動選択することもできます.
- **cache.globals**: (NULL; character).: このチャンクで作成されない変数の名前のベクトルを指定します. このオプションは主に autodep = TRUE オプションをより正確に動作させたいときに使います. チャンク B で使われているグローバル変数が チャンク A のローカル変数として使われているときなど. グローバル変数の自動検出に失敗した際に使う場合, こに

[*3] https://www.overleaf.com/learn/latex/Font_sizes,_families,_and_styles
[*4] 訳注: \normalsize, \Large, \LARGE など LaTeX で指定できるフォントサイズを表すマクロのことを指しています
[*5] 訳注: R Markdown では背景色は CSS や class.output などで設定する必要があります. 詳細は R Markdown Cookbook などを参照してください
[*6] 訳注: R Markdown の場合は **knitr** 以外の中間処理があるため, 必ずしもこのルールを守りません
[*7] https://gedevan-aleksizde.github.io/knitr-doc-ja/cache.html#cache

オプションを使って手動でグローバル変数の名前を指定してください (具体例として issue #1403[*8] を参照してください).

- **cache.lazy**: (TRUE; logical).: 遅延読み込み lazyLoad() を使うか, 直接 load() でオブジェクトを読み込むかを指定します. 非常に大きなオブジェクトに対しては, 遅延読み込みは機能しないかもしれません. よってこの場合は cache.lazy = FALSE が望ましいかもしれません (issue #572[*9] を参照してください).
- **cache.comments**: (NULL; logical).: FALSE の場合, R コードチャンク内のコメントを書き換えてもキャッシュが無効になりません.
- **cache.rebuild**: (FALSE; logical).: TRUE の場合, キャッシュが有効であってもチャンクのコードの再評価を行います. このオプションはキャッシュの無効化の条件を指定したいときに有用です. 例えば cache.rebuild = !file.exists("some-file") とすれば some-file が存在しないときにチャンクが評価されキャッシュが再構成されます (issue #238[*10] を参照).
- **dependson**: (NULL; character または numeric).: このチャンクが依存しているチャンクのラベル名を文字ベクトルで指定します. このオプションはキャッシュされたチャンクでのみ適用されます. キャッシュされたチャンク内のオブジェクトは, 他のキャッシュされたチャンクに依存しているかもしれず, 他のチャンクの変更に合わせてこのチャンクも更新する必要があるかもしれません.
  - dependson に numeric ベクトルを与えた場合, それはチャンクラベルのインデックスを意味します. 例えば dependson = 1 ならばこの文書の 1 番目のチャンクに依存することを意味し, dependson = c(-1, -2) は直前の 2 つのチャンクを意味します (負のインデックスは現在のチャンクからの相対的な位置を表します).
  - opts_chunk$set() によってグローバルにチャンクオプションを設定した場合, このオプションは機能しません. ローカルなチャンクオプションとして設定しなければなりません.
- **autodep**: (FALSE; logical).: グローバル変数を検出することでチャンク間の依存関係を分析するかどうかを指定します (あまり信頼できません). よって, dependson を明示的に指定する必要はありません.

A.1.5 グラフ関連

- **fig.path**: ('figure/'; character).: 図の保存ファイルパスを生成する際の接尾語. fig.path とチャンクラベルを連結したものがフルパスになります. figure/prefix- のようにディレクトリ名が含まれて, それが存在しない場合はディレクトリが作成されます.

[*8] https://github.com/yihui/knitr/issues/1403
[*9] https://github.com/yihui/knitr/issues/572
[*10] https://github.com/yihui/knitr/issues/238

- **fig.keep**: ('high'; character).: グラフをどのように保存するかです. 可能な値は次のとおりです.
  - high: 高水準プロットのみ保存 (低水準の変更は全て高水準プロットに統合されます).
  - none: 全て破棄します.
  - all: 全てのプロットを保存します (低水準プロットでの変更は新しいグラフとして保存されます).
  - first: 最初のプロットのみ保存します.
  - last: 最後のプロットのみ保存します.
  - 数値ベクトルを指定した場合, その値は保存する低水準プロットのインデックスとなります. 低水準プロットとは lines() や points() などの関数によるグラフ描画のことです. fig.keep についてより理解するには次のようなチャンクを考えてください. 通常はこれで 2 つのグラフを出力します (fig.keep = 'high' を指定したので). fig.keep = 'none' としたなら, いかなるグラフも保存されません. fig.keep = 'all' ならば, 4 つのグラフとして保存されます. fig.keep = 'first' ならば plot(1) によって作成されたグラフが保存されます. fig.keep = 'last', なら, 最後の 10 本の垂線を描画したグラフが保存されます.

```
plot(1)  # 高水準プロット
abline(0, 1)  # 低水準の作図
plot(rnorm(10))  # 高水準プロット
# ループ内での複数の低水準作図 (R 評価式としては 1 つ)
for (i in 1:10) {
  abline(v = i, lty = 2)
}
```

- **fig.show**: ('asis'; character).: グラフをどのように表示し, 配置するかです. 可能な値は次のとおりです.
  - asis: グラフが生成された場所にそのまま出力します (R ターミナルで実行した場合とおなじように).
  - hold: 全てのグラフをまとめてチャンクの最後に出力します.
  - animate: チャンクに複数のグラフがある場合, 連結して 1 つのアニメーションにします.
  - hide: グラフをファイルに保存しますが, 出力時は隠します.
- **dev**: (LaTeX の場合は 'pdf'[*11], HTML/Markdown の場合は 'png'; character).: グラフをファイルに保存する際のグラフィックデバイスです. base R および, **Cairo**, **cairoDevice**,

[*11] 訳注: pdf は日本語表示に向いていないため, cairo_pdf などを利用することをおすすめします

svglite, **ragg**, **tikzDevice** パッケージの提供するデバイスに対応しています．デバイスの例: pdf, png, svg, jpeg, tiff, cairo_pdf, CairoJPEG, CairoPNG, Cairo_pdf, Cairo_png, svglite, ragg_png, tikz, など．有効なデバイスの一覧は names(knitr:::auto_exts) を参照してください．また, function(filename, width, height) という引数を定義した関数名を文字列で与えることでも指定できます．画像サイズの単位は **常にインチ**です．ビットマップであってもインチで指定したものがピクセルに変換されます．

チャンクオプション dev, fig.ext, fig.width, fig.height, dpi はベクトルを与えられます (長さが足りない場合は再利用されます)．例えば dev = c('pdf', 'png') は 1 つのグラフに対して 1 つづつ PDF と PNG ファイルを作成します．

- **dev.args**: (NULL; list).: グラフィックデバイスに与える追加の引数です．例えば dev.args = list(bg = 'yellow', pointsize = 10) を dev = 'png' に与えられます．特定のデバイスに依存するオプション (詳細はそれぞれのデバイスのドキュメントを確認してくださささい)．dev に複数のデバイスが指定されている場合は dev.args を引数のリストをさらにリストでくくることになるでしょう．それぞれの引数リストが対応するデバイスに与えられます．例: dev = c('pdf', 'tiff'), dev.args = list(pdf = list(colormodel = 'cmyk', useDingats = TRUE), tiff = list(compression = 'lzw'))．
- **fig.ext**: (NULL; character).: 出力するグラフのファイル拡張子です．NULL ならばグラフィックデバイスに応じて自動決定されます．詳細は knitr:::auto_exts を確認してください．
- **dpi**: (72; numeric)．ビットマップデバイスに対する DPI (インチ毎ドット, dpi * inches = pixels) です．
- **fig.width**, **fig.height**: (いずれも 7; numeric).: グラフの幅と高さです．単位はインチです．グラフィックデバイスに与えられます．
- **fig.asp**: (NULL; numeric).: グラフのアスペクト比, つまり 高さ / 幅 の比です．fig.asp が指定された場合, 高さ (fig.height) は fig.width * fig.asp によって自動設定されます．
- **fig.dim**: (NULL; numeric).: fig.width と fig.height を指定する長さ 2 の数値のベクトルです．例: fig.dim = c(5, 7) は fig.width = 5, fig.height = 7 の省略形です．fig.asp と fig.dim が指定された場合, fig.asp は無視されます (警告文が出力されます)．
- **out.width**, **out.height**: (NULL; character).: 出力時の画像の幅と高さです．実体としての幅と高さである fig.width と fig.height とは異なります．つまりグラフは文書に表示される際にスケールが調整されます．出力フォーマットに応じて, これら 2 つのオプションはそれぞれ特殊な値を取ることができます．例えば LaTeX ならば .8\\linewidth, 3in, 8cm などと指定でき, HTML ならば 300px と指定できます．.Rnw ならば out.width のデフォルト値は \\maxwidth に変更され, その値は framed のページ[*12] で書いたように定義されます．例え

[*12] https://gedevan-aleksizde.github.io/knitr-doc-ja/framed.html#framed

ば '40%' のようにパーセンテージで指定もでき, これは LaTeX では 0.4\linewidth に置き換えられます.

- **out.extra**: (NULL; character).: 図の表示に関するその他のオプションです. LaTeX で出力する場合は \includegraphics[] に挿入される任意の文字に対応し (例: out.extra = 'angle=90' ならば図の 90 度回転), HTML なら <img /> に挿入されます (例: out.extra = 'style="border:5px solid orange;"').
- **fig.retina**: (1; numeric).: このオプションは HTML で出力する際にのみ適用されます. Retina ディスプレイ[*13] に対して画像サイズを調整する比率 (多くの場合は 2 を指定します) です. チャンクオプションの dpi を dpi * fig.retina で, out.width を fig.width * dpi / fig.retina で計算します. 例えば fig.retina = 2 なら, 画像の物理サイズが 2 倍となり, その表示サイズは半分になります.
- **resize.width**, **resize.height**: (NULL; character).: LaTeX で出力する際に \resizebox{}{} コマンドで使われます. これら 2 つのオプションは Tikz グラフィックスをリサイズしたい場合のみ必要になります. それ以外に通常使うことはありません. しかし **tikzDevice** の開発者によれば, 他の箇所のテキストとの一貫性のため, Tikz グラフィックスはリサイズを想定していません. 値の 1 つでも NULL ならば, ! が使用されます (この意味がわからない方は **graphicx** のドキュメントを読んでください).
- **fig.align**: ('default'; character).: 出力時の画像の位置揃え (アラインメント) です. 可能な値は default, left, right, center です. default は位置について特に何も調整しません.
- **fig.link**: (NULL; character) 画像に与えるリンク.
- **fig.env**: ('figure'; character).: 画像に使われる LaTeX 環境. 例えば fig.env = 'marginfigure' ならば \begin{marginfigure} で囲まれます. このオプションの使用は fig.cap が指定されいることが条件です.
- **fig.cap**: (NULL; character).: 図のキャプションです.
- **fig.alt**: (NULL; character) HTML 出力時の図の <img> タグの alt 属性に使う代替テキストです. デフォルトでは, 代替テキストが与えられた場合チャンクオプション fig.cap には代替テキストが使われます.
- **fig.scap**: (NULL; character).: 図の短縮キャプションです. 出力が LaTeX の場合のみ意味をなします. 短縮キャプションは \caption[] コマンドに挿入され, 大抵の場合は PDF 出力時の「図一覧」で表示される見出しとして使われます.
- **fig.lp**: ('fig:'; character).; 図の相互参照に使われるラベル[*14]の接頭語で, \label{} コマンドに挿入されます. 実際のラベルはこの接頭語とチャンクラベルを連結して作られます. 例えば図のラベルが ```{r, foo-plot} will be ならば, デフォルトでは図のラベルは fig:foo-plot になります.

[*13] https://ja.wikipedia.org/wiki/Retina%E3%83%87%E3%82%A3%E3%82%B9%E3%83%97%E3%83%AC%E3%82%A4

[*14] 訳注: チャンクラベルと混同しないでください

- **fig.pos**: (''; character).: LaTeX の \begin{figure}[] に使われる, 画像の位置調整オプション[*15]を指定します.
- **fig.subcap**: (NULL).: subfigures のためのキャプションです. 複数のグラフが 1 つのチャンクにあり, かつ fig.subcap も fig.cap is NULL である場合, \subfloat{} が個別の画像の表示に使われます (この場合はプリアンブルに \usepackage{subfig} と書く必要があります). 具体例は 067-graphics-options.Rnw[*16] を参照してください.
- **fig.ncol**: (NULL; integer). subfigure の数です. 例えばこの issue[*17] を見てください (fig.ncol も fig.sep も LaTeX でのみ機能します).
- **fig.sep**: (NULL; character).: subfigures どうしの間に挿入されるセパレータを指定する文字ベクトルです. fig.ncol が指定された場合, デフォルトでは fig.sep に N 個ごとに \newline が挿入されます (N は列の数です). 例えば fig.ncol = 2 ならばデフォルトは fig.sep = c('', '', '\\newline', '', '', '\\newline', '', ...) となります.
- **fig.process**: (NULL; function).: 画像ファイルに対する後処理の関数です. 関数は画像のファイルパスを引数として, 挿入したい新しい画像のファイルを返すものであるべきです. 関数に options 引数がある場合, この引数にチャンクオプションのリストが与えられます.
- **fig.showtext**: (NULL; logical).: TRUE ならばグラフの描画前に showtext::showtext_begin() が呼ばれます. 詳細は **showtext**[*18] パッケージのドキュメントを参照してください[*19].
- **external**: (TRUE; logical).: tikz グラフィックの処理 (PDF 生成時のコンパイル前の処理) を外部化するかどうかです. **tikzDevice** パッケージの tikz() デバイスを使う場合 (つまり dev='tikz' を指定したとき) のみ使用され, コンパイル時間を短縮することが可能です.
- **sanitize**: (FALSE; character). tikz グラフィックでサニタイズ (ここでは, LaTeX で特殊な意味を持つ文字のエスケープ処理) するかどうかです. 詳細は **tikzDevice** パッケージのドキュメントを参照してください.

さらにこの他に, ユーザーが使用することを想定していない隠しオプションが 2 つあります. fig.cur (複数の図表がある場合の, 現在の図番号 / インデックス) と fig.num (チャンク内の図の合計数) です. これら 2 つのオプションは **knitr** が複数の図そしてアニメーションを処理するためにあります. 場合によっては手動で保存した画像ファイルを使ってアニメーションを書き出す場合などに役に立つかもしれません (使用例として graphics manual[*22] を参照してください).

[*15] 訳注: LaTeX では通常は図の位置は調整されますが, fig.pos='H' ならばその位置で固定されます

[*16] https://github.com/yihui/knitr-examples/blob/master/067-graphics-options.Rnw

[*17] https://github.com/yihui/knitr/issues/1327#issuecomment-346242532

[*18] http://cran.rstudio.com/package=showtext

[*19] 訳注: showtext は手っ取り早く日本語を表示できますが, いくつかの制約があります. 詳細は『おまえはもう R のグラフの日本語表示に悩まない (各 OS 対応)[*20]』『R でのフォントの扱い[*21]』などを見てください.

[*22] https://github.com/yihui/knitr/releases/download/doc/knitr-graphics.pdf

A.1.6 アニメーション関連

- **interval**: (1; numeric).: アニメーションの 1 フレームごとの時間 (単位は秒) です.
- **animation.hook**: (knitr::hook_ffmpeg_html; function または character). HTML 出力時のアニメーション作成用のフック関数を指定します. デフォルトでは FFmpeg を使って WebM 動画ファイルに変換します.
  - 別のフック関数として **gifski**[*23] パッケージの knitr::hook_gifski 関数は GIF アニメーションを作ることができます.
  - このオプションは 'ffmpeg' や 'gifski' といった文字列を指定することもできます. これらは対応するフック関数の省略形です. 例: animation.hook = 'gifski' は animation.hook = knitr::hook_gifski を意味します.
- **aniopts**: ('controls,loop'; character).: アニメーションに対する追加のオプションです. 詳細は LaTeX の **animate** パッケージのドキュメント[*24]を参照してください.
- **ffmpeg.bitrate**: (1M; character).: WebM 動画の質を制御するための FFmpeg の引数 -b:v に対応する値を指定できます.
- **ffmpeg.format**: (webm; character).: FFmpeg の出力する動画フォーマットです. つまり, 動画ファイルの拡張子名です.

A.1.7 コードチャンク関連

- **code**: (NULL; character).: 指定された場合, そのチャンクのコードを上書きします. この機能によって, プログラミング的にコード挿入が可能になります. 例えば code = readLines('test.R') とすれば test.R の内容を現在のチャンクで実行します.
- **ref.label**: (NULL; character).: 現在のチャンクのコードに引き継ぐ, 別のチャンクのラベルの文字列ベクトルを指定します (動作例は チャンク参照[*25] を確認してください).

A.1.8 子文書関連

- **child**: (NULL; character).: 親文書に挿入する子文書のファイルパスを示す文字ベクトルを指定します.

[*23] https://cran.r-project.org/package=gifski
[*24] http://ctan.org/pkg/animate
[*25] https://gedevan-aleksizde.github.io/knitr-doc-ja/reference.html#reference

A.1.9 言語エンジン関連

- **engine**: ('R'; character).: コードチャンクの言語名です. 指定可能な名前は names(knitr::knit_engines$get()) で確認できます. 例: python, sql, julia, bash, c, など. knitr::knit_engines で他の言語を使うためのセットアップが可能です.
- **engine.path**: (NULL; character).: 実行可能なエンジンのパスを指定します. あなたのお使いのシステムの別の実行ファイルを使用するためのオプションです. 例えば python はデフォルトでは /usr/bin/python を参照しますが, 他のバージョンを使うため engine.path = '~/anaconda/bin/python' などと指定することもできます[*26]. engine.path もまたパスのリストを与えられます. これによってエンジンごとにそれぞれパスを指定することができます. 以下のコードが例です. リストの名前はエンジン名と一致する必要があります.

```
knitr::opts_chunk$set(engine.path = list(python = "~/anaconda/bin/python",
  ruby = "/usr/local/bin/ruby"))
```

- **engine.opts**: (NULL; character) 言語エンジンに与える追加引数. チャンクの段階ではオプションを文字列またはリストで指定することができます.

````
```{bash, engine.opts='-l'}
echo $PATH
```

```{cat, engine.opts = list(file = "my_custom.css")}
h2 {
  color: blue;
}
```
````

グローバルレベルでは, 要素名に言語名を与えた文字列のリストが使用できます. engine.path と同様に, knitr::opts_chunk$set() で引数のテンプレートを作ると便利です.

```
knitr::opts_chunk$set(engine.opts = list(perl = "-Mstrict -Mwarnings",
```

[*26] 訳注: R Markdown の場合, Python のバージョンは reticulate パッケージでも制御できます. むしろそちらをつかったほうが便利だと思われます.

```
02    bash = "-o errexit"))
```

各エンジンはそれぞれ自身の engine.opts を持ち, 固有のオプションを定義します. 言語エンジンのドキュメントを調べるべきでしょう. R Markdown クックブックには cat[*27][*28], sass/scss[*29][*30]エンジンの例が掲載されています.

A.1.10 オプションテンプレート関連

- **opts.label**: (NULL; character).: knitr::opts_template でのオプションのラベルです. オプションセットのラベルは knitr::opts_template で設定できます (?knitr::opts_template を参照してください). このオプションにより, 頻繁に使うチャンクオプションのタイピング労力を削減できます.
  訳注: 例えば次のように, echo=F を設定するテンプレート noecho をどこかで作成したとします. すると, 以降のチャンクで opts.label="noecho" を設定すると opts_template で設定した noecho のオプションが全て適用されます. もちろん複数のオプションをまとめることもできるので, 定番の設定を使いまわすのが簡単になります.

```
01  knitr::opts_template$set(noecho = list(echo = F))
```

A.1.11 ソースコードの抽出関連

- **purl**: (TRUE; logical).: ソースドキュメントから knitr::purl() でソースコードを取りだす時, このチャンクを含めるか除外するかどうかです.

A.1.12 その他のチャンクオプション

- **R.options**: (NULL; list).: コードチャンク内でのローカルな R オプションを指定します. これらは options() によってこのコードチャンクの直前に一時的に設定され, 実行後に戻され

[*27] https://bookdown.org/yihui/rmarkdown-cookbook/eng-cat.html
[*28] 翻訳版: https://gedevan-aleksizde.github.io/rmarkdown-cookbook/eng-cat.html
[*29] https://bookdown.org/yihui/rmarkdown-cookbook/eng-sass.html
[*30] 翻訳版: https://gedevan-aleksizde.github.io/rmarkdown-cookbook/eng-sass.html

ます.

A.2 パッケージオプション一覧

パッケージオプションは knitr::opts_knit[*31] を使用することで変更できます. **knitr::opts_chunk と混同しないでください**. 使用例は以下のとおりです.

```
```{.r .numberLines .lineAnchors}
knitr::opts_knit$set(progress = TRUE, verbose = TRUE)
```
```

別の方法として, R の基本関数である options() を使ってパッケージオプションを設定する場合は ?knitr::opts_knit を参照してください.

可能なオプションは次のとおりです.

- **aliases**: (NULL; character).: チャンクオプションのエイリアスを指定する名前付きベクトルです. 例えば c(h = 'fig.height', w = 'fig.width') は **knitr** に h は fig.height w は fig.width と同じ意味だと認識させます. このオプションは名前の長いチャンクオプションのタイピング労力を削減できます.
- **base.dir**: (NULL; character).: グラフを生成する際のディレクトリの絶対パスです.
- **base.url**: (NULL; character).: HTML ページに掲載する画像のベース URL です.
- **concordance**: (FALSE; logical).: この機能は RStudio によって実装されている機能で, .Rnw でのみ有効です. 入力ファイルの行番号に対応した行番号を出力ファイルに書き出すかどうかを指定します. これにより, 出力から入力の誘導が可能になり, 特に LaTeX のエラー発生時に役に立ちます.
- **eval.after**: (c('fig.cap', 'fig.alt';character).: **オプション名の文字ベクトルを指定します. このオプションはチャンクが評価された******後で******評価され, 他の全てのオプションはチャンクが評価される前に評価されます. 例えば** eval.after = 'fig.cap' **が指定されているときに** fig.cap = paste('p-value is', t.test(x)$p.value) **とすると,**eval.after **にはチャンクの評価後の** x' の値が使用されます.
- **global.par**: (FALSE; logical).: TRUE にすると, それ以前のコードチャンクでの par() での設定が引き継がれます (もちろん, この設定は R グラフィックスのみで有効です). デフォルトでは **knitr** はグラフの記録のために新規のグラフィックデバイスを開き, コードの評価後に閉じるため, par() による設定はその都度破棄されます.

[*31] https://gedevan-aleksizde.github.io/knitr-doc-ja/objects.html#objects

- **header**: (NULL; character).: 文書の開始前に挿入するテキストを指定します. (例えば, LaTeX ならば \documentclass{article} の直後, HTML ならば <head> タグの直後). このオプションは LaTeX プリアンブルや HTML ヘッダでコマンドやスタイルの定義をするのに有用です. ドキュメントの開始地点は knitr::knit_patterns$get('document.begin') で知ることができます. このオプションは .Rnw と .Rhtml 限定の機能です[*32].
- label.prefix: (c(table = 'tab:'); character) ラベルの接頭語を指定します. 現時点では kable::kable() によって生成される表のラベルに対する接頭語のみサポートしています.
- **latex.options.color**, **latex.options.graphicx**: (NULL).: それぞれ LaTeX パッケージの **color** と **graphicx** に対するオプションを指定します. これらのオプションは .Rnw 限定の機能です[*33].
- **latex.tilde** (NULL): .Rnw 文書のハイライト出力部でのチルダを表す LaTeX コマンドです (使用例は issue #1992[*34] を見てください).
- **out.format**: (NULL; character).: 可能な値は latex, sweave, html, markdown, jekyll です. このオプションは入力ファイル名に応じて自動で決定され, 自動設定されるフック関数に影響します. 例えば ?knitr::render_latex を参考にしてください. このオプションは knitr::knit() が実行される**前に**設定する必要があります (文書内で設定しても機能しません).
- **progress**: (TRUE; logical).: knitr::knit() の実行中にプログレスバーを表示するかどうかを指定します.
- **root.dir**: (NULL; character).: コードチャンク評価時のルートディレクトリを指定します. NULL の場合, 入力ファイルと同じ場所が指定されます.
- **self.contained**: (TRUE; logical).: 出力する文書が自己完結的であるべきかどうかを指定します (.tex ファイルにスタイルを書き出すか, html に CSS を書き出すか). このオプションは .Rnw と .Rhtml でのみ機能します[*35].
- **unnamed.chunk.label**: (unnamed-chunk; character).: ラベルを設定していないチャンクのラベルの接頭語を指定します.
- **upload.fun**: (identity; function).: ファイルパスを引数にとり, ファイルに対して処理を行い出力フォーマットが HTML または Markdown の場合に文字列を返す関数を指定します. 典型的な使い方として, 画像をアップロードしそのリンクを返す関数を指定します. 例えば knitr::opts_knit$set(upload.fun = knitr::imgur_upload) でファイルを http://imgur.com にアップロードできます (?knitr::imgur_upload を参照してください).
- **verbose**: (FALSE; logical).: 情報を冗長に詳細するか (例えば各チャンクで実行された R コードやメッセージログなど), チャンクラベルとオプションのみ表示するかを指定します.

[*32] 訳注: R Markdown ではヘッダの設定は YAML フロントマターで行います
[*33] 訳注: R Markdown ではこの機能もやはり YAML フロントマターが担当しています
[*34] https://github.com/yihui/knitr/issues/1992
[*35] 訳注: R Markdown では出力フォーマット関数に同様のオプションが用意されていることが多いです

Bibliography

[1] Adler, Daniel, Duncan Murdoch (2021). "*rgl: 3D Visualization Using OpenGL*". R package version 0.107.14. URL: here[*36].

[2] Allaire, JJ, Rich Iannone, Alison Presmanes Hill, Yihui Xie (2021). "*distill: R Markdown Format for Scientific and Technical Writing*". R package version 1.2. URL: here[*37].

[3] Allaire, JJ, Yihui Xie, Christophe Dervieux, R Foundation, Hadley Wickham, Journal of Statistical Software, Ramnath Vaidyanathan, Association for Computing Machinery, Carl Boettiger, Elsevier et al. (2021). "*rticles: Article Formats for R Markdown*". R package version 0.21. URL: here[*38].

[4] Allaire, JJ, Yihui Xie, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, Hadley Wickham, Joe Cheng, Winston Chang, Richard Iannone (2021). "*rmarkdown: Dynamic Documents for R*". R package version 2.11. URL: here[*39].

[5] Anderson, Daniel, Andrew Heiss, Jay Sumners (2021). "*equatiomatic: Transform Models into LaTeX Equations*". R package version 0.2.0. URL: here[*40].

[6] Attali, Dean (2016). "*ezknitr: Avoid the Typical Working Directory Pain When Using knitr*". R package version 0.6. URL: here[*41].

[7] Barnier, Julien (2021). "*rmdformats: HTML Output Formats and Templates for rmarkdown Documents*". R package version 1.0.2. URL: here[*42].

[8] Barrett, Malcolm (2021). "*ggdag: Analyze and Create Elegant Directed Acyclic Graphs*". R package version 0.2.3. URL: here[*43].

[*36] https://CRAN.R-project.org/package=rgl
[*37] https://CRAN.R-project.org/package=distill
[*38] https://github.com/rstudio/rticles
[*39] https://CRAN.R-project.org/package=rmarkdown
[*40] https://github.com/datalorax/equatiomatic
[*41] https://github.com/ropenscilabs/ezknitr
[*42] https://github.com/juba/rmdformats
[*43] https://github.com/malcolmbarrett/ggdag

[9] Blischak, John, Peter Carbonetto, Matthew Stephens (2020). "*workflowr: A Framework for Reproducible and Collaborative Data Science*". R package version 1.6.2. URL: here[*44].

[10] Blischak, John D, Peter Carbonetto, Matthew Stephens (2019). "Creating and sharing reproducible research code the workflowr way [version 1; peer review: 3 approved]". In: *F1000Research* 8.1749. DOI: 10.12688/f1000research.20843.1[*45]. URL: here[*46].

[11] Bodwin, Kelly, Hunter Glanz (2020). "*flair: Highlight, Annotate, and Format your R Source Code*". R package version 0.0.2. URL: here[*47].

[12] Boettiger, Carl (2021). "*knitcitations: Citations for Knitr Markdown Files*". R package version 1.0.12. URL: here[*48].

[13] Chang, Winston (2019). "*webshot: Take Screenshots of Web Pages*". R package version 0.5.2. URL: here[*49].

[14] Cheng, Joe, Timothy Mastny, Richard Iannone, Barret Schloerke, Carson Sievert (2021). "*sass: Syntactically Awesome Style Sheets (Sass)*". R package version 0.4.0. URL: here[*50].

[15] D'Agostino McGowan, Lucy, Jennifer Bryan (2021). "*googledrive: An Interface to Google Drive*". R package version 2.0.0. URL: here[*51].

[16] Dahl, David B. David Scott, Charles Roosen, Arni Magnusson, Jonathan Swinton (2019). "*xtable: Export Tables to LaTeX or HTML*". R package version 1.8-4. URL: here[*52].

[17] Daróczi, Gergely, Roman Tsegelskyi (2021). "*pander: An R Pandoc Writer*". R package version 0.6.4. URL: here[*53].

[18] de Vries, Andrie, Javier Luraschi (2020). "*nomnoml: Sassy UML Diagrams*". R package version 0.2.3. URL: here[*54].

[19] El Hattab, Hakim, JJ Allaire (2017). "*revealjs: R Markdown Format for reveal.js Presentations*". R package version 0.9. URL: here[*55].

[*44] https://github.com/jdblischak/workflowr

[*45] https://doi.org/10.12688/f1000research.20843.1

[*46] https://doi.org/10.12688/f1000research.20843.1

[*47] https://CRAN.R-project.org/package=flair

[*48] https://github.com/cboettig/knitcitations

[*49] https://github.com/wch/webshot/

[*50] https://github.com/rstudio/sass

[*51] https://CRAN.R-project.org/package=googledrive

[*52] http://xtable.r-forge.r-project.org/

[*53] https://rapporter.github.io/pander/

[*54] https://github.com/rstudio/nomnoml

[*55] https://github.com/rstudio/revealjs

[20] Garbett, Shawn (2020). "*tangram: The Grammar of Tables*". R package version 0.7.1. URL: here[*56].

[21] Garmonsway, Duncan (2021). "*govdown: GOV.UK Style Templates for R Markdown*". R package version 0.10.1. URL: here[*57].

[22] Gohel, David (2021a). "*flextable: Functions for Tabular Reporting*". R package version 0.6.8. URL: here[*58].

[23] — (2021b). "*officer: Manipulation of Microsoft Word and PowerPoint Documents*". R package version 0.4.0. URL: here[*59].

[24] Gohel, David, Noam Ross (2021). "*officedown: Enhanced R Markdown Format for Word and PowerPoint*". R package version 0.2.2. URL: here[*60].

[25] Hlavac, Marek (2018). "*stargazer: Well-Formatted Regression and Summary Statistics Tables*". R package version 5.2.2. URL: here[*61].

[26] Hugh-Jones, David (2021). "*huxtable: Easily Create and Style Tables for LaTeX, HTML and Other Formats*". R package version 5.4.0. URL: here[*62].

[27] Iannone, Richard (2020). "*DiagrammeR: Graph/Network Visualization*". R package version 1.0.6.1. URL: here[*63].

[28] Iannone, Richard, JJ Allaire, Barbara Borges (2020). "*flexdashboard: R Markdown Format for Flexible Dashboards*". R package version 0.5.2. URL: here[*64].

[29] Iannone, Richard, Joe Cheng (2020). "*blastula: Easily Send HTML Email Messages*". R package version 0.3.2. URL: here[*65].

[30] Iannone, Richard, Joe Cheng, Barret Schloerke (2021). "*gt: Easily Create Presentation-Ready Display Tables*". R package version 0.3.1. URL: here[*66].

[31] Kothe, Emily, Claudio Zandonella Callegher, Filippo Gambarota, Janosch Linkersdörfer, Mathew Ling (2021). "*trackdown: Collaborative Writing and Editing of R Markdown (or Sweave) Documents in Google Drive*". R package version 1.0.0. URL: here[*67].

[*56] https://github.com/spgarbet/tangram
[*57] https://ukgovdatascience.github.io/govdown/
[*58] https://CRAN.R-project.org/package=flextable
[*59] https://CRAN.R-project.org/package=officer
[*60] https://CRAN.R-project.org/package=officedown
[*61] https://CRAN.R-project.org/package=stargazer
[*62] https://hughjonesd.github.io/huxtable/
[*63] https://github.com/rich-iannone/DiagrammeR
[*64] http://rmarkdown.rstudio.com/flexdashboard
[*65] https://github.com/rich-iannone/blastula
[*66] https://CRAN.R-project.org/package=gt
[*67] https://CRAN.R-project.org/package=trackdown

[32] Lawrence, Michael (2020). "*cairoDevice: Embeddable Cairo Graphics Device Driver*". R package version 2.28.2. URL: here[*68].

[33] Lin, Greg (2020). "*reactable: Interactive Data Tables Based on React Table*". R package version 0.2.3. URL: here[*69].

[34] Mattioni Maturana, Felipe (2020). "*downloadthis: Implement Download Buttons in rmarkdown*". R package version 0.2.1. URL: here[*70].

[35] McPherson, Jonathan, JJ Allaire (2021). "*rsconnect: Deployment Interface for R Markdown Documents and Shiny Applications*". R package version 0.8.24. URL: here[*71].

[36] Moon, Keon-Woong (2020). "*ztable: Zebra-Striped Tables in LaTeX and HTML Formats*". R package version 0.2.2. URL: here[*72].

[37] Müller, Kirill (2020). "*here: A Simpler Way to Find Your Files*". https://here.r-lib.org/.

[38] Müller, Kirill, Lorenz Walthert (2021). "*styler: Non-Invasive Pretty Printing of R Code*". R package version 1.5.1. URL: here[*73].

[39] Murdoch, Duncan (2020). "*tables: Formula-Driven Table Generation*". R package version 0.9.6. URL: here[*74].

[40] Nutter, Benjamin (2021). "*pixiedust: Tables so Beautifully Fine-Tuned You Will Believe It's Magic*". R package version 0.9.1. URL: here[*75].

[41] Oller Moreno, Sergio (2020). "*condformat: Conditional Formatting in Data Frames*". R package version 0.9.0. URL: here[*76].

[42] Ooms, Jeroen (2021a). "*gifski: Highest Quality GIF Encoder*". R package version 1.4.3-1. URL: here[*77].

[43] — (2021b). "*magick: Advanced Graphics and Image-Processing in R*". https://docs.ropensci.org/magick/ (website) https://github.com/ropensci/magick (devel).

[44] Ooms, Jeroen, Jim Hester (2020). "*spelling: Tools for Spell Checking in R*". R package version 2.2. URL: here[*78].

[45] Owen, Jonathan (2021). "*rhandsontable: Interface to the Handsontable.js Library*". R package version 0.3.8. URL: here[*79].

[*68] https://CRAN.R-project.org/package=cairoDevice
[*69] https://CRAN.R-project.org/package=reactable
[*70] https://github.com/fmmattioni/downloadthis
[*71] https://github.com/rstudio/rsconnect
[*72] https://github.com/cardiomoon/ztable
[*73] https://CRAN.R-project.org/package=styler
[*74] https://r-forge.r-project.org/projects/tables/
[*75] https://github.com/nutterb/pixiedust
[*76] http://github.com/zeehio/condformat
[*77] https://CRAN.R-project.org/package=gifski
[*78] https://CRAN.R-project.org/package=spelling
[*79] http://jrowen.github.io/rhandsontable/

[46] Pedersen, Thomas Lin, David Robinson (2020). "*gganimate: A Grammar of Animated Graphics*". R package version 1.0.7. URL: here[*80].

[47] R Core Team (2021). "*R: A Language and Environment for Statistical Computing*". R Foundation for Statistical Computing. Vienna, Austria. URL: here[*81].

[48] Ren, Kun, Kenton Russell (2021). "*formattable: Create Formattable Data Structures*". R package version 0.2.1. URL: here[*82].

[49] Robinson, David, Alex Hayes, Simon Couch (2021). "*broom: Convert Statistical Objects into Tidy Tibbles*". R package version 0.7.9. URL: here[*83].

[50] Schloerke, Barret, JJ Allaire, Barbara Borges (2020). "*learnr: Interactive Tutorials for R*". R package version 0.10.1. URL: here[*84].

[51] Sharpsteen, Charlie, Cameron Bracken (2020). "*tikzDevice: R Graphics Output in LaTeX Format*". R package version 0.12.3.1. URL: here[*85].

[52] Sjoberg, Daniel D. Michael Curry, Margie Hannum, Joseph Larmarange, Karissa Whiting, Emily C. Zabor (2021). "*gtsummary: Presentation-Ready Data Summary and Analytic Result Tables*". R package version 1.4.2. URL: here[*86].

[53] Soetaert, Karline (2020). "*diagram: Functions for Visualising Simple Graphs (Networks), Plotting Flow Diagrams*". R package version 1.6.5. URL: here[*87].

[54] Stephens, Jeremy, Kirill Simonov, Yihui Xie, Zhuoer Dong, Hadley Wickham, Jeffrey Horner, reikoch, Will Beasley, Brendan O'Connor, Gregory R. Warnes (2020). "*yaml: Methods to Convert R Data to YAML and Back*". R package version 2.2.1. URL: here[*88].

[55] Strayer, Nick, Javier Luraschi, JJ Allaire (2020). "*r2d3: Interface to D3 Visualizations*". R package version 0.2.5. URL: here[*89].

[56] Textor, Johannes, Benito van der Zander, Ankur Ankan (2021). "*dagitty: Graphical Analysis of Structural Causal Models*". R package version 0.3-1. URL: here[*90].

[57] Urbanek, Simon, Jeffrey Horner (2020). "*Cairo: R Graphics Device using Cairo Graphics Library for Creating High-Quality Bitmap (PNG, JPEG, TIFF), Vector (PDF, SVG,*

[*80] https://CRAN.R-project.org/package=gganimate

[*81] https://www.R-project.org/

[*82] https://CRAN.R-project.org/package=formattable

[*83] https://CRAN.R-project.org/package=broom

[*84] https://CRAN.R-project.org/package=learnr

[*85] https://github.com/daqana/tikzDevice

[*86] https://CRAN.R-project.org/package=gtsummary

[*87] https://CRAN.R-project.org/package=diagram

[*88] https://github.com/viking/r-yaml/

[*89] https://github.com/rstudio/r2d3

[*90] https://CRAN.R-project.org/package=dagitty

PostScript) and Display (X11 and Win32) Output". R package version 1.5-12.2. URL: here[*91].

[58] Ushey, Kevin, JJ Allaire, Yuan Tang (2021). "*reticulate: Interface to Python*". R package version 1.20. URL: here[*92].

[59] Wickham, Hadley, Jennifer Bryan (2021). "*usethis: Automate Package and Project Setup*". R package version 2.0.1. URL: here[*93].

[60] Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, Dewey Dunnington (2021). "*ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*". R package version 3.3.5. URL: here[*94].

[61] Wickham, Hadley, Peter Danenberg, Gábor Csárdi, Manuel Eugster (2021). "*roxygen2: In-Line Documentation for R*". R package version 7.1.2. URL: here[*95].

[62] Wickham, Hadley, Garrett Grolemund (2016) "*R for Data Science*". O'Reilly Media, Inc.

. [63] Wickham, Hadley, Garrett Grolemund (2016) "*R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*". O'Reilly. ISBN: 978-1-4919-1039-9 978-1-4919-1036-8. URL: here[*96]. 黒川利明・大橋真也 訳『R で始めるデータサイエンス』. 2017 年. オライリー・ジャパン

. [64] Wickham, Hadley, Lionel Henry, Thomas Lin Pedersen, T Jake Luciani, Matthieu Decorde, Vaudor Lise (2021). "*svglite: An SVG Graphics Device*". R package version 2.0.0. URL: here[*97].

[65] Wickham, Hadley, Jay Hesselberth (2020). "*pkgdown: Make Static HTML Documentation for a Package*". R package version 1.6.1. URL: here[*98].

[66] Xie, Yihui (2018). "*animation: A Gallery of Animations in Statistics and Utilities to Create Animations*". R package version 2.6. URL: here[*99].

[67] Xie, Yihui (2016) "*bookdown: Authoring Books and Technical Documents with R Markdown*". Chapman and Hall/CRC. URL: here[*100]. ISBN 978-1138700109

. [68] — (2021a). "*bookdown: Authoring Books and Technical Documents with R Markdown*". https://github.com/rstudio/bookdown.

[*91] http://www.rforge.net/Cairo/

[*92] https://github.com/rstudio/reticulate

[*93] https://CRAN.R-project.org/package=usethis

[*94] https://CRAN.R-project.org/package=ggplot2

[*95] https://CRAN.R-project.org/package=roxygen2

[*96] https://r4ds.had.co.nz/

[*97] https://CRAN.R-project.org/package=svglite

[*98] https://CRAN.R-project.org/package=pkgdown

[*99] https://yihui.name/animation

[*100] https://bookdown.org/yihui/bookdown

[69] Xie, Yihui (2015) "*Dynamic Documents with R and knitr*". Chapman and Hall/CRC. URL: here[*101]. ISBN 978-1498716963

. [70] — (2021b). "*formatR: Format R Code Automatically*". R package version 1.11. URL: here[*102].

[71] — (2021c). "*knitr: A General-Purpose Package for Dynamic Report Generation in R*". R package version 1.34. URL: here[*103].

[72] — (2021d). "*printr: Automatically Print R Objects to Appropriate Formats According to the knitr Output Format*". R package version 0.1.1. URL: here[*104].

[73] — (2019). "TinyTeX: A lightweight, cross-platform, and easy-to-maintain LaTeX distribution based on TeX Live". In: *TUGboat* 1, pp. 30–32. URL: here[*105].

[74] — (2021e). "*tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents*". R package version 0.33. URL: here[*106].

[75] — (2021f). "*xaringan: Presentation Ninja*". R package version 0.22. URL: here[*107].

[76] — (2021g). "*xfun: Supporting Functions for Packages Maintained by Yihui Xie*". R package version 0.26. URL: here[*108].

[77] Xie, Yihui, J.J. Allaire, Garrett Grolemund (2018) "*R Markdown: The Definitive Guide*". Chapman and Hall/CRC. URL: here[*109]. ISBN 9781138359338

. [78] Xie, Yihui, Joe Cheng, Xianying Tan (2021). "*DT: A Wrapper of the JavaScript Library DataTables*". R package version 0.19. URL: here[*110].

[79] Xie, Yihui, Christophe Dervieux, Alison Presmanes Hill (2021). "*blogdown: Create Blogs and Websites with R Markdown*". https://github.com/rstudio/blogdown.

[80] Xie, Yihui, Alison Presmanes Hill, Amber Thomas (2017) "*blogdown: Creating Websites with R Markdown*". Chapman and Hall/CRC. URL: here[*111]. ISBN 978-0815363729

. [81] Xie, Yihui, Romain Lesur, Brent Thorne, Xianying Tan (2021). "*pagedown: Paginate the HTML Output of R Markdown with CSS for Print*". R package version 0.15. URL: here[*112].

[*101] https://yihui.org/knitr/
[*102] https://github.com/yihui/formatR
[*103] https://yihui.org/knitr/
[*104] https://yihui.org/printr/
[*105] http://tug.org/TUGboat/Contents/contents40-1.html
[*106] https://github.com/yihui/tinytex
[*107] https://github.com/yihui/xaringan
[*108] https://github.com/yihui/xfun
[*109] https://bookdown.org/yihui/rmarkdown
[*110] https://github.com/rstudio/DT
[*111] https://bookdown.org/yihui/blogdown/
[*112] https://github.com/rstudio/pagedown

[82] Zhu, Hao (2021). "*kableExtra: Construct Complex Table with kable and Pipe Syntax*". R package version 1.3.4. URL: here[*113].

[*113] https://CRAN.R-project.org/package=kableExtra

索引