

rmdjaによる多様な形式の日本語技術文書の作成

Katagiri, Satoshi (ill-identified)

2021/06/25

目次

第 I 部 イントロダクション	9
序文	11
本稿の目的	13
R Markdown の現状と問題意識	13
rmdja の利点	14
R 使用経験のないユーザへ	16
第 II 部 最低限のチュートリアル	17
第 1 章 クイックスタート	19
第 2 章 下準備	21
2.1 より丁寧なインストール解説	21
2.2 書籍形式のビルド操作	21
第 III 部 R Markdown と Bookdown の基本機能	25
このパートの概要	27
第 3 章 静的なコンテンツの作成	29
3.1 Markdown の基本構文	29
3.1.1 インラインでの書式変更	29
3.1.2 ブロック要素	30
3.2 Markdown を使った図表の挿入	32

3.2.1	コメントアウト	32
3.3	数式	33
3.4	カスタムブロック	34
3.5	脚注	35
3.6	改行・改頁(改ページ)・改丁	36
第4章	動的なコンテンツの作成	37
4.1	プログラムチャンク	37
4.2	プログラムで数式を生成する	40
4.3	プログラムを使った図の挿入	41
4.4	(WIP): デフォルトフォントの設定	42
4.5	TODO: 図のレイアウト設定	43
4.6	R プログラムを使った表の装飾	44
第5章	相互参照と引用	51
5.1	相互参照	51
5.1.1	図表や式へのアンカーリンク	51
5.1.2	表への相互参照	51
5.1.3	章への相互参照	52
5.1.4	特殊な相互参照	52
5.2	文献引用	53
5.2.1	文献引用スタイルのカスタマイズ	54
5.2.2	文献リスト生成エンジンの違いについて	55
第6章	(WIP) 簡単なレイアウト・スタイル変更	57
6.1	フォント変更	57
6.1.1	どのフォントを使用すべきか	59
6.2	YAML フロントマターの設定によるスタイル変更	60
6.2.1	ハイパーリンクの配色変更	60
6.2.2	L ^A T _E X エンジンの変更	61
6.2.3	文書クラスのカスタマイズ	61
6.2.4	LaTeX のカスタマイズ	62

6.3	チャンク/コードおよび出力ブロックのスタイルの一括変更	63
6.3.1	行番号の表示	63
6.3.2	コードブロックのページまたぎ禁止	64
第 7 章	(WIP) <code>rmdja</code> による文書作成支援機能	65
7.0.1	クリエイティブ・コモンズの表記	65
7.0.2	ルビ表記	65
第 IV 部	応用編	67
このパートについて		69
第 8 章	様々なグラフィックプログラムの埋め込み	71
8.1	<code>tikz</code> を使う	71
8.2	<code>Asymptote</code> を使う	71
8.3	(TODO) その他のプログラム	72
8.4	(TODO) その他の R プログラム	72
8.5	DOT 言語とグラフィカルモデル	72
第 9 章	表のデザイン	75
9.1	<code>kableExtra</code> による表のスタイルのカスタマイズ	75
9.2	<code>formattable</code> パッケージとの併用	78
9.3	<code>huxtable</code> パッケージによる作表	78
9.4	<code>TeX/HTML</code> を出力する関数	80
9.5	その他の作表パッケージ	82
第 10 章	文献引用の詳細設定	83
10.1	(u)pB <small>B</small> T <small>E</small> X を使う	83
10.1.1	(TODO) <code>pandoc-citeproc</code> と CSL について	84
10.1.2	(WIP) BibLaTeX について	84
第 11 章	(TODO) Web アプレットの挿入	87
11.0.1	TODO: <code>plotly</code>	87
11.0.2	TODO: <code>shiny</code>	87

第 12 章 Python スクリプトの埋め込み	89
12.1 Python のグラフィックに関する制約	90
第 V 部 製本と多様な形式への対応	93
第 13 章 PDF の文書クラス	95
13.1 プレゼンテーション資料の作成	96
13.2 (WIP) 卒業論文の作成	97
13.3 (WIP) 小説の執筆	98
第 14 章 製本方法の詳細	101
14.1 ファイル構成	101
14.1.1 _output.yml	101
14.1.2 _bookdown.yml	102
第 15 章 出力形式による表現の限界	105
15.1 HTML と PDF で処理を場合分けする	105
15.2 絵文字の出力	106
15.3 画像の保存形式	106
15.4 デフォルトの保存形式	106
第 16 章 製本した文書を配布する	107
16.1 Wep ページのホスティング	107
16.2 (WIP) 入稿するには	108
16.2.1 トンボの表示	108
16.2.2 (TODO) 隠しノンブルとヘッダ・フッタのカスタマイズ	109
16.2.3 フォントの埋め込み	109
第 VI 部 デバッグとトラブルシュート	111
このパートで説明すること	113

第 17 章 製本時のエラーへの対処	115
17.1 エラーがどのタイミングで発生したかを特定する	115
17.2 YAML フロントマターを確認する	116
17.3 PDF 生成時のエラーを確認する	117
17.4 よくあるエラーメッセージ	117
17.4.1 The File XXX.Rmd Exists.	117
17.4.2 No Site Generator Found.	118
第 18 章 その他のトラブルシュート	119
18.1 コードブロックや出力テキストの折り返し・改行位置がおかしい	119
18.2 (TODO) Windows 特有の問題	119
付録 A デフォルト値の自動調整	121
A.1 デフォルトのフォント	121
A.2 チャンクオプションのデフォルト設定	122
A.3 コードブロックの整形と自動折り返し	122
付録 B PDF の組版に関する細かい話	125
B.1 画像の配置	125
B.2 取り消し線	125
B.3 TODO: しかし英文で書きたい場合	126
付録 C 参考文献リストの書式にこだわる: jecon.bst	127
付録 D fontregisterer でグラフ用フォントを自動登録	129
参考文献	129

第Ⅰ部

イントロダクション

序文



注意!

現在、このドキュメントは煩雑で **rmdja** パッケージの更新に対して追いついていません。あまり信用しないでください。

R Markdown 全般の使用例については以下を参考にしてください。これらは R Markdown 開発の中心人物が執筆しているためこのページよりも信頼できます。

- Xie “[knitr: Elegant, flexible, and fast dynamic report generation with R](#)” (拙訳: 謝『knitr: R による美麗で柔軟そして高速な動的レポート生成』)
- Xie “[TinyTeX: A lightweight, cross-platform, portable, and easy-to-maintain LaTeX distribution based on TeX Live](#)” (拙訳:『TinyTeX 非公式日本語版ドキュメント』)
- Xie, Dervieux, and Riederer “[R Markdown Cookbook](#)” (拙訳: 謝・デルヴュー・リーデラー『R Markdown クックブック』が公開されています)
- Xie, Allaire and Grolemund “[R Markdown: The Definitive Guide](#)” (邦訳未刊、近日作成予定)
- Xie “[bookdown: Authoring Books and Technical Documents with R Markdown](#)” (邦訳未刊、近日作成予定)

日本語で書かれた書籍には以下があります。

- 高橋康介 (2018) 『再現可能性のすゝめ RStudio によるデータ解析とレポート作成』共立出版

一旦残り 2 作の翻訳を済ませてから、これらで言及されていない使用法(特に PDF での日本語の適切な表示方法など)について本稿で補足説明したいと思います。それまではこちらのドキュメントはほとんど更新されないことに注意してください。

長大な技術文書や良質な技術文書を作成するには手間がかかる。しかし時間をかけて良い文書になるわけではない。無駄な手間を省き、効率よく快適に文書を作成するべきである。

たとえばこういう経験はないだろうか。

- プログラムの解説のため、外部サービスでシンタックスハイライトしてもらったテキストをコピペーストで貼り付ける
- グラフや図解を専用アプリケーションで作成し貼り付ける。修正のたびに貼り付け直す
- 図表に言及する際に「図 1」「表 2」と番号をタイプし、参照先へハイパーリンクを指定する
- 本文中で引用した参考文献のリストを巻末にコピペーストし、過不足がないか目視で確認
- $\sum_{k=1}^K \int_0^\infty f_k(x) dx$ などといった複雑な数式はプレーンテキストや HTML では表現できないため、画像を生成して貼り付ける
- 冒頭にかっこいいエピグラフを掲載したいので、1 時間かけて特別に枠やフォントを作成した
- 市販のワードプロセッサで作成した文書を渡したら、レイアウトが崩れて読めないと言われた

本稿は文書作成者をこのような数え切れないブルシットから解放するのが目的である。

R Markdown (`rmarkdown`) は、R プログラムを埋め込んだ動的なドキュメントから pandoc を利用して PDF や HTML 形式の文書を作成するパッケージであり、数式、図表の挿入、シンタックスハイライトされたプログラムなどを簡単な記述で掲載できる。名前の通り、その基本構文は Markdown である。よって、Markdown と R の知識が最低限あれば（R プログラムが必要ないなら Markdown だけでも）文書を作成することができる。

bookdown パッケージは `rmarkdown` をもとに、ページ数の多い文書を作成し、配布するための機能を拡張したものである。しかし、PDF の出力に関しては欧文を前提としたフォーマットを使用しているため、日本語の適切な表示（組版やフォントの埋め込みなど）のできる文書を作成するには高いハードルが存在した。

本稿では、`rmarkdown` および `bookdown` で日本語文書を作成する際の設定を容易にしたパッケージ `rmdja` を利用した日本語技術文書の作成方法を解説する。現在、書籍、論文、プレゼンテーションの体裁での文書を作成するテンプレートが用意されており、このドキュメント自体も `rmdja` を利用して作成されている。

本稿の目的

R Markdown の現状と問題意識

R Markdown を利用した文書作成方法について、すでにそれなりの数と質の日本語資料が存在する。R Markdown で HTML ファイルのみを作成する場合、日本語であるか欧文であるかはあまり気にする必要はない。HTML であれば既存の資料でも十分に役に立つ。

- kazutan 『R Markdown 再入門』
- kazutan 『R Markdown によるスライド生成』

しかし、PDF を出力する、あるいは HTML と PDF を同時に出力したい、となると、組版に関して細かな設定が必要になるため難易度は一気に上昇する。

実のところ PDF でも日本語を表示する最低限の設定は、YAML フロントマターだけで行える。例えば Atusy 氏が『R Markdown + XeLaTeX で日本語含め好きなフォントを使って PDF を出力する』で紹介しているが、よりシンプルな書き方もできる。

```
output: pdf_document:
  latex_engine: xelatex
documentclass: bxjsarticle
classoption:
  - xelatex
  - ja=standard
  - jafont=noto
```

このままでも、とりあえず文字化けすることなく日本語を表示できる。しかし実際に作ってみると、スタイルを調整したり、画像を埋め込んだりといった細かいカスタマイズが必要になる。すると上記の設定だけではいろいろな障害が立ちはだかり、文書として整ったものにするのは難しい。このままで参考文献リストの表示も不自然なままである。だがこれ以上のカスタマイズは Atusy 氏がやっているようにテンプレートを修正することでしか対処できないものもあり、L^AT_EX に対するそれなりの知識が必要となる。

さらに、同じソースファイルから HTML と PDF を同時に生成すると、また別種の問題が発生する。HTML と PDF は根本的に規格が違うため、様々な場合分け処理が必要であ

り、それは pandoc だけでは対応しきれない。

HTML 出力に限らない R Markdown の全般的な情報は、既に充実した英語の（公式）ドキュメントが多く存在する^{*1}。

- “Dynamic Documents for R • rmarkdown”
- “bookdown demo”
- “bookdown: Authoring Books and Technical Documents with R Markdown”
- “R Markdown Definitive Guide”
- “R Markdown Cookbook”^{*2}

しかしながらこれらを元に 1 からいろいろな調整を施すのはとても骨が折れるため、rmdja パッケージは日本語文書で HTML や PDF を同時に生成する場合の定番の処理をフォーマットに内蔵することにした。

rmdja の利点

従来 LaTeX や Word、あるいは他の媒体で文書を作成していたユーザにとっても、文書の内容から面倒な設定を分離するため、効率的に執筆できる。

Word ユーザにとっては、以下のような利点がある^{*3}。

- 数十、数百ページの文書を書いてもクラッシュすることがあまりない
- 輪郭のはっきりしたベクタ画像を簡単に貼り付けられる
- 図表の配置や相互参照を手動で書く必要がない
- 読み手の環境に依存してレイアウトが崩れにくい PDF ファイルを出力できる

ただし、.docx ファイルの出力はできない。私が Word を持っておらず、R Markdown や pandoc がサポートしていても動作確認のしようがないため。

jupyter は Python のコードチャンクとその結果を簡単に表示できる文書作成ツールである。出力オプションの少なさ（たとえば長大なコードもそのまま掲載されてしまう）や、IDE として見ても機能が少ないとからあまり使い勝手がよくなかったが、rmdja では Python スクリプトの埋め込みにもある程度対応している。

LATEX のユーザー（シンプルなテキストエディタで書いているユーザも、Overleaf や LyX といった強力なエディタを使用しているユーザー）にとっては、LATEX とほぼ同じ構文で数式を入力でき、かつ操作を大きく簡略化でき、実験結果などをソースに埋め込むことが

^{*1} 基本的なことがらの多くは上記を読めば分かるのでここでは基本機能をダイジェストで伝えた上で、これらの資料に書いてない応用技を紹介する。YAML のオプションの意味についてはソースコードにコメントを書いた。以下、単に、BKD と書けば “bookdown: Authoring Books and Technical Documents with R Markdown” (Xie, 2020) を、RDG と書けば “R Markdown: The Definitive GUide” (Xie et al., 2018) を、RCB と書けば “R Markdown Cookbook” (Xie et al., 2020) を指すこととする。

^{*2} 2020/10/19 に書籍としても発売されるらしい。

^{*3} ただし筆者は数年来 Word を使っていなかったため、これらのいくつかは既に改善されているかもしれない。

でき、外部プログラムからいちいちコピペする必要がなくなる。ただし、なるべく選択肢は広げておきたいが、なんでもありではかえって余計なことをしがちである。よって既に作成した beamer フォーマットと同様に、X_ELATEX および LuaLATEXのみの対応を想定している。pLATEX や upLATEX には対応していない。

これまでも R Markdown を使用してきたユーザにとっては、YAML フロントマターに数十行に渡っていた書いていた日本語表示のための設定の多くがフォーマットのデフォルト値になったため、かなり楽になると思われる。

たとえば、

```
bookdown::pdf_book:
  toc_depth: 3
  toc_appendix: true
  toc_bib: true
  latex_engine: xelatex
  keep_tex: true
  keep_md: true
  citation_package: natbib
  pandoc_args:
    - '--top-level-division=chapter'
    - '--extract-media'
    - '.'
  template: XXXXX.tex.template'
  dev: "cairo_pdf"
  out_width: "100%"
  out_height: "100%"
  quote_footer: ["\\VA{", "}{"]'
  extra_dependencies: gentombow
```

のように書いていたものがこうなる。

```
rmdja::pdf_book_ja:
  keep_tex: true
  keep_md: true
  tombow: true
```

さらにチャンクオプションを書いたり場合によっては.tex ファイルのテンプレートすら調整する必要もあった。それらも rmdja 内で調整している。

さらに、作成した文書は PDF 形式で出力することはもちろん、HTML 形式で様々なサイ

トで掲載でき^{*4}たり、電子書籍ファイルとしても出力可能である。このような多様な出力形式への対応しているソフトウェアはあまり例を見ない。

R 使用経験のないユーザへ

R を使わない、あるいはそもそもプログラミングに詳しくない、という人間にもある使用機会がある。たとえば R を普段使わない人間でも bookdown で同人技術書を執筆したという事例がある^{*5}。この事例は主に数式と画像の貼付けのみだから、数式出力に必要な LATEX の知識があればほとんどのことはできてしまう。そして rmdja ではこの事例で言及されている LATEX の設定の多くは自動で制御される。また、小説などはほぼテキストであり、最低限のレイアウトさえ用意すれば数式も、あるいは画像の挿入すらいらないことが多い。rmdja では縦書き文書を PDF で出力する方法も用意している。

^{*4} bookdown 同様に R Markdown で作成した文書をブログ風のフォーマットで出力する blogdown パッケージというものも存在する。

^{*5} <https://teastat.blogspot.com/2019/01/bookdown.html>

第Ⅱ部

最低限のチュートリアル

第1章

クイックスタート

`rmdja` パッケージをインストールする。github からインストールするため, `remotes` パッケージが必要になる。依存している `rmarkdown`, `bookdown`, `knitr` なども同時にインストールされる^{*1}。

```
01 install.packages("remotes")
02 remotes::install_github("Gedevan-Aleksizde/rmdja", repos = NULL)
```

まだ RStudio を使っていないのなら, RStudio 上で作業することを強く推奨する。さらに, もしも R の操作自体にあまり慣れていないのなら, 森知晴『卒業論文のための R 入門』などを読むことを薦める。

加えて, 以下のパッケージが役に立つので気に入ったらインストールしていただきたい。

```
01 install.packages(c("tidyverse", "ggthemes", "citr", "clipr", "kableExtra"))
```

RStudio を起動し, 左上から新規作成を選び, “R Markdown” を選ぶ (図 1.1)。

“From Template” からテンプレートを選択する (1.2)。

現在 (Ver. 0.4.6.9) 用意されているのは以下の 4 つである。

- プрезンテーション用スライド形式のテンプレート - Beamer in Japanese
- 論文形式のテンプレート - pdf article in Japanese
- 書籍形式のテンプレート - pdf book in Japanese
- 縦書き文書のテンプレート - pdf vertical writing in Japanese

動作確認として, 今回はシンプルな論文形式を選ぶ。ファイルを開いたら, 適当な名称で保存し, “knit” ボタンを押すと PDF が作成される。

^{*1} Windows の場合, Rtools をインストールしていないと依存パッケージがインストールされないことがある。Rtools をインストールするか, 依存パッケージを手動でインストールしてほしい

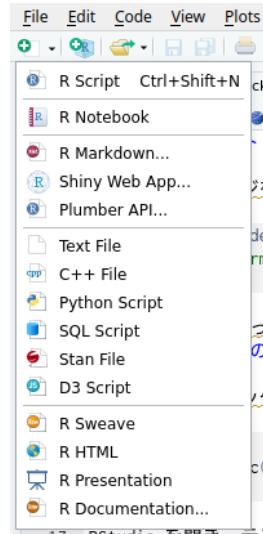


図1.1: 新規作成

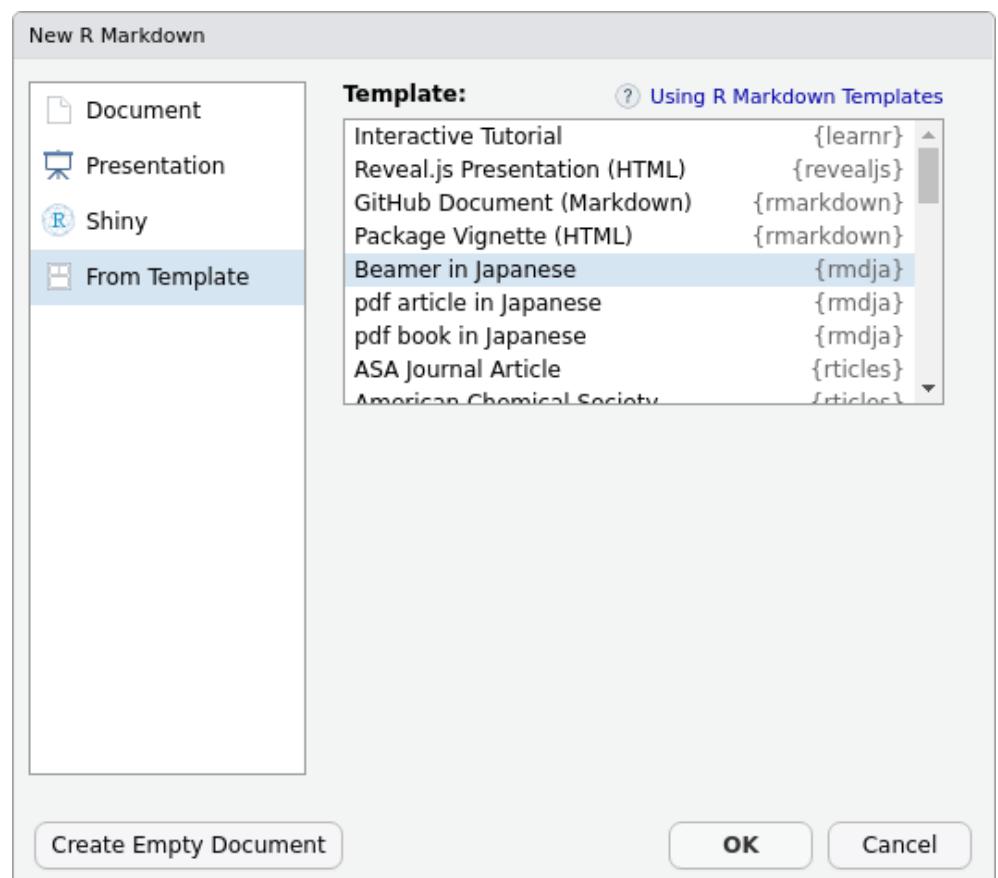


図1.2: R Markdown のテンプレート

第 2 章

下準備

以降は順を追って細かい解説をする。

2.1 より丁寧なインストール解説

文書を生成するのに必要なものをインストールする。

このドキュメントは `rmdja` パッケージに含まれている。よってまずはこれをダウンロードしてほしい。

3種類のテンプレートのうち、`pdf book in Japanese` のみ、文書のビルドのための下準備が追加で必要になるため、その方法を解説する。それ以外は第 1 節で書いたように “knit” ボタンを押すだけで良い。

最低限のファイルやパッケージで動くほうのデモ用ディレクトリをコピーする。ただし、`tidyverse` と `kableExtra` のインストールも必要である。

```

01 file.copy(
02   system.file("resources/examples/bookdown-minimal",
03     package = "rmdja"
04   ),
05   "./",
06   recursive = T
07 )

```

2.2 書籍形式のビルド操作

書籍形式のテンプレートである `pdf book in Japanese` を選択した場合、論文形式やスライド形式とは違いフォルダが作られ、その中に `_bookdown.yml`, `_output.yml` というファイルが作られる。これらは書籍の細かいフォーマットを設定するためのファイルである。新

規作成ファイルも同じフォルダに `index.Rmd` という名前で保存する。この名前は最初に読み込むファイル名のデフォルト名として決まっているため、他の名前で保存すると正しく動作しないことがある。さらに bookdown の文書生成は従来の R Markdown と違い、RStudio の knit ボタンではできない。代わりに、以下の 2通りの方法がある。

1. `bookdown::render_book('index.Rmd', format = "bookdown::gitbook")`などを呼び出す
2. RStudio の Build ペーンを使う

前者の場合は、Rmd ファイルのあるディレクトリに移動して以下の関数を実行する。順に HTML, PDF, epub を出力している

```
01 bookdown::render_book("index.Rmd", "rmdja::gitbook_ja")
02 bookdown::render_book("index.Rmd", "rmdja::pdf_book_ja")
03 bookdown::render_book("index.Rmd", "bookdown::epub_book")
```

コピーしたディレクトリ `bookdown-minimal` を設定する（図 2.1, 2.2）。

Build ペーンの “Build Book” の三角形を押すと、使用できるフォーマット一覧が表示される。これはスライド、縦書き文書、書籍などといった文書の種類と 1 対 1 で対応しているわけではなく、フォーマット関数に対応している。

- HTML 形式 - `rmdja::gitbook_ja`
- PDF 形式 - `rmdja::pdf_book_ja`
- 電子書籍 (EPUB) 形式 - `bookdown::epub_book`

デフォルトでは “All Formats” にチェックが入っているため、これら 3 種類のファイル形式を一度に生成する。

これで `_book` フォルダに出力がされる。

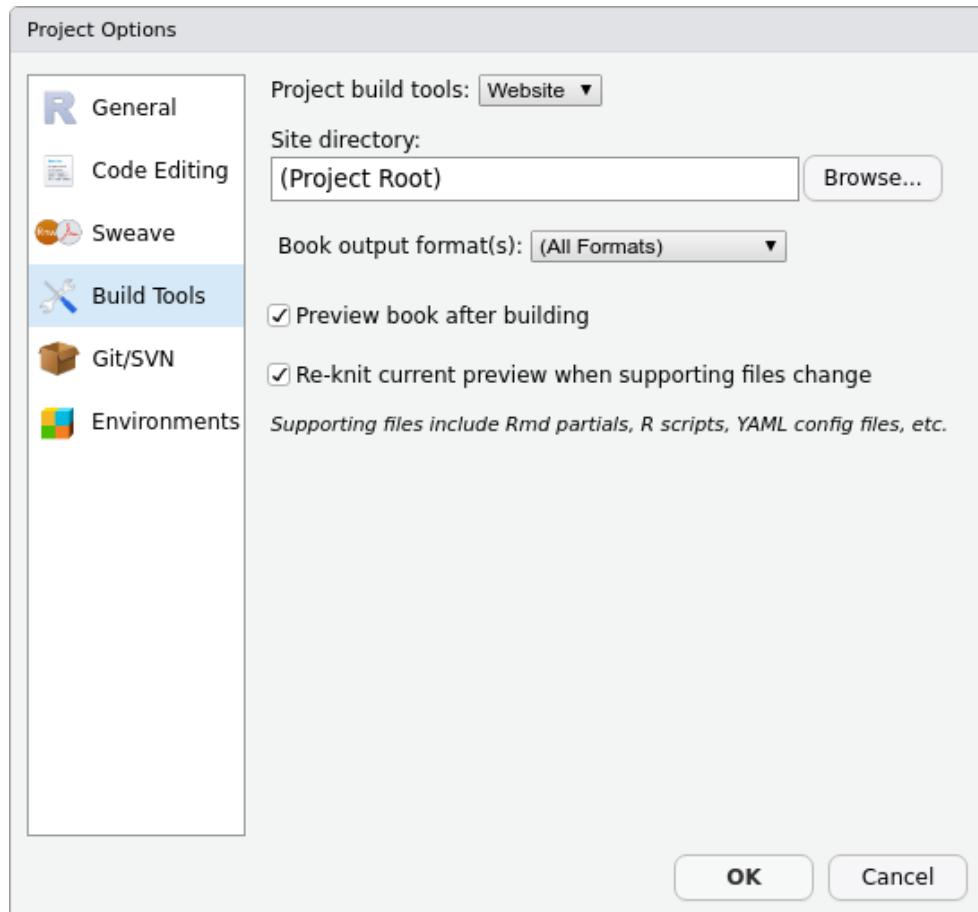


図2.1: Build ペーンの手動設定



図2.2: Build ペーンの手動設定

第 III 部

R Markdown と Bookdown の基 本機能

このパートの概要

ここではまず, R Markdown の基本的な機能を紹介する. つまり bookdown 特有のものではなく, R Markdown 全般で使用できる機能も含めて紹介する. これ以降は自己言及的な説明が多いため, この文書を生成しているソースコードと比較しながら確認することをおすすめする. ここで紹介する機能は BKD, RDG, RCB での記述に基づく. これら 3 つのドキュメントを読めば, ほとんどのことは可能になる — rmdja を作る理由になった LaTeX テンプレートの修正以外は — のだが, 本稿の重要な目的の 1 つは複数のファイル形式をなるべく簡単に両立することであるので, それができない書き方には触れないし, 技術文書の作成にあまり使わないような機能の動作確認はおこなわず, 技術文書作成で頻繁に使われ, 便利と思える機能のみ紹介する.

どちらにしろそのうちこれらを翻訳してくれる人が現れることだろう...たぶん.

第3章

静的なコンテンツの作成

まずは、単なるマークアップ、つまりプログラミングの複雑な処理を考えなくても良いタイプの、簡単な構文を紹介する。それらの多くは一般的な Markdown のものと同じである。

日本語で書かれた資料でごく基本的なことについて、『R Markdown 入門』で一通り紹介されている。やや応用的なことも『R Markdown ユーザーのための Pandoc's Markdown』に書かれている。

3.1 Markdown の基本構文

一応基本の Markdown の構文も挙げておく。詳細は (ref: BKDB) “Ch. 2.2 Markdown Syntax” を参照。

3.1.1 インラインでの書式変更

テキストの一部のみ書式を変える

アンダースコアで強調 (イタリック)

`_underscore_`

underscore

`** 2 つで太字強調`

`** 太字強調 **`

太字強調

等幅フォント

`'bookdown' と 'rmdja'`

`bookdown` と `rmdja`

本文中に入力した URL は自動判別され、ハイパーリンクが付けられる。また、`[テキスト](URL)` という書式で、テキストに対してハイパーリンクを付けることができる。

URL は自動判別される：

↳ https://github.com/Gedevan-Aleksizde/my_latex_templates/tree/master/rmdja

`['rmdja']` の github リポジトリ

↳ [リ \]\(https://github.com/Gedevan-Aleksizde/my_latex_templates/tree/master/rmdja\)](https://github.com/Gedevan-Aleksizde/my_latex_templates/tree/master/rmdja)

URL は自動判別される：https://github.com/Gedevan-Aleksizde/my_latex_templates/tree/master/rmdja

`rmdja` の github リポジトリ

3.1.2 ブロック要素

以降は行内では使えず、適切に表示するには前後に改行を挟む必要のあるタイプの構文である。

まず、引用ブロックを使えばかっこいいエピグラフを書き放題である。

```
> Нужны новые формы. Новые формы нужны, а если
↳ их нет, то лучше ничего не нужно.
>
>新しいフォーマットが必要なんですよ。新しいフォーマットが、それがないというな
↳ ら、いっそ何もないほうがいい。
>
> `\\r tufte::quote_footer('--- A. チェーホフ『かもめ』')`
```

新しいフォーマットが必要なんですよ。新しいフォーマットが、それがないというなら、いっそ何もないほうがいい。

— A. チェーホフ『かもめ』

rmdja では、HTML と PDF 両方で同様のデザインの枠で表示するようにしている。

Markdown では # は見出しを意味するが、bookdown にはさらにオプションが用意されている。

見出し名 {-} で、セクション番号のつかない見出しを用意できる。序文、章末の参考文献、付録のセクションに使えるだろう。さらに、bookdown では # (PART) 見出し名で「部」の見出しを作ることができる。この見出しあはセクションの合間に挟まるが、選択することはできない。文書が長くなったときに、より大きな区切りを付けるのに役に立つだろう。さらに、# (APPENDIX) 見出し名 {-} で、以降の見出しの頭に「補遺 A, B, C, ...」と付番できる。

箇条書きは以下のように書ける。

```
* iris setosa  
* iris versicolor  
* iris virginica
```

- iris setosa
- iris versicolor
- iris virginica

```
1. iris setosa  
2. iris versicolor  
3. iris virginica
```

1. iris setosa
2. iris versicolor
3. iris virginica

インデントを使えばネストできる。

- 課長
 - 課長補佐
 - * 課長補佐代理
 - 課長補佐代理心得

3.2 Markdown を使った図表の挿入

markdown は表を記入することもできる。

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

表3.1: Markdown 記法の表

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

画像ファイルも貼り付けられる。

しかし、キャプションを付けたり、表示位置やサイズを細かく調整したり、注釈を付けたりするためには、後述するように **R プログラムを経由して出力** したほうが良い。

TODO: md 記法で画像貼り付けたときのサイズ統一

3.2.1 コメントアウト

HTML 式の `<!-- -->` でコメントアウトできる。コメントアウトされた箇所は生成ファイルでもコメントアウトされるのではなく、そもそも出力されなくなる。



図3.1: Johannes Gutenberg

3.3 数式

LaTeX 記法で数式を記述できる。HTML ならば Mathjax によってレンダリングされる。数式の記述ルールは少々ややこしい。これは現在の pandoc の仕様で HTML および LaTeX の規格で矛盾なく出力するためやむをえない措置である。

1. 改行をしない行内数式は \$ で囲む、または \(), \() で囲む。
2. 改行を伴う数式ブロックは \$\$ で囲む、または \[, \] で囲む。
3. align, equation 環境等を使う場合は、上記の記号を使わず、直接 LaTeX コマンド \begin{align}... を打ち込む。

```
\@ref(eq:binom) は二項分布の確率関数である
\begin{align}
f(k) &= {n \choose k} p^k (1-p)^{n-k} (\#eq:binom)
\end{align}
```

その出力は、以下のようになる。

(3.1) は二項分布の確率関数である

$$f(k) = \binom{n}{k} p^k (1-p)^{n-k} \quad (3.1)$$

Bookdown では従来の R Markdown できなかった数式への付番と、本文中の参照アンカーリンクの自動生成が可能となっている（詳細は 5.1 章で）。LaTeX にすでに慣れている読者に注意が必要だが、Bookdown 特有の制約として、付番したい場合は `\label{ID}` ではなく `(\#eq:ID)` を使う。また、PDF (LaTeX) と HTML (Mathjax) の仕様には

1. PDF では `align` は常に数式が付番され、`align*` 等はどうやっても付番されない
2. HTML では `align` でも `align*` であってもラベルを書かなければ付番されず、書けば付番される。

という違いがある。両者で同じ表示にこだわるのなら、付番を取り消す `\notag` を多用することになるだろう。

さらに、bookdown の機能として、LaTeX の「定理」「定義」「証明」などの環境に対応するものが提供されている（参考：BKD Ch. 2.2 Markdown extensions by bookdown）。これらの相互参照も可能である。

例：以下に補題 3.1、定理 3.1 を示す。

補題 3.1 (ボレル-カントリの補題). E_1, E_2, \dots をある確率空間の事象とする。これらの事象の確率の和が有限であるなら、それらが無限に多く起こる確率はゼロである。つまり、

$$\sum_{n=1}^{\infty} P(X_n) < \infty \Rightarrow P\left(\lim_{n \rightarrow \infty} \sup X_n\right) = 0,$$

$$\lim_{n \rightarrow \infty} \sup X_n = \bigcap_{n=1}^{\infty} \bigcup_{k \leq n}^{\infty} E_k$$

である。

Proof. 証明は読者の課題とする. □

定理 3.1 (無限の猿定理). 猿がほとんど確実にタイプライタの全てのキーを無限回叩くならば、ほとんど確実にテキストには任意の作品が含まれる。

Proof. 補題 3.1 より自明. □

3.4 カスタムブロック

数式のセクションの定理ブロックの応用で、独自のブロックセクションを定義することができる。rmdja では BKD Ch. 2.7 Custom blocks で紹介されている例を予め使えるようにしている。それらは `type="..."` で指定できて、以下の 5 種類がある。

- `caution`
- `important`

- memo
- tip
- warning

である。

このブロック内では Markdown の基本構文しか使えず、引用や相互参照などは使えない。これらをブロック内で使いたい場合は block の代わりに block2 と書く。ただしこちらは pandoc の機能のハックであるため、将来使えなくなる可能性もある。

やや煩雑になるが、Pandoc の fenced Div を利用した書き方は、より安全である。

```
:::{.infobox .important data-latex="{warning}"}
fenced Div によるブロック
:::
```



fenced Div によるブロック

この構文の意味の解説は『R Markdown クックブック』などを参考に。

3.5 脚注

脚注はインラインと、巻末に書く 2 通りがある。

ここにインラインで脚注 ^[脚注の本文]

ここにインラインで脚注^{*1}

本文は巻末に書く [^example-1][^example-2]。

[^example-1]: 脚注の本文その 2

[^example-2]: 脚注の本文その 2

本文は巻末に書く ^{*2 *3}。

ここにインラインで脚注 [^ 脚注の本文]

*1 脚注の本文

*2 脚注の本文その 2

*3 脚注の本文その 2

インラインで書くほうがシンプルに見えるが、この記法では間を空けずに連続して脚注を書くことができない。

このように書くと^[脚注その 1]^[脚注その 2]上付きとして認識される

3.6 改行・改頁（改ページ）・改丁

HTML や Tex をそのまま書く場合と違い、Markdown での改行はそのまま反映されるため、通常は `
` や `\\"`などを書く必要はない。HTML の場合、紙への印刷を想定しないため任意のタイミングで改ページするという考え方ではない。Web ページの分割は章やセクション単位でなされる。PDF の場合は `\newpage`, `\clearpage`, `\cleardoublepage` という 3 種類の命令文が用意されている。

1. `\newpage` はページを改めるが、段組の場合は次のページではなく次の段に飛ぶ。
2. `\clearpage` は段組みでもページを改める。またそれ以前に配置した図表のフロートの配置を確定させる。つまり図表をフロートにしても `\clearpage` をまたいで位置が変わることはない。
3. `\cleardoublepage` は次の奇数ページまで飛ぶ（いわゆる改丁）。書籍では通例、章の始めなどを奇数ページに揃える。なお # で章見出しを書いた場合は自動でこの命令文が適用されるため手作業でこの命令文を書く必要はない。

第4章

動的なコンテンツの作成

4.1 プログラムチャンク

プログラムチャンクは、R Markdown 最大の特徴であり、R のソースコードや、その実行結果を Markdown に挿入できる。さらには **R 以外の言語の動作も可能**である。順番が前後してしまったが、定理などのカスタムブロックは本来はプログラムを入力するためのチャンクブロックであり、それを静的なテキストコンテンツの挿入に流用しているだけである。

以降は R で多くのユーザが頻繁に使うパッケージと、いくつかの技術文書作成に役に立つパッケージをインポートしている前提の説明とする。なお、rmarkdown, bookdown はチャンク内で特に読み込む必要がない。

```

01 pkgs <- installed.packages()
02 for (p in c("tidyverse", "ggthemes", "equatiomatic", "tufte", "kableExtra")) {
03   if (!p %in% pkgs) install.packages(p)
04 }
05 if (!"rmarkdown" %in% pkgs) remotes::install_github("rstudio/rmarkdown")
06 if (!"bookdown" %in% pkgs) remotes::install_github("rstudio/bookdown")
07 require(tidyverse)
```

要求されたパッケージ tidyverse をロード中です

```
-- Attaching packages ----- tidyverse 1.3.1 --
v ggplot2 3.3.3     v purrr    0.3.4
v tibble   3.1.2     v dplyr    1.0.6
v tidyverse 1.1.3     v stringr  1.4.0
v readr    1.4.0     vforcats  0.5.1

-- Conflicts ----- tidyverse_conflicts() --

```

```
x dplyr::filter()      masks stats::filter()
x dplyr::group_rows() masks kableExtra::group_rows()
x dplyr::lag()        masks stats::lag()
```

```
01 require(ggthemes)
```

要求されたパッケージ `ggthemes` をロード中です

```
01 require(equatiomatic)
```

要求されたパッケージ `equatiomatic` をロード中です

```
01 require(kableExtra)
```

このように、ログを掲載することもできる。これは再現性を重視する際に重宝するが、一方で単に画像などの出力だけを掲載したい場合もあるだろう。あるいは、プログラムを解説するためにプログラムは掲載するが実行しない、ということも必要になるかもしれない。プログラムと結果の表示/非表示はどちらも簡単に切り替え可能である。そのためには、チャンクオプションを指定する。

- `echo`: プログラムを掲載するかどうか
- `message`: プログラム実行結果の標準出力を掲載するかどうか
- `warning`: プログラム実行結果の警告を掲載するかどうか
- `error`: プログラム実行結果のエラーを掲載するかどうか
- `eval`: 文書作成時にプログラムを実行するかどうか
- `include`: 文書作成時にプログラムを実行し、かつ掲載しないかどうか
- `results`: 出力をいつもの R の出力風にするか (`markup`)、隠すか (`"hide"`)、出力を区切らずまとめるか (`"hold"`)、テキストをそのまま出力するか (`"asis"`)。最後は R Markdown のソースコードを動的に生成したい場合などに使う。

チャンクごとに個別に設定することも、デフォルト値を一括設定することもできる。前者の場合、チャンクオプションは {} 内部にカンマ、で区切って書く。`r` は R で実行するという意味である。チャンクの一般的な記法は以下のようになる。

```
```{r [<label>], [<options>]}
data(cars)
summary(cars)
```

```
```
```

r の直後の <label> はラベルと呼ばれ、チャンクの ID としての機能を持つ（省略された場合は自動で適当な名前がつけられる）。ラベルは主に後述の図表の相互参照に使われる。ラベルは英数字とハイフンを使って重複しない範囲で自由に命名できる。

一括設定の場合、以下のようなプログラムでデフォルト値を上書きできる。

```
01 knitr::opts_chunk$set(  
02   echo = F,  
03   message = T,  
04   warnings = F,  
05   error = F  
06 )
```

なおこのチャンクは eval=F を設定することで、実行されることなくプログラムのみ掲載している。ただし、プログラムのみを掲載するなら、以下のように Markdown の機能でも可能である。こちらの記法は {} がなくなっていることに注意する。

```
```sh
```

```
echo Hello, Bookdown
```
```

{ } ブロック内の値にはさらに R プログラムで与えることができる。この使い方は後の一章で解説する。

これらのオプションがあるおかげでプログラムとその結果の再現を説明したい場合はソースコードも表示させたり、回帰分析やシミュレーションの結果だけを掲載したい時は結果のみ表示したりできる。これが R Markdown のチャンクの強みである。例えば Jupyter notebook/lab などは従来、コードセルと出力セルを自由に隠すことができなかった。

チャンクに使用できる言語は R だけではない。つまり Python なども使用できる（詳細は 12 章を参照）。以下で対応しているエンジンの一覧を表示できる。

```
01 names(knitr::knit_engines$get())
```

```
[1] "awk"      "bash"     "coffee"    "gawk"     "groovy"  
[6] "haskell"  "lein"     "mysql"     "node"     "octave"  
[11] "perl"     "psql"     "Rscript"   "ruby"     "sas"  
[16] "scala"    "sed"      "sh"       "stata"    "zsh"  
[21] "highlight" "Rcpp"     "tikz"     "dot"      "c"
```

```
[26] "cc"          "fortran"      "fortran95"    "asy"        "cat"
[31] "asis"         "stan"         "block"        "block2"      "js"
[36] "css"          "sql"          "go"          "python"     "julia"
[41] "sass"         "scss"         "R"           "bslib"      "theorem"
[46] "lemma"        "corollary"    "proposition" "conjecture" "definition"
[51] "example"      "exercise"    "hypothesis"  "proof"      "remark"
[56] "solution"
```

また、新たにプログラムを追加することもできる。詳細は RDG Ch. 2.7 Other language engines を参考に。

TODO: 他の言語のプログラムを実行する際の注意点

4.2 プログラムで数式を生成する

プログラムチャンクは、単にプログラムの計算結果を埋め込むだけでなく、静的なコンテンツを臨機応変に変更して出力させたり、あるいは手作業でやるには煩雑な加工処理を挟んでから表示させるのに役に立つ。

R のプログラムと組み合わせることで回帰分析の結果の数値をコピペすることなく数式で表示することができる。そのためには `equatiomatic` パッケージの `extract_eq()` を使う。

まずは、回帰係数を記号で表現するタイプ。LaTeX 数式をそのまま出力するため、チャンクオプションに `results="asis"` を付ける必要があることに注意する。

```
01 data(mtcars)
02 fit <- lm(mpg ~ ., data = mtcars)
03 extract_eq(fit, wrap = T, ital_vars = T, align_env = "aligned")
```

$$\begin{aligned} mpg = \alpha + \beta_1(cyl) + \beta_2(disp) + \beta_3(hp) + \\ \beta_4(drat) + \beta_5(wt) + \beta_6(qsec) + \beta_7(vs) + \\ \beta_8(am) + \beta_9(gear) + \beta_{10}(carb) + \epsilon \end{aligned}$$

さらに `use_coef = T` で係数を推定結果の数値に置き換えた。

```
01 extract_eq(fit, wrap = T, ital_vars = T, use_coef = T, align_env = "aligned")
```

$$\begin{aligned} \widehat{mpg} = 12.3 - 0.11(cyl) + 0.01(disp) - 0.02(hp) + \\ 0.79(drat) - 3.72(wt) + 0.82(qsec) + 0.32(vs) + \\ 2.52(am) + 0.66(gear) - 0.2(carb) \end{aligned}$$

`equatiomatic` パッケージは現時点では `lm` `glm` に対応しており、`lmer` への対応も進めているようだ。

TODO: この書き方だと PDF で付番できない

4.3 プログラムを使った図の挿入

既に Markdown 記法による図表の挿入方法を紹介したが³、プログラムチャンクを介して画像を読み込み表示させることもできる。まずは、R のプログラムで既存の画像ファイルを表示させる方法。

```
01 knitr::include_graphics(file.path(img_dir, "Johannes_Gutenberg.jpg"))
```



図4.1: Johannes Gutenberg

もちろんのこと既存の画像だけでなく、データを読み込んでヒストグラムや散布図などを描いた結果を画像として掲載することもできる。

技術文書や学術論文では、画像の上か下に「図 1: XXXXX」のようなキャプションを付けることが多い。紙の書籍では絵本のように本文と図の順序を厳密に守るより、余白を作らないよう図の掲載位置を調整する必要があるからだ。

プログラムチャンクにはこのキャプションを入力するオプション `fig.cap` があるため、`plot()` 側でタイトルを付けないほうが良い。例えば `ggplot2` パッケージの関数を使い以下のようなチャンクを書く^{*1}。

^{*1} なお、R ユーザーならば標準グラフィック関数である `plot()` 関数をご存知だろうが、本稿では基本的により便利な `ggplot2` パッケージを使用してグラフを作成している。

```
```{r plot-sample, echo=T, fig.cap="`ggplot2` によるグラフ"}
data("diamonds")
diamonds <- diamonds[sample(1:NROW(diamonds), size =),]
ggplot(diamonds, aes(x=carat, y=price, color=clarity)) +
 geom_point() +
 labs(x = "カラット数", y = "価格") + scale_color_pander(name = "クラリティ") +
 theme_classic(base_family = "Noto Sans CJK JP") + theme(legend.position = "bottom")
```

```

実際の表示は図 4.2 のようになる。

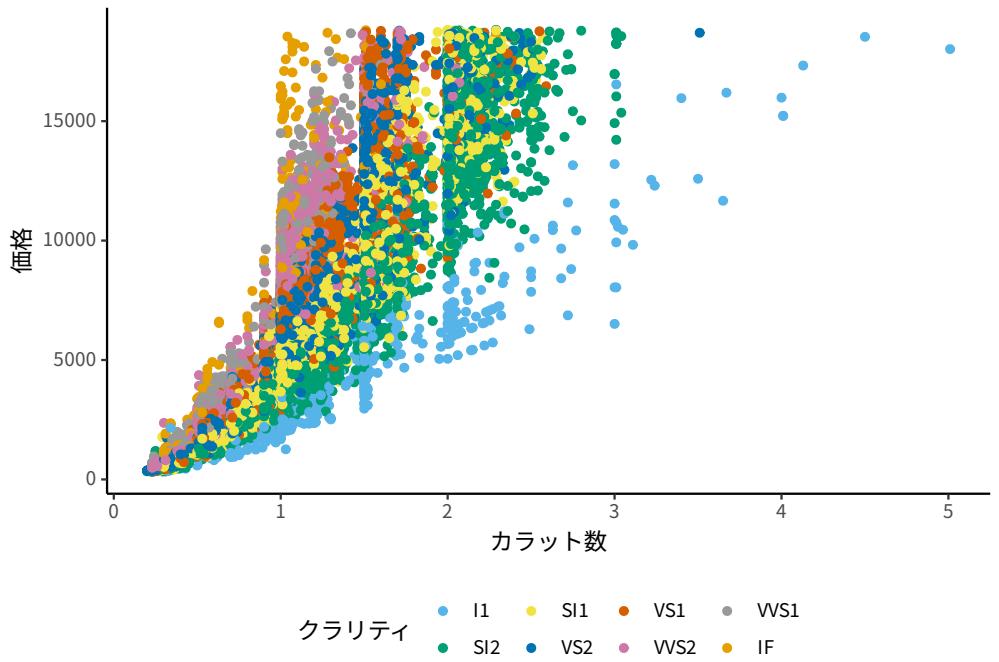


図4.2: ‘ggplot2’ によるグラフ

ggplot2 以外のパッケージや言語、たとえば tikz や asymptote, DOT 言語も使用できる。これらは 8 章で紹介する。

4.4 (WIP): デフォルトフォントの設定

Windows や Mac では、デフォルトのフォントが日本語グリフを持たないのでグラフが文字化けする。現時点では最低限 `rmdja::set_graphics_font()` という関数を呼び出す処理を手動で書き加えなければならない。本文のフォントと異なり、現時点 (ver. 0.4.2) では手動設定が必要になる。OS ごとのフォント名を調べて指定するのが大変なら、私が作成した `fontregisterer` パッケージを使うのも 1 つの手である。その解説は『[おまえはもう R のグラフの日本語表示に悩まない \(各 OS 対応\)](#)』に書いた通りである。`get_standard_font()` で使用中の OS で標準インストールされているセリフ (明朝), サンセリフ (ゴシック) のフォントファミリ名を 1 つづつ取得するので、その値のどちらかを

`rmdja::set_graphics_font()` に与えれば, `ggplot2` および標準グラフィックスのデフォルトのフォントが日本語対応フォントになる.

しかしこの関数は `ggplot2` のデフォルトのテーマを更新するだけなので `ggthemes` パッケージなどが用意するテーマプリセットを使用したい場合はその都度設定が必要である.

```
01 require(fontregisterer)
02 theme_set(ggthemes::theme_pander(base_family = get_standard_font()$serif))
03
04 ggplot(DATA, aes(...)) +
05   geom_point() +
06   ... +
07   theme_economist(base_family = get_standard_font()$sans)
```

4.5 TODO: 図のレイアウト設定

PDF ならばフロート設定のため, 図が離れた位置に配置されることがある. そのため, 「図 4.2」のような相互参照を使うと良いだろう. フロートを使うかどうかは, 後のセクションで解説する TODO

R のグラフィックデバイスを使っている限り, 通常の R のコンソールと同じコードをチャネル内に書くだけで表示できる.

R のグラフィックデバイスではないとは, RGL や `plotly` など外部ライブラリに頼ったグラフ作成ツールのことである. 判断できない人は, RStudio 上で実行して, “Plots” ペーンに表示されたら R のグラフィックデバイス, “Viewer” ペーンに表示されたらそうでない, で覚えていただきたい. 後者を表示する方法は 11 章で後述する. R をこれまで使ったことがなく, それすらも何を言っているのか分からない, という場合は `ggplot2` を使ってもらう.

最後の `fig.cap=""` がキャプションである. ただし, どうも日本語キャプションを書いたあとに他のチャネルオプションを指定するとエラーになるようだ. よって `fig.cap=` はオプションの末尾に書くべきである. また, `fig.cap=""` に数式や一部の特殊なテキストを直接入力することができない. この問題は相互参照について解説するセクション 5.1 で詳細を述べる.

`fig.cap` 以外のオプションはおそらく頻繁には変えないため, 冒頭でまとめて設定したほうが楽だろう.

```
01 knitr::opts_chunk$set(
02   fig.align = "center",
```

```

03     fig.width = 6.5,
04     fig.height = 4.5,
05     out.width = "100%",
06     out.height = "100%"
07 )

```

なお、これらは `rmdja` でのデフォルト値であるため、実際にこの値をあえて記述する必要はない。

ここで、`fig.width` と `out.width` の違いも述べておく。`out.width/out.height` は表示する画像サイズの違いで、`fig.width/fig.height` はプログラムが output した画像の保存サイズである。よって `ggplot2` などを使わず画像ファイルを貼り付けるだけの場合は `fig.*` は意味をなさない。

4.6 R プログラムを使った表の装飾

Markdown 記法を使った表記は既に紹介した。しかしこれは表の数値を全て手動で書かなければならぬ。R はテーブル状のデータ処理に長けているため、このような煩雑さを省くことができないか、とあなたは思っていないだろうか。もちろん R Markdown では R での作業中に使用しているデータをいちいち手書きなどせずとも表示できるし、テーブルのデザインもある程度自由に設定できる。

R Markdown のデフォルトでは R のコンソールと同様にテキストとして出力されるが、`rmdja` では異なるデザインで表示されている。これは `knitr`, `kableExtra` パッケージなどで事後処理をかけることで見やすいデザインの表に変換しているからである。R Markdown の基本ルールとして、チャック内で最後に呼び出したオブジェクトが表示される。例えば `mtcars` という R が用意する練習用データフレームを、チャック内で上から 10 行までを呼び出してみると、以下のように表示される。

```

01 data(mtcars)
02 mtcars[1:10, ]

```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4

```
Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
```

これは R のコンソール出力と同じで、プレーンテキストでの出力である。表として出力する最も簡単な方法は、フォーマット関数に `df_print` を指定することである。たとえば `df_print: kable` を指定すると、表 4.1 のようになる。

```
output: ...
df_print: kable
```

```
01 mtcars[1:10, ]
```

表4.1: `df_print: kable` の場合

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

このオプションは R Markdown の処理中にデータフレームの呼び出しを検出し、`df_print` のオプションに対応したスタイルを変換する関数を適用している。他のオプションとして、`tibble`、`paged` などがあるが現時点の `rmdja` では大差がないので詳細な説明を省略する（図 4.2）。

表4.2: `df_print` のオプション一覧

オプション	効果
<code>default</code>	<code>print()</code> 、コンソール出力と同じ
<code>tibble</code>	<code>tibble</code> 対応版 <code>print()</code>
<code>paged</code>	<code>rmarkdown::paged_table()</code> による表示、これもオプション引数を指定しなければ大差なし

表4.3: `booktabs = T` は PDF にのみ影響する

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

オプション	効果
<code>kable</code>	<code>knitr::kable()</code> による表スタイル

よって、これらの関数をチャンク内で呼び出すことで、手動で表のスタイルを指定することも可能である。表のスタイルにこだわりたい、相互参照やキャプションを付けたい、といった場合はこれらのうち `knitr::kable()` 関数を手動で使うのが 1 つの手である。実は、先ほどの `df_print` の例も、実際にはこの関数を呼び出して出力している。この場合、表のキャプションは `kable()` 内で指定できる（現時点では、図とは異なりチャンクオプションではキャプションを指定できない）。デフォルトでは `caption =` の文字列はそのまま出力されるため、太字強調など Markdown 記法も変換されずそのまま表示されてしまう。これには対処方法がいくつかある。

1. `rmdja` パッケージの提供する `knitr::kable()` または `kableExtra::kbl()` 関数のラッパを使用する（表 4.3）
2. `escape = F` および `format = "pandoc"` を指定する
3. (非推奨) HTML と PDF でそれぞれの構文で表を描く処理を自分で書く

```
01 rmdja::kable(mtcars[1:10, ], caption = "``booktabs = T`` は PDF にのみ影響する",
  ↪ booktabs = T)
```

(1) の方法が現在最も簡単である。ただし、`LATEX` の構文が評価されなくなるため同時に使うことはできない。例えば太字強調と数式を両方表示したい場合は、`knitr::is_latex_output()` PDF の場合は完全に `LATEX` で、HTML の場合は Markdown で書く、という場合分けを自分で書いて `knitr::kable()` に与えなければ

表4.4: 数式 a と 太字

	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

ばならない(表 4.4). また, キャプションではなく表内の markdown 構文も評価されない. 表内の markdown 構文を PDF でも反映するには, (2) の方法が必要である.

TODO: この仕様は使いづらいのでそのうちなんとかしたい.

(2) についても, `kable()` 単体であれば問題ないが, 後に紹介する `kableExtra` パッケージを併用すると書式設定がうまく反映されなくなることがある. (3) は表 4.4 の記述をキャプションだけでなく, HTML ならば Markdown または HTML タグで, PDF ならば LATEX で表全体を書き分ける, という方法である. 1 つの表を描くのに多大な労力がかかるため推奨しない.

```

01 cap <- if (knitr:::is_latex_output()) "数式 $a$ と \\textbf{太字}" else "数式 $a$  

02   & ** 太字 **"  

03  

04 kable(  

05   head(mtcars),  

06   caption = cap,  

07   booktabs = T  

08 )

```

さらに, デフォルトでは `kable()` が PDF に出力する表のデザインはあまりよろしくないが, `kable()` 関数は過剰な罫線のない表の出力も簡単である. LATEX を使ったことのある人は知っているかもしれないが, これは `booktabs.sty` を使った表のスタイルになっている^{*2}.

また, `kable()` を使う利点として, 表の絡む名に好きな名前を与えられるというものがある. データフレームの列(変数)名は, 括弧などプログラミングで特別な意味を持つ文字を

^{*2} もし何らかの理由でこのスタイルにならない, あるいはあえてしたくない, と言う場合は `kable()` 関数で `booktabs = T` を指定せよ.

使うことができない。そこで, `kable()` の `col.names` 引数に表のカラム名を改めて与えることで、こういった文字も出力できる。

`kable()` による表のスタイルは `kableExtra` パッケージを使うことで様々にカスタマイズできる。例えば HTML 版ではデフォルトで奇数偶数行の背景色が異なるが、PDF ではそうなっていない。また、図表の位置は常にフロートであり、余白ができにくくように表示位置が前後する（これは技術文書や学術論文では普通のことだが）。さらに、表が本文の領域からはみ出しており見栄えが悪い。これらの設定を HTML 版に近づけたい場合は `kableExtra::kable_styling()` を使って簡単にデザインを変えることができる（表 4.5）。以下のように、`full_width` は表の幅を本文幅にそろえるオプションである。や十分に幅の小さい表に対しては逆に間延びして見づらいためデフォルトでは無効となっているが、このようにして表幅を調整するのに使える。さらに `latex_options` は PDF にのみ有効なオプションである。`"striped"` が奇数偶数の色分け^{*3}, `"hold_position"` が表示位置を「なるべく」固定するオプションである（それでも表示位置が大きくずれて気に入らない場合 `"HOLD_position"` を代わりに使うとよい）。ただし HTML と違い PDF では改ページがあるためこのオプションを多様すると、以下のように本文に無駄な余白が増えることに注意する。

```

01 rmdja::kable(
02   mtcars[1:10, ],
03   booktabs = T,
04   caption = "奇数行を強調し、PDF では `booktabs` を利用"
05 ) %>%
06   kable_styling(
07     full_width = if (knitr:::is_latex_output()) T else NULL,
08     latex_options = c("striped", "hold_position")
09   )

```

このように、R Markdown ではまず表示したい表と同じ構造のデータフレームを作ることで、簡単にスタイルの調整された表を掲載できる。

他にもいくつか表のスタイルをカスタマイズするためのパッケージが存在する。より発展的な表のスタイル指定方法については 9 章で話す。

^{*3} ただし、`full_width = T` を指定した時、`striped`、あるいは他の色の指定の命令が反映されないことがある。これは 2019 年時点での `tabu.sty` の不具合であるため、Issues #1 で配布されている開発者によるパッチを適用しなければならない。また、それ以外にも表の幅を調整する方法がある。詳細は 9 章を参考に。

表4.5: 奇数行を強調し, PDF では booktabs を利用

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

第5章

相互参照と引用

5.1 相互参照

5.1.1 図表や式へのアンカーリンク

図, 表, 式などに番号を自動で割り当て, さらにハイパーリンクを付加できる. `\@ref(ID)` を使う. 現状では `refstyle` や `prettyref` のように接頭語を自動で付けてくれないが, そのうちなんとかなるかもしれない.

bookdown の相互参照は, LaTeX の `prettyref.sty` のように, 接頭語: 参照 ID という記法になる. 参照 ID は通常, チャンク ID と同じである. 既に紹介したように, 数式参照の接頭語は `eq` で, 定理は `thm` である. 図表は `fig`, `tab`. その他の接頭語は BKD Ch. 2.2 Markdown extensions by bookdown を参考に.

5.1.2 表への相互参照

Markdown 記法で表を書く場合, 以下のように `Table:` の直後にラベルを記入する (表 5.1).

Table: (\#tab:tab-md) Markdown 記法の表

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

表5.1: Markdown 記法の表

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

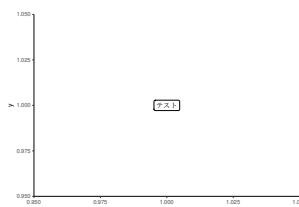
5.1.3 章への相互参照

章見出しへの相互参照も可能である。これは Pandoc の機能を利用しているため、接頭辞は不要である。Pandoc の仕様により欧文であればタイトルがそのまま参照 ID となるが、非欧文の文字に対して適用されないため、基本的に日本語文書の場合は参照したい章の見出しの後にスペースを入れて {# 参照 ID} と書く必要がある。そして本文中で参照する場合 \@ref(参照 ID) と表記する。

5.1.4 特殊な相互参照

チャックオプションの `fig.cap` などに TeX 数式を書いても正しく表示できない。そのような場合は `ref` 参照を使う。`(ref:figcap1) \coloremoji{[]} $ \sum \oint \mathfrak{A} \mathfrak{B} \mathfrak{C} \mathfrak{D}` と書くと、図 5.1 のキャプションにも特殊な記号が使える。

なお、複数指定する場合は連続させず、改行で 1 行空けて宣言する必要がある。

図5.1: $\sum \oint \mathfrak{A} \mathfrak{B} \mathfrak{C} \mathfrak{D}$

この参照は一度しか使えない。

PDF での表示では、図 5.1 のキャプションの外側が文字化けしていることだろう。これは 絵文字出力に関する問題で、別のセクションで解説する。

これはかなり強力で、

1. 定義される前の行にも適用される

2. チャンクオプションだけでなく出力結果にも適用される

という仕様である。

TODO: 自己言及的な文章は書かないならこれくらいの認識でいいだろうが、より正確な話はどうするか

5.2 文献引用

YAML フロントマターの `bibliography:` に文献管理ファイル (.bib, .json 等) を指定することで、ファイルに含まれる文献への参照が可能になる。@ 引用 ID で本文に引用を与えられ、文書に引用した文献の一覧が自動で生成される。また、`citr` パッケージにより、RStudio Addins に文献に対応する引用 ID を取り出して挿入する機能が追加される。

```
01 knitr::include_graphics(file.path(img_dir, "citr.png"))
```

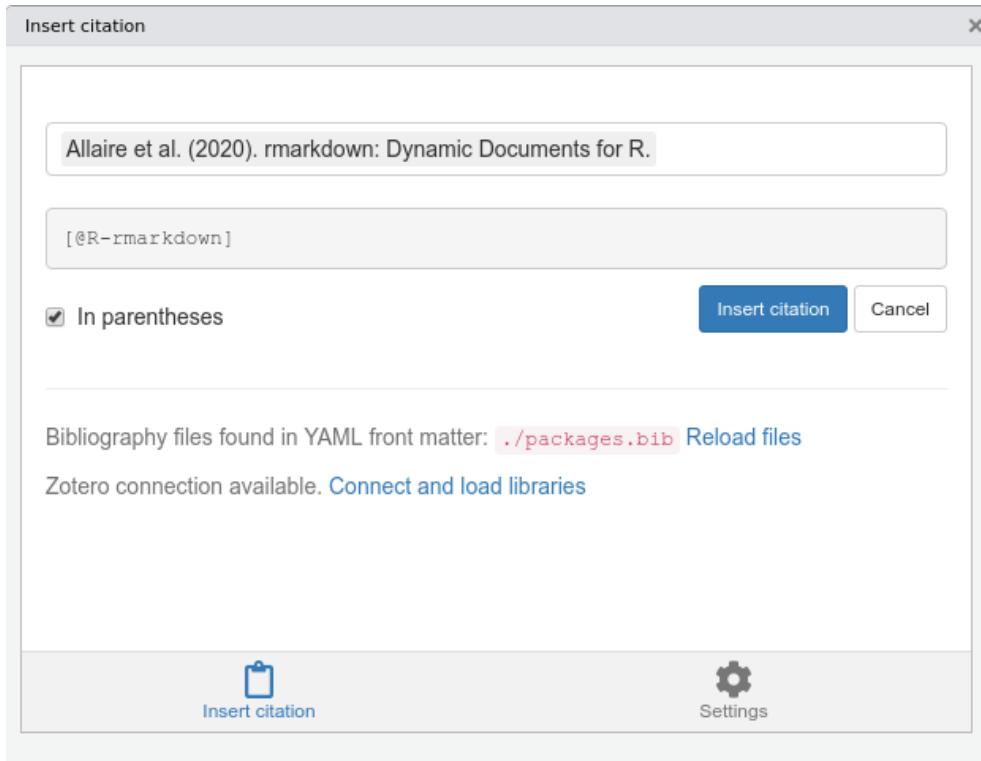


図5.2: `citr` パッケージの例

一方で、この記述が文書においてどのようなスタイルで出力されるかは文献引用を処理するプログラムによって変化する。そのプログラムには 3 つの候補がある。R Markdown の文献引用は pandoc を経由して処理され、現時点では `pandoc-citeproc` (default), `BIBTEX(natbib)`, `BibLaTeX (biblatex)` の選択をサポートしている。`pandoc-citeproc` 以

外はもともと L^AT_EX 用に作られたため、HTML では常に pandoc-citeproc で処理される。PDF ではそれに加えて bibtex, biblatex を指定することもできる。(default とは別なのでややこしいが) rmdja はデフォルトでは PDF 出力時に biblatex を使用する。これはフォーマット引数の citation_package で変更できる。正確には以下の 3 つの値のどれかを指定する。

- default: pandoc-citeproc を使用する。
- biblatex: BibLaTeX を使用する。デフォルト。スタイルのデフォルトはこちらが用意した jauthoryear というもの。
- natbib: Bi^ET_EX を使用し、本文中の参照には natbib.sty が使われる [natbib-constraint]。ただし、日本語（マルチバイト文字）の含まれる文献情報を出力する場合は特殊な設定をしないと製本処理がハングアップする（後述）。

5.2.1 文献引用スタイルのカスタマイズ

rmdja では、本文中の引用トークンのデフォルト設定を、文書タイプでは「著者-年」形式に、スライドでは番号形式にしている。このカスタマイズについて簡単な解説をする。従来の R Markdown ではカスタマイズに以下のような YAML フロントマター項目を使っていた。

- biblio-style: PDF 用スタイルファイル
- natbiboptions/biblatexoptions: それぞれ natbib または biblatex を使う場合のスタイルに関するオプション
- csl: CSL 用スタイルファイル, biblio-title: 「参考文献」タイトルの文字列

このうち biblio-style, natbiboptions, biblatexoptions はフォーマット関数で指定する。例えば以下のように。

```
output:
  rmdja::pdf_book_ja:
    citation_package: biblatex
    citation_options:
      - style=jauthoryear
      - natbib=true
```

これは pandoc の記法を利用した従来の R Markdown で以下のように書いてているのと同様であり、citation_package: natbib ならば biblatexoptions が natbiboptions に置き換わる。

表5.2: 引用プログラムごとの違い

item	HTML	PDF	日本語	指定名	文献ファイル	文献スタイル
pandoc-citeproc	TRUE	TRUE	TRUE	default	.json	.csl
biblatex	FALSE	TRUE	TRUE	biblatex	.bib	.bbx/.cbx
bibtex	FALSE	TRUE	FALSE	natbib	.bib/.bibtex	.bst

```
output:
....:
citation_package: natbib
biblio-style: jauthoryear
biblatexoptions:
- natbib=true
```

2通りの記法が存在するのはやや混乱するかもしれないが、後方互換性を考慮し rmdja ではこれらの2通りの記法どちらでも受け付けるようにしている。

biblatex 以外のエンジンで出力したい、例えば指定された .bst のスタイルで文献一覧を出力したい場合は、(u)pBIBTEX が必要になる。その操作の詳細は 10 章を参照。

5.2.2 文献リスト生成エンジンの違いについて

pandoc-citeproc, bibtex, biblatex はそれぞれ引用文献リストのスタイルを記述するファイルがあり、それぞれ拡張子は .csl, .bst, .bbx/.cbx, である。.csl は MS Word のスタイルと同じもので、XML で記述されている^{*1}。.bst は BIBTEX 用のフォーマットで、自分で改造するには逆ポーランド記法の構文に慣れねばならない。そして BibLaTeX はスタイルを LATEX のマクロで記述でき、さらにそういった細かい記述のスタイルファイルを用意しなくとも指定できるオプションがいくつか存在する（ここまで、表 5.2）。

現バージョンでは biblatex がデフォルトである。現在の日本語圏の LATEX 使用者にとっては .bst ファイルの種類が充実しているため natbib を使いたいところだが、R Markdown の場合エンジンが BIBTEX であるため日本語が使えない。(u)pBIBTEX を使うにはやや複雑な手順が必要である。よって、デフォルトでそのような下準備をさせるべきでないと考えたので rmdja では biblatex をデフォルトとし、日本語表示に最低限のスタイルだけを用意している。

^{*1} 簡単なカスタマイズなら CSL editor という Web サービスができる。しかしあくまで XML なので、あまり複雑な処理はできないことに注意する。

第 6 章

(WIP) 簡単なレイアウト・スタイル変更

前章では文書の内容に関する構文を紹介した。ここでは、文書全体のデザインやスタイルの設定方法のうち, rmdja が用意している簡単なものを紹介する。

6.1 フォント変更

フォント変更の方法は HTML と PDF で大きく違う。まずは HTML について。`_output.yml` にはデフォルトのフォントを設定できるが、選択肢は `sans` と `serif` しかない。

```
output: rmdja::gitbook_ja:
  config:
    fontsettings:
      family: serif
```

しかし HTML は文字通り HTML で出力しているため、CSS の使い方次第でいくらでもデザインを変えることができる。例えば自作した CSS ファイルを以下のように指定できる。

```
output:
  output: rmdja::gitbook_ja:
    css: ABC.css
```

PDF を生成する場合、ver 0.3 以降ではデフォルトのフォントファミリを OS に応じて変えている。もし変更したい場合は YAML フロントマターの以下の項目を変更する

- `mainfont: 欧文セリフフォントファミリ`

- `sansfont`: 欧文サンセリフフォントファミリ
- `monofont`: 等幅フォントファミリ (コードの表示などに使用)
- `jfontpreset`: 和文フォントファミリのプリセット
- `jmainfont`: 和文メインフォントファミリ (一般に明朝体を指定)
- `jsansfont`: 和文セリフフォントファミリ (一般にゴシック体を指定)
- `jmonofont`: 和文等幅フォントファミリ (コードの表示などに使用)

`jfontpreset` は `zxjafont` または `luatex-ja` によるプリセットで、3種類の和文フォントを一括指定できる。個別指定したフォントはこれを上書きする。特にこだわりがないなら一括指定で良いが、ソースコードを多く掲載する場合は `M+` や `Ricty` などのフォントを用意すると良いだろう。`rmdja` ではデフォルトで3種類の和文フォントファミリに対して、OSごとの標準日本語フォントが選択される(図 6.1)。いずれも各 OS で標準でインストールされているはずであるが、現時点ではフォントが実際にインストールされているか確認する機能はない。

表6.1: デフォルトで使用される日本語フォントファミリ

	Mac	Linux	Windows (8 以降)	Windows (それ以前)
X _E LATEX	游書体	Noto	游書体	MS フォント
LuaLATEX	ヒラギノ ProN	Noto	游書体	MS フォント

それ以外で使用可能な主なプリセット名は表 6.2 の通り。これらは X_ELATEX, LuaLATEX でそれぞれ `zxjafont.sty`, `luatex-ja.sty` を利用してフォントが埋め込まれる。両者の多くではプリセット名が共通しているが、一部例外もあることに注意(特に X_ELATEX は `luatex-ja` との互換性を考慮してエイリアスをいくつも用意している)。また、より詳細な一覧やオプションの全貌については、八登崇之氏の『PXchfon パッケージ』および `zxjafont` のマニュアルと、『luatex-ja の使い方』を確認してほしい。

さらに、それぞれの項目に対してオプションを設定する場合、`options` と接尾辞のついた項目が用意されている。欧文と和文フォントで全く異なるタイプのフォントを使ったために相対的なサイズが合わず不格好な場合は

```
mainfont: Palatinno
mainfontoptions:
  - Scale=0.9
```

などと書いて調整できる。

表6.2: 主な指定可能なフォントプリセット名

フォント	X _L A _T E _X	Lu _L A _T E _X	備考
MS ゴシック/明朝	ms	ms	X _L A _T E _X のみ HG フォントと併用する ms-hg などのバリエーションあり
游書体	yu-win10	yu-win10	Windows 8 以前は yu-win, Mac では yu-osx
ヒラギノ系	hiragino-pro	hiragino-pro	hiragino-pron で ProN/StdN 版を指定
Noto フォント	noto/noto-jp	noto-otf/noto-otc	
源ノ角ゴ/明朝	sourcechan-jp	sourcechan-jp	
原ノ味フォント	haranoaji	haranoaji	
梅フォント	ume	ume	
小塚フォント	kozuka-pro	kozuka-pro	-pr6 で ProVI 版, -pr6n で Pro6N 版を指定なども指定可能
IPA (Ex) フォント	ipa/ipaex	ipa/ipaex	X _L A _T E _X のみ ipa-hg などのバリエーションあり

6.1.1 どのフォントを使用すべきか

既に書いたように rmdja では, Windows なら游書体シリーズや BIZ UD フォントシリーズ, Mac ならヒラギノや游書体といったように OS 標準のフォントが自動で選択される. よく, LaTeX やグラフで日本語が表示できないときは「IPA フォント」をダウンロードして使え, という記述がネット上に出回っているが, **これは正確ではない**とわかる.

文字化けは多くの場合, 文字コードの指定ミスか日本語のグリフ(文字)が存在しない欧文フォントを使用しているのが原因であり, このようにしかるべき方法でフォントを指定すれば使い慣れた OS 標準のフォントが使用でき, BIZ UD フォントのような可読性に優れたフォントも使用可能である.

また, 無料公開されているフォントとしては, Noto フォントや原ノ味が存在する. 特に後者は見た目こそ既存の有名フォント(原ノ角や Noto)と大差ないが, CID を持つおり組版の点で効率的であり, Tex Live にも同梱されるようになった.

一方で, IPA フォントシリーズのうち, 「IPAmj 明朝」フォントは行政の戸籍管理システ

ム用途で開発されたため^{*1} 約6万字の漢字を収録するなど他の無料公開フォントと比べ格段に収録グリフ数が多いという特徴がある。さらに、CJK統合漢字のフルサポートをしているフォントとして、9万字近い漢字を収録した花園明朝シリーズがある。

言うまでもなくこれらは明朝体のみであるため単体では整った日本語文書に使用することが難しいが、異字体や稀にしか使われない（あるいは日本語では通常使われない）漢字の表記にこだわりたい場合は必要になるだろう^{*2}。

6.2 YAML フロントマターの設定によるスタイル変更

RMDファイルの冒頭に書かれている YAML フロントマターは様々なことが設定できる。これらのオプションは Pandoc のものに対応している。HTML はスタイルのほとんどが規格化された CSS で操作できるため、主に PDF に関するものである。また、本章はあくまで簡易的な機能解説であるため、より詳細な解説は後の章で再度取り上げる。

6.2.1 ハイパーリンクの配色変更

たとえば PDF では、以下のようにしてハイパーリンクの色を変更できる。

```
link-citations: true
linkcolor: blue
citecolor: blue
urlcolor: magenta
```

まず、ハイパーリンクを有効にする、link-citations:true が必要である。次に、リンクの種類ごとに色を指定できる。linkcolor は文書内のハイパーリンクの色、citecolor は巻末の参考文献リストへのリンクの色、そして urlcolor は文書外の URL へのリンクの色である。

また、ページのヘッダ・フッタは pagestyle で指定できる。

```
pagestyle: headings
```

文書クラスによって多少変わるが、プリセットが4種類用意されている。

1. empty: 何も表示しない
2. plain: ページ数のみ表示

^{*1} https://www.ipa.go.jp/about/press/20111026_2.html

^{*2} LaTeX での使い方は <https://texwiki.texjp.org/?TeX%E3%81%A8%E5%A4%96%E5%AD%97>などを参照

3. headings: ヘッダに罫線を引き、ページ数だけでなく章のタイトルも表示する
4. fancy: ユーザによるカスタマイズ (WIP)

(3) が一般的な書籍のものに近いスタイルである。(1-3) は、タイトルページや調整のための白紙ページなどはかならずしも変更されない。そういった根本的な変更がしたい場合のため、(4) が用意されている。しかし、現時点では `fancyhdr.sty` の使い方を知る必要がある。

6.2.2 L^AT_EX エンジンの変更

PDF の出力は `rrmdja::texlogo("LaTeX")` を使用する。その処理プログラムにはいくつかバリエーションがあり、`rmdja` では `\XeLaTeX` と `rmdja::texlogo("LuaLaTeX")` の使用を想定している (`R Markdown` では `\pdfLaTeX` もサポートしているが、これは日本語表示が難しいため `rmdja`⁴ では採用していない)。

両者は多少の違いがある（正確には、両者それぞれに対応した和文組版パッケージの違いにも由来する）。

1. 組版の違い。`XeLATEX` で和文組版を制御する `zxjatype.sty` にはいくつか改善の余地が残っている。
2. 速度の違い。年々改善されているようだが、それでも `LuaLATEX` は処理の遅さが目立つ^{*3}。
3. フォントレンダの違い。一概に言るのは難しいが、デフォルト設定では文字のウェイトや和文・欧文の相対的なバランスが微妙に異なる。
4. 和文・欧文を同時に扱う際の挙動の違い。`LuaLATEX` は和文と欧文の処理ルールが競合する時、和文の処理を優先する傾向がある。一方で `rrmdja::texlogo("XeLaTeX")`（および `zxjatype.sty`⁴）はなるべく両者を共存できるように作られている。

もし和文組版の厳格さを優先したいのなら `rrmdja::texlogo("LuaLaTeX")` を使うべきである。しかし処理速度があまりに遅いとか、欧文の扱いが気に入らないとかの場合は `rmdja::texlogo("XeLaTeX")` を使うと良い。特に書籍形式でなくプレゼンテーション資料であれば、組版ルールの厳格さはあまり気にならないことだろう。

6.2.3 文書クラスのカスタマイズ

デフォルトでは `bxjsbook` という文書クラスを使用している。この他 `rmdja` では `bxjsreport`, `ltjsbook`, `ltjsreport`, `ljreq` という文書クラスに対応している。ただし `ljreq` は `LuaLATEX` でのみ動作する。

^{*3} ただし `R Markdown` に限って言えば `R` コードの処理時間のほうが問題となることもある。また、`XeLATEX` であっても文献処理や索引処理など関連プログラムを多く使用すればそれなりに遅くなる。

6.2.4 LaTeX のカスタマイズ

PDF のデザインを微調整したい場合、自分で \LaTeX のソースに変更を加える必要がある。よってこの機能は \LaTeX の使用に慣れたユーザのみが使用してほしい。

\LaTeX のソースに変更を加える方法は 2 つ。1 つは `header-includes` を使うことである。

```
header-includes:
  - ...
  - ...
```

これでリスト風にプリアンブルを書くことができる。ただし、% を使った改行エスケープは使用できない。もう 1 つは、出力フォーマットの `includes` に設定する方法で、以下のように命令文ではなく \TeX ファイルを指定する。

```
output: rmdja::pdf_book_ja:
includes:
  in_header: i.tex
  before_body: b.tex
  after_body: a.tex
```

それぞれ、プリアンブル、本文冒頭 (`document` 環境、表紙や目次等の直後)、本文後に挿入される。しかし、 \TeX ファイルの大枠は Pandoc のテンプレートファイルで決まっているため競合することもある。よってテンプレートを確認しながら作る必要がある。テンプレートファイルは Pandoc 独自の簡易なマクロが挿入されている以外は通常の \LaTeX ソースと同じである^{*4}。

テンプレートファイルは R で以下を実行して得られるディレクトリにある。`beamer-j-a.tex.template` は beamer 用、`document-j-a.tex.template` はそれ以外の文書用である。

```
01 system.file("resources/pandoc-templates", package = "rmdja")
```

テンプレートを修正しなければ意図した変更ができない場合、以下のようにして別のテンプレートファイルを指定できる。

^{*4} 現時点では日本語版ユーザーガイドでも未翻訳のパート。しかし if, for 文など簡単な制御構文しかないため、結局 \LaTeX のマクロを使うことが多いだろう。 <https://pandoc-doc-ja.readthedocs.io/ja/latest/users-guide.html#templates>, <https://pandoc.org/MANUAL.html#templates>

```
output: rmdja::pdf_book_ja:  
template: new-file
```

6.3 チャンク/コードおよび出力ブロックのスタイルの一括変更

すでに書いたように、コードブロックのシンタックスハイライトはいくつかのプリセットを適用できる。また、チャンクオプションにはいろいろなものがあり、それらは全て、`knitr::opts_chunk::set()` で以降のチャンクに対して一括して適用できる。詳細は <https://yihui.org/knitr/options/#code-decoration> などを見てもらうとして、いくつかを抜粋する。YAML フォーマットと違い、チャンクオプションは全て R のコードとして評価されるため、論理値は TRUE/FALSE または T/F と表記すること。

- `highlight`: シンタックスハイライトを適用するかどうか。
- `background`: コードブロックの背景色。デフォルトは #F7F7F7 つまり灰色である。
- `tidy`, `tidy.opts`: コードの自動整形を適用するかどうかと、そのオプション。`rmdja` のデフォルト設定では `styler` パッケージを使用している。詳しくは付録 `?autoformatter` を参照。
- `prompt`: コードブロックにプロンプト記号 > を表示するかどうか
- `comment`: 出力ブロックの行頭に表示する記号。デフォルトは ##
- `size`: 出力ブロックのフォントサイズ
- `indent`: 出力ブロックのインデント

6.3.1 行番号の表示

行番号の表示は `attr.source = c(".numberLines", .lineAnchors")` を指定する。

`rmdja` で提供するフォーマットでは、「YAML フロントマターで一括して行番号を表示すること」を指定できる。

```
output: rmdja::pdf_document_ja:  
add_rownumber: true
```

このオプションは上記のチャンクオプションのデフォルト値を変更するものである。よって、これを設定した状態で任意のコードブロックの行番号を非表示としたい場合は逆にチャンクオプション `attr.source = NULL` を指定する。

6.3.2 コードブロックのページまたぎ禁止

長いソースコードを掲載するとページまたぎが発生する。現時点では、`rmdja` の機能としてコードブロックに個別にページまたぎを許可・禁止を指定する機能はない（Pandoc の基本機能の範囲でサポートされていないため）が、一括で指定することはできる。例えば

```
header-includes:  
- \@ifpackageloaded{fvextra}{}{\usepackage{fvextra}}  
-  
↪ \DefineVerbatimEnvironment{Highlighting}{Verbatim}{commandchars=\\"\\{}\\},breaklines,breakanywhere}
```

と書く。ただし直接関係あるのは `samepage=true` だけである。これはテンプレートの記述を上書きするという乱暴な方法なので、他のデフォルトオプションも混在している。`commandchars=\\"\\{}\\` はシンタックスハイライトに関する設定で、基本的に消すべきでない。`breaklines` は行途中の折り返しを許可するもの、`breakanywhere` はどこでも行の折り返しを許可するというもので、いずれもデフォルトのコード自動整形と関係がある。ちなみに TEX ファイルを開いて `highliting` 環境に個別に `samepage=true` オプションを指定すれば個別に折り返しを禁止できる。

第 7 章

(WIP) rmdja による文書作成支援機能

7.0.1 クリエイティブ・コモンズの表記

Web 公開する文書ならばクリエイティブ・コモンズの表記をつけたいところだ。公式サイトで毎回発行するのは面倒なので表示する関数を用意した。ハイパーリンクも付けるようにしている。チャンクでは `results="asis"` オプションが必要になる。また、通常は `echo=F` を設定すべきだろう。冒頭の表記もこれで作成している。もちろんそれぞれの媒体に対応している。

文言の生成は未対応

7.0.2 ルビ表記

ルビはおそらく CJK 言語など一部の言語でしか使われていない（アラビア語とかヘブライ語とかの補助記号は詳しく知らないが多分グリフとしてサポートされてるっぽいので無視）ため、ルビ表記も R Markdown ではサポートされていない。そこで簡単にルビを表示できる関数 `rmdja::ruby()` を用意した。インライン実行で使う。PDF での配置は `pxrubrica.sty` を利用したグループルビである。よって、1字ごとに配置（モノルビ）にしたいとか、突出指定とか、細かいことは HTML タグや CSS や LaTeX コマンドを自分で書く。妥協案として、1字ごとに呼び出す手もある。

グループルビの例: とある科学の超電磁砲, 皇帝ラインハルト, 格館,
 シュワルツ・ランツェンレイター きれうりわり カイザー シュテッヒバルムシュロス
 黒色槍騎兵, 喜連瓜破, MEXICO

分割して出力した例: 喜連瓜破, 黒色槍騎兵,

TODO: それ以外にも便利機能を少しづつ増やしていく予定

第 IV 部

應用編

このパートについて

このパートでは、ここまでで紹介した基本機能の応用で、さまざまな R パッケージやその他の外部プログラムの出力を埋め込む方法を紹介する

第8章

様々なグラフィックプログラムの埋め込み

8.1 tikz を使う

L^AT_EXで使われる `tikzdevice` を利用して、直接 `tikz` の記述による画像を埋め込むことができる。チャンクのエンジンを `tikz` とすることで使用でき、相互参照やキャプション、画像サイズの指定といったチャンクオプションも使える。図 8.1 は `mathcha` というサービスで作成した図を `tikz` のソースとしてエクスポートしたものから生成した図である（ソースコードは長大なので掲載を省略してある）。HTML 出力ではデフォルトでラスタ画像に変換され表示される。

TODO: しかし現状では pdflatex 以外のエンジンに変更できないため、日本語表示が難しい。

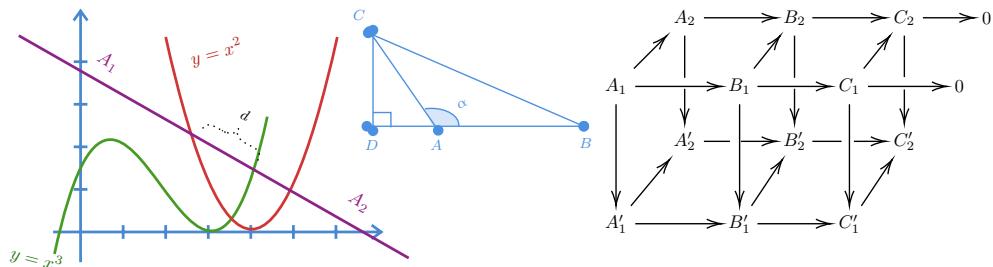


図8.1: `mathcha` からエクスポートした `tikz` の表示

8.2 Asymptote を使う

同様に、Asymptote のプログラムを埋め込むこともできる。私は Asymptote が分からないので RCB Ch. 15.9 Asymptote でグラフィックを作成する と同様のプログラムを書いておく。（図 8.2）。

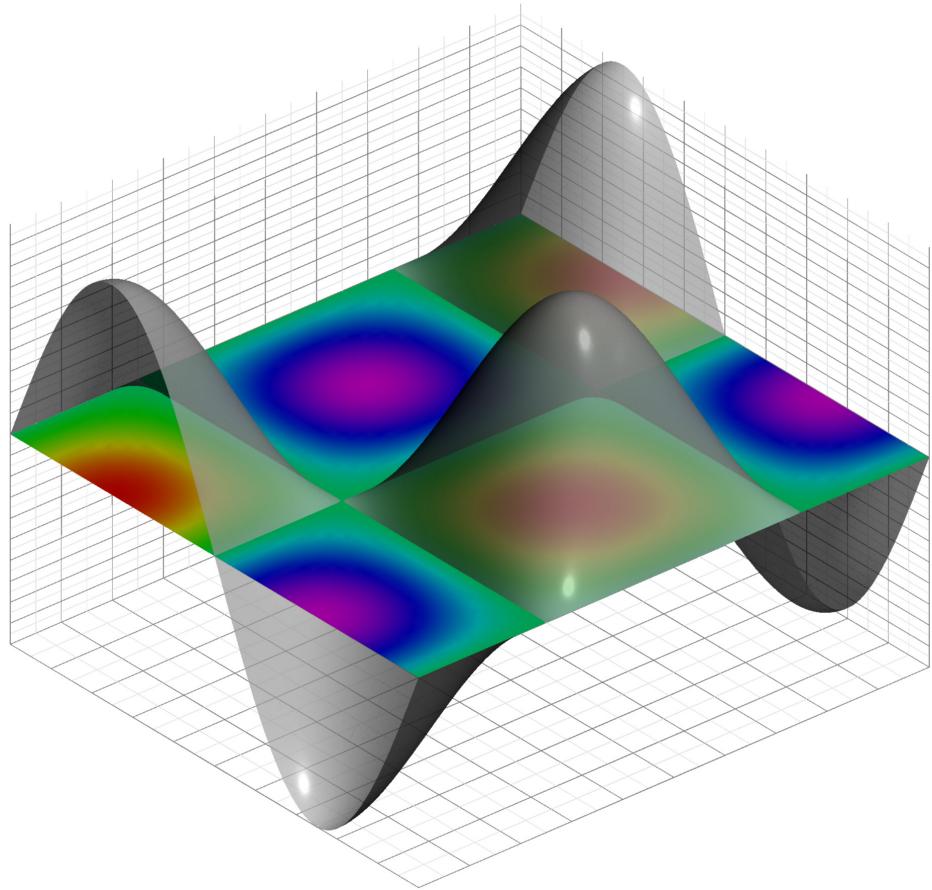


図8.2: Asymptote による画像

8.3 (TODO) その他のプログラム

D3.js なども使える

8.4 (TODO) その他の R プログラム

8.5 DOT 言語とグラフィカルモデル

graphviz などで使用される DOT 言語を使用してグラフィカルモデルを描画することもできる。この場合、チャンクのエンジンを `dot` にするのではなく、エンジンは `r` のままで、`engine="dot"` を指定すると、コードブロックが DOT 言語として評価される。図8.3 がその結果である。

```
01 digraph test {  
02     graph [layout = dot, rankdir = TB];
```

```

03 node [shape = rectangle];
04 rec1 [label = "Step 1. 起床する"];
05 rec2 [label = "Step 2. コードを書く"];
06 rec3 [label = "Step 3. ???", color=blue, style=filled];
07 rec4 [label = "Step 4. 給料をもらう", fontsize=20, fontcolor=red];
08 rec1 -> rec2 -> rec3 -> rec4;
09 }
```

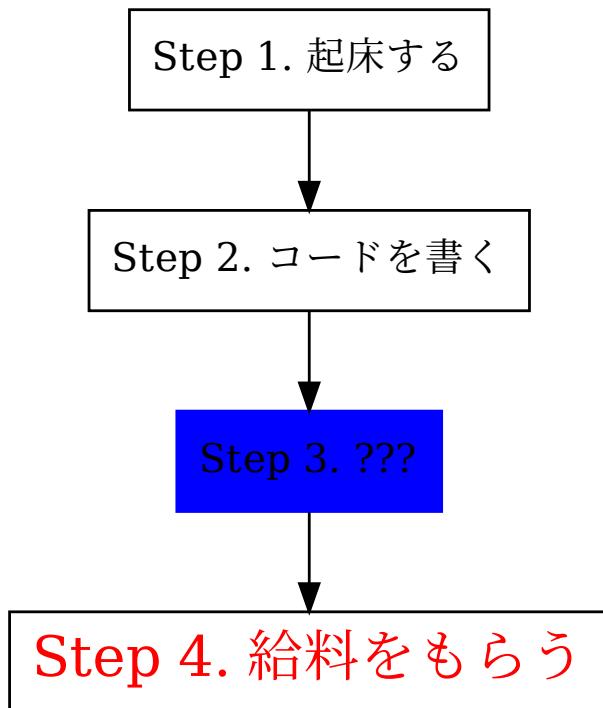


図8.3: DOT の動作確認

なお, RCB, Ch. 4.15 でも紹介されているように DOT 言語は DiagrammeR パッケージを経由して使うこともできる^{*1}が, grViz() 関数の出力は HTML を前提としているため, PDF での出力時のサイズや解像度の調整がうまくいかないことが多い.

一方で, ggdag パッケージは ggplot2 ベースのパッケージなので, 一般的な R グラフィックスと同じ扱いでグラフィカルモデルを描画できる(つまり名前に反して DAG 以外のネットワーク図も記述可能である). ggplot2 風の構文で記述できるので習熟も容易である.

^{*1} 私も詳しいことは知らないが, DiagrammeR::grViz() の構文は本来の DOT と少し異なるようだ, 本来はステートメントごとにセミコロンで区切ることが必要であり, 文字列もダブルクオーテーションで囲まなければならぬが, grViz() ではそのような制約がない.

第 9 章

表のデザイン

9.1 kableExtra による表のスタイルのカスタマイズ



kableExtra パッケージは knitr::kable の拡張であり、様々なスタイルの表を出力できる。そしてそれは HTML でも PDF でも有効である。

まず、knitrExtra::kbl() は既に紹介した kable() のラッパーであり、内部で呼び出すのは kable() のため booktabs といった従来の引数が使える上に、オプション設定の構文がより分かりやすくなっている。さらに kableExtra の独自機能として、表 9.1 にみられるように条件書式のような装飾が可能である^{*1}。

```

01 colnames_mtcars_ja <- c(
02   "ガロン毎マイル", "シリンド", "排気量", "総馬力",
03   "ギア比", "重量", "加速性能", "トランスミッション", "ギア数", "キャブレター
04   数"
05 )
06 that_cell <- c(rep(F, 7), T)
07 mtcars[1:8, 1:8] %>%
08   kbl(
09     booktabs = T, linesep = "",
10     format = if (knitr:::is_latex_output()) "latex" else "html",
11     caption = "(ref:kableextra-color-cap)",
12     col.names = colnames_mtcars_ja[1:8]
13   ) %>%
```

^{*1} ところで私は自動車の性能を表す用語に詳しくない。これは mtcars データセットの列名を日本語訳したのだが、した表記に誤りがあれば指摘してほしい。

表9.1: kableExtra パッケージを利用した表の作成、公式ドキュメントの用例より

	ガロン毎マイル	シリンド	排気量	総馬力	ギア比	重量	加速性能	トランスマイッション
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	4

```

13  kable_paper(full_width = F) %>%
14  kable_styling(latex_options = "scale_down") %>%
15  column_spec(2,
16    color = spec_color(mtcars$mpg[1:8]),
17    link = "https://haozhu233.github.io/kableExtra"
18  ) %>%
19  column_spec(6,
20    color = "white",
21    background = spec_color(mtcars$drat[1:8], end = 0.7),
22    popover = paste("am:", mtcars$am[1:8])
23  ) %>%
24  column_spec(9,
25    strikeout = that_cell, bold = that_cell,
26    color = c(rep("black", 7), "red")
27  )

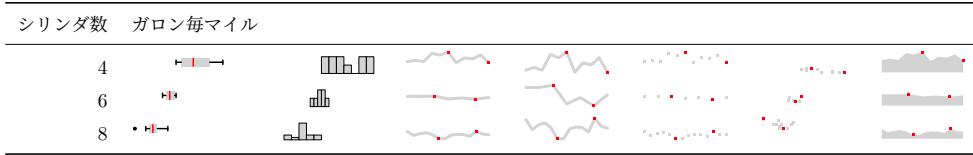
```

上記の例で使われている、kableExtra の便利な機能をいくつか挙げる。

- kbl(col.names =) で列ラベルを指定できる（これは kable() でも使える）。列名の変更ではないので以降も同じ名前で変数を参照できるが、表示されるのは列ラベルである。現状、日本語（マルチバイト文字）を変数名に与えることを想定していないパッケージはまだ多く、またデータフレームの仕様として列名に () などを使えないためこの機能が役に立つ。
- PDF 出力時の表の幅調整を簡単にするオプションがいくつか用意されている。kable_styling() の latex_options = "scale_down" や full_width = T である。前者は本文領域に収まるように自動で表を縮小するもので、後者は表内での改行を許容することで超過した表の幅を本文の幅に合わせるオプションである。もしより細かい調整が必要なら、column_spec() で列ごとに幅を指定することもできる。

グラフのインライン挿入も可能である（表 9.2）。しかしこのような細かいグラフの羅列は可読性に欠けることに注意する。

表9.2: kabeExtra パッケージによる表内グラフ、公式ドキュメントの用例より



```

01 mpg_list <- split(mtcars$mpg, mtcars$cyl)
02 disp_list <- split(mtcars$disp, mtcars$cyl)
03 inline_plot <- data.frame(
04   cyl = c(4, 6, 8), mpg_box = "", mpg_hist = "",
05   mpg_line1 = "", mpg_line2 = "", mpg_points1 = "", mpg_points2 = "", mpg_poly = ""
06 )
07 inline_plot %>%
08   kbl(
09     booktabs = T, format = if (knitr:::is_latex_output()) "latex" else "html",
10     caption = "(ref:kableextra-plot-cap)",
11     col.names = c("シリンド数", "ガロン毎マイル", "", "", "", "", "", ""))
12   ) %>%
13   kable_paper(full_width = FALSE) %>%
14   kable_styling(latex_options = "scale_down") %>%
15   column_spec(2, image = spec_boxplot(mpg_list)) %>%
16   column_spec(3, image = spec_hist(mpg_list)) %>%
17   column_spec(4, image = spec_plot(mpg_list, same_lim = TRUE)) %>%
18   column_spec(5, image = spec_plot(mpg_list, same_lim = FALSE)) %>%
19   column_spec(6, image = spec_plot(mpg_list, type = "p")) %>%
20   column_spec(7, image = spec_plot(mpg_list, disp_list, type = "p")) %>%
21   column_spec(8, image = spec_plot(mpg_list, polymin = 5))

```

その他細かい使用上の注意をいくつか挙げる。

- kableExtra::: で参照するのではなく、最初にパッケージをロードしたほうが不具合が起きにくい。
- PDF 出力する場合、多くの L^AT_EX パッケージのロードが必要だが、rmdja の PDF フォーマットはいずれもテンプレートに組み込んでいるため手動設定は必要ない。
- knitr:::kable() または kableExtra:::kbl() の format で HTML/tex の出力を決める。現在は判定が自動化されたとのことだが、まれに不具合があるという報告もみられる。よって、どちらも出力したい場合は上記のように format = knitr:::is_latex_output() で条件分岐させるのが 1 つの手である。
- 表のキャプションは図のようにチャunkオプションに指定するのではなく、

`kbl()/kable()` の `caption` 引数に指定する

- キャプション内に Markdown 記法や相互参照など特殊な構文を含めたい場合は、`escape = F` を指定する。
- もし画像が大きすぎて PDF で余白からはみ出てしまうならば、`kable_styling(latex_options = "scale_down")` を追加することで自動縮小してくれる。

その他、テキストの回り込み、画像の挿入など様々なことが可能である。詳細は公式の解説である “[Create Awesome HTML Table with knitr::kable and kableExtra](#)” および [PDF 版](#) が役に立つ。

9.2 `formattable` パッケージとの併用

`formattable` パッケージは以前からある表を装飾するパッケージである。`kableExtra` との併用も可能だが、`LATEX` に対応しておらず、HTML タグを tex ファイルに吐き出してしまうため動作しない。PDF にも同様に表示するには [StackOverflow](#) で提案されているように、`webshot` 使うなど工夫が必要である。そこまでしてこの装飾にこだわるメリットは薄いと私は考えるので現在この問題に対応する予定はない。`kableExtra` か後述する `huxtable` を使うべきだと考える。

9.3 `huxtable` パッケージによる作表

表9.3: ‘`huxtable`’

のロゴはランダム

に生成される。

			h			
		u				
				x		
t				a	b	
				l	e	

`huxtable` は HTML と `LATEX` に対応した作表パッケージであり、公式ドキュメントによると他の類似パッケージと比較して多機能であることを強調している。全体的に `tidyverse` を意識した構文が用意され、`kableExtra` のようにパイプラインを使った記述が摂る。さらに「1行ごとに背景色を変更」「`stargazer` 風の表」などよく使われるスタイルを簡単に設定できるようになっていたり、はては `tidyverse` のような表のロング・ワイド変形機能まで備えている。例えば公式用例集を参考に、条件書式を付けた表 9.4 を作成する。

```

01 require(huxtable)
02 head(mtcars[1:5]) %>%
03   set_names(colnames_mtcars_ja[1:5]) %>%
04   as_huxtable(add_rownames = "Model") %>%
05   set_caption("(ref:huxtable-example-cap)") %>%
06   set_bold(1, everywhere, T) %>%
07   theme_article() %>%
08   map_text_color(
09     everywhere, "ガロン毎マイル", by_colorspace("navy", "red", "yellow")
10   ) %>%
11   map_background_color(
12     everywhere, "総馬力", by_quantiles(0.8, c("white", "yellow"))
13   ) %>%
14   map_italic(everywhere, "Model", by_regex("Merc.*" = T)) %>%
15   set_number_format(col = "ギア比", value = fmt_percent(digits = 2))

```

表9.4: huxtable パッケージによる作表

Model	ガロン毎マイル	シリンド	排気量	総馬力	ギア比
Mazda RX4	21	6	160	110	390.00%
Mazda RX4 Wag	21	6	160	110	390.00%
Datsun 710	22.8	4	108	93	385.00%
Hornet 4 Drive	21.4	6	258	110	308.00%
Hornet Sportabout	18.7	8	360	175	315.00%
Valiant	18.1	6	225	105	276.00%

huxtable パッケージの関数の多くには `set_` という接頭辞がついているものとそうでないものがある。ついているものは上記のようにパイプラインでつなげて使うために用意された関数群で、ついていないものは R 組み込みの `colnames()` のように 1 行ごとに処理を書くスタイルに向いている。そのままでは罫線の設定が `set_top_border()`, `set_bottom_border()`, などしかなく、複雑な条件を指定するのが大変だが、`ggplot2` のテーマ関数のようにスタイルのプリセットが `theme_*`() の名前でいくつか用意されている。例えば上記では `theme_article()` という学術論文風テーマを適用し、表の上下とヘッダにだけ罫線を引いている。条件書き式は `map_*`() 関数群で実行できる。また、フォーマットは `set_number_format()` に値を変換するフォーマット関数を与える形で適用できる。こちらはパーセンテージなども正しく表示できる。

- テーマ設定はグローバルオプションでも設定できる。例えば `options(huxtable.knit_print_df_theme`

```
= theme_article).
```

なお、動作させるにあたっていくつか注意が必要である。

- `huxtable` は使用しているフォーマットを自動判別して相互参照用のラベルを生成しているが、`rmdja` で HTML を出力する際に正しく認識されないようである。`options(huxtable.bookdown = T)` Rmd の冒頭で実行して、`bookdown` としての処理を強制することで解決できる^{*2}。
- `huxtable` は EPUB 形式に対応していない。警告が表示されるだけのこともあれば、`knit` 処理がハングアップしてしまうこともある。

また、`huxreg()` は名前の通り回帰分析の結果を表にするなど `stargazer` パッケージに似た機能を提供する。これも同じクラスなので同様にスタイル設定が可能である（表 9.5）。

```

01 lm1 <- lm(mpg ~ cyl, mtcars)
02 lm2 <- lm(mpg ~ cyl + hp, mtcars)
03 glm1 <- glm(I(mpg > 20) ~ cyl, mtcars,
04   family = binomial
05 )
06 huxreg(lm1, lm2, glm1, stars = NULL, error_pos = "below", note = "括弧内は標準誤
  ↪ 差") %>%
07   set_caption("(ref:huxreg-example-cap)") %>%
08   set_text_color(everywhere, "model1", "green") %>%
09   set_text_color(everywhere, "model2", "blue")

```

その他の作例は CRAN の公式 vignettes を参考にせよ。

9.4 TeX/HTML を出力する関数

`stargazer` や `pander` のように表を出力するための HTML や LATEX や Markdown のソースコードを出力してくれるパッケージがある。これらは `results='asis'` のチャンクオプションを指定することで関数の出力するテキストをそのまま埋め込むことができる。よって、あとは HTML カ LATEX カといった出力形式の違いに気をつければ表示できる。`stargazer` はオプションが豊富で、例えば表 9.6では、`align = T` を指定することで、`dcolumn.sty` を使って数字のアラインメントを小数点で揃えることができる。その他、複数の回帰モデルの結果を並べて表示したり、その際の表示スタイルも比較的かんたんに調整できる。`stargazer` の詳しい使い方はむかし私が書いた『R での分析結果を LaTeX 形式で出力するパッケージ比較（後編）』を参考に。

^{*2} おそらくはこの辺の処理の問題だが、未解決 <https://github.com/hughjonesd/huxtable/blob/3eb96b62a5fde1000924daba39078f2e72839383/R/knitr.R>

表9.5: `huxtable::huxreg()` による出力

	(1)	(2)	(3)
(Intercept)	37.885 (2.074)	36.908 (2.191)	64.400 (17449.775)
cyl	-2.876 (0.322)	-2.265 (0.576)	-10.781 (2908.296)
hp		-0.019 (0.015)	
N	32	32	32
R2	0.726	0.741	
logLik	-81.653	-80.781	-4.780
AIC	169.306	169.562	13.561

括弧内は標準誤差

```

01 require(stargazer)
02 stargazer(mtcars,
03   header = F, align = T,
04   type = if (knitr:::is_latex_output()) "latex" else "html",
05   title = "(ref:stargazer-title)",
06   label = knitr:::opts_current$get("label")
07 )

```

ただし `stargazer` はここ数年更新されておらず, R Markdown に対応した機能追加なども行われていないため, 相互参照に対応していない. `bookdown` リポジトリの [issue #175](#) にあるように, PDF に限れば簡易的な方法で対処できるが, HTML でも相互参照するには `stargazer_bookdown` のインストールが必要になる. これはインストールしただけで従来の `stargazer` が相互参照に対応するようになる.

その他, `Hmisc::latex()`, `stats::xtable()` という古典的な関数がある. 後者は名前の通り `LATEX` のソースをかなりの自由度で出力できるが, ここまでやるならもう最初から全部 `LATEX` で書いたほうがいいのでは, という気もする. `LATEX` に詳しくない場合, かえって難しいかも知れない. 既に紹介した `kableExtra`, `huxtable` などでできる範囲でやったほうが簡単だろう.

表9.6: **stargazer** による要約統計量の出力

Statistic	N	Mean	St. Dev.	Min	Max
mpg	32	20.091	6.027	10.400	33.900
cyl	32	6.188	1.786	4	8
disp	32	230.722	123.939	71.100	472.000
hp	32	146.688	68.563	52	335
drat	32	3.597	0.535	2.760	4.930
wt	32	3.217	0.978	1.513	5.424
qsec	32	17.849	1.787	14.500	22.900
vs	32	0.438	0.504	0	1
am	32	0.406	0.499	0	1
gear	32	3.688	0.738	3	5
carb	32	2.812	1.615	1	8

9.5 その他の作表パッケージ

そのほか有名なパッケージとして, **DT**, **flextable**, **gt** などがある. **DT** は [DataTables ライブラリ][<https://datatables.net/>] を利用してインタラクティブな表ウィジェットを作成し, **flextable** は Word へのエクスポート機能をフィーチャーしているが, PDF に対しては画像として出力するなどくせがある. **gt** は RStudio 社が開発しているパッケージで, **huxtable** のように **tidyverse** 的なシンプルな構文が用意されている一方で, まだ R Markdown の相互参照機能に対応していない^{*3}. 以上からすでに紹介した **kableExtra** や **huxtable** がより **rmdja** の対応する出力媒体に適したパッケージであり, それ以外のパッケージの詳しい紹介は避ける.

RCB 10.3 他の表作成パッケージ も参考にせよ.

^{*3} Issue #115 にあるように, 機能を追加したいという声はある. しかし現時点では **gt** の R Markdown 対応作業の優先度は高くないようである.

第 10 章

文献引用の詳細設定

10.1 (u)pBIBTEX を使う

.bst ファイルのスタイルを使いたい場合、(u)pBIBTEX が必要であり、そのためには現在の R Markdown および **rmdja** の仕様では、YAML フロントマターとグローバルオプションを変更する必要がある。例えば `jecon.bst` を使いたい参考文献リストを出力したい場合、YAML フロントマターは以下のような記述となる。

```
output:
  rmdja::pdf_book_ja:
    citation_package: natbib
    citation_options: numbers
  biblio-style: jecon
  bibliography: citations.bib
```

BibLaTeX では `citation_options` にスタイルまで指定していたが、`natbib` を選択した場合 `biblio-style` に .bst ファイルを指定し、フォーマット関数の `citation_options` 引数は `natbib.sty` に対するオプションを指定する項目となる（トップレベルの `natbiboptions` でも可）。上記の例では `numbers` を指定しているため本文中の参照トークンは [1], [2, 3] のような番号形式となる。デフォルトは `authoryear`、つまり「著者 - 出版年」形式である。

次に、最初のチャンク、またはコンソールでグローバルオプションを変更する。

```
options(tinytex.latexmk.emulation = F)
```

この状態で `knit` または `build` すれば .bst ファイルのスタイルが適用される。

このような操作が必要な理由を説明する。`rmarkdown` は `tinytex` というパッケージでインストールされたスタンドアローンな LATEX 处理系で PDF を生成してい

表10.1: biblatex の ‘bilio-style’ で指定できるもの一覧

名称	概要
‘numeric’	’[1]’ のような番号
‘alphabetic’	著者名の略称 + 出版年 2 枠
‘authoryear’	著者-出版年形式, natbib の標準と同じ
‘authortitle’	著者名のみ, リストでは出版年は後置され, 引用子では脚注になる
‘verbose’	authortitle と同じだが, 引用子にリスト同様の内容を出力する
‘reading’	個人的なリーディングリスト向け. ファイルやメモ欄も出力する
‘draft’	.bib ファイルの ID で表示. 名前通り下書き
‘debug’	‘bib’ の全フィールドを表示

る。しかしこれは (u)pBBIBTEX の使用が想定されていない。 (u)pBBIBTEX は日本語コミュニティで開発されたマルチバイト文字対応版 BBIBTEX だから, rmarkdown 開発メンバーたちがこれらの存在に詳しくないのも仕方ないことだ (YiHui 氏は中国出身だが, 中国語圏では BibLaTeX を使うことが多いようだ)。冒頭のチャンクで options(tinytex.latexmk.emulation = F) を指定することで, 自分のマシンにインストールされた, おそらくあなたが普段使っているであろう LATEX 处理系に処理させることができる。この方法では latexmk コマンドを使用して PDF の生成が行われる, その場合 TeX Wiki に記載のあるように, 日本語出力のため .latexmkrc ファイルが必要となっている。 rmdja では natbib を指定した場合に自動でカレントディレクトリに .latexmkrc をコピーするようにしている。しかしログが残らないなどデバッグしづらいところがあるため, このやり方はやや使いづらく LATEX に対するそれなりの知識を要する。たとえばこの説明を読んで初めて latexmk の存在を知った, そもそも LATEX をどうインストールしたか記憶がない, といった人は慣れるまで大変かもしれない。

10.1.1 (TODO) pandoc-citeproc と CSL について

TODO: セメテ日本語文献の姓名表示をなんとかしたスタイルを用意する

10.1.2 (WIP) BibLaTeX について

BibLaTeX の全てのオプションに対応しているわけではないので詳しいことは BibLaTeX のドキュメントを読んでいただきたい。残念ながら, 日本語の情報は非常に乏しい。ここではよく使う style のことだけ言及する。

citation_packages/biblatextoptions で指定できるのは, インクルード時の style= に指定できるものに対応する (表 10.1)。つまり, 引用文献の見出しや, 表示順といった設定である。これは引用リストと本文中の引用子のスタイル両方に影響する。

その他 apa, ieee など特定の学会が使用するスタイルも用意されているが, これらは基本欧文しか想定していないし, アカデミックの事情に詳しい人しかこれらの使用に

こだわらないだろうから詳しくは解説しない。これらを含めたそれぞれの出力例は https://www.overleaf.com/learn/latex/biblatex_bibliography_styles に一覧があるのでそちらを参考に。なお、引用リストのスタイルと引用子のスタイルを個別にすることはできる (bibstyle/citestyle)

現時点では各分野の学会で日本語文献に対応した BibLaTeX フォーマットを配布しているという情報は見つけられなかった。参考として私のブログで対応のヒントについて書いた^{*1}。

TODO: その他の非ラテン文字、キリル文字、アラビア文字へプライ文字等は？

なお、普段文献管理ソフトを使っていないが、数本程度の文献を引用リストに載せたい利用者は、biblatex の構文を利用して書くのがよいかかもしれない。

^{*1} <https://ill-identified.hatenablog.com/entry/2020/09/20/231335>

第 11 章

(TODO) Web アプレットの挿入

11.0.1 TODO: plotly

11.0.2 TODO: shiny

第 12 章

Python スクリプトの埋め込み

Python スクリプトを埋め込むこともできる。方法は 2 通りあり、都度システムコマンドから呼び出す方法と、`reticulate` パッケージを使うものがある。`reticulate` 登場以前はチャンクごとに呼び出していたため複数のチャンクに分割して記述するのが難しかったが、現在は `reticulate`¹ パッケージを利用することで R と同じような感覚で、あるいは Jupyter のコードセルと同じような感覚で書ける。

`pyenv` を使用する場合、共有ライブラリのインストールが必要なことに注意¹。

初めて使う場合は先に `reticulate` 単体で Python が実行できるか検証してからの方が良い。

```
01 require(reticulate)
02 # Python エンジンを認識しているか確認
03 py_discover_config()
04 repl_python()
05
06 # 以下、Python コマンド実行、`exit` で抜ける
```

現在 (`knitr` 1.18 以降) は R Markdown はデフォルトで `reticulate` を使う。システムコマンド経由にしたい場合はチャンクオプション `python.reticulate=F` を設定する、あるいは `reticulate::eng_python` に変わるエンジンを自作する²。

あとはチャンクのエンジンに `r` ではなく `python` を指定することで Python コードを埋め込める。

`matplotlib` エンジンで描いたグラフに日本語フォントを埋め込む場合、`matplotlib-japreset` を使えば必要な設定を一括で行う。

*1 詳しくはこちらを参考に <https://ill-identified.hatenablog.com/entry/2019/11/15/010746>

*2 参考: https://rstudio.github.io/reticulate/articles/r_markdown.html

```
pip install -U
  ↵ git+https://github.com/Gedevan-Aleksizde/matplotlib-japreset.git@master
```

```
01 from matplotlib_japreset import mplj_cairo
```

```
Noto Sans CJK JP not found
matplotlib-japreset Cairo mode
font: Noto Sans CJK JP
```

```
01 from matplotlib import rcParams
02 from plotnine import *
03 from plotnine.data import mtcars
04 rcParams['font.family']
```

```
['Noto Sans CJK JP']
```

`matplotlib-japreset` によって主要 OS で標準インストールされている日本語フォントが自動的に選ばれる。気に入らない場合は `rcParams['font.family']` に好きなフォント名を上書きする。

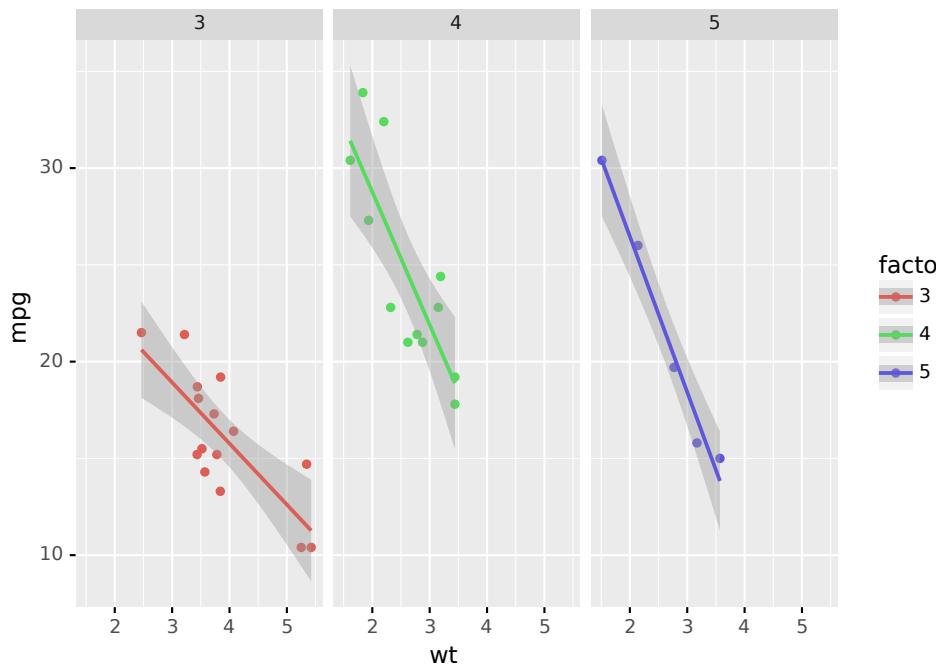
```
01 ggplot(mtcars, aes('wt', 'mpg', color='factor(gear)')
02 ) + geom_point() + stat_smooth(method='lm') + facet_wrap(~gear')
```

現状では `matplotlib` の標準出力や警告も表示されてしまうため、チャンクオプション `results='hide'`, `warning=F`, `message=F` で隠すと良い。

12.1 Python のグラフィックに関する制約

`matplotlib` ベースのグラフィックを出力したい場合、いくつかの制約がある。

- `matplotlib > 3.2` では R がクラッシュするため、3.2 を使用する必要
- `axes` を使用した場合 (`subplot` などが依存), `matplotlib.pyplot.show` の呼び出しと、次に別のグラフを呼び出す前の `matplotlib.pyplot.close()` が必要
- `seaborn.FacetGrid` を Cairo デバイスで保存できない (= フォントのサブセット化処理が複雑になる)

図12.1: Python の `plotnine` によるグラフ表示例

よって、現状では `matplotlib` エンジンでグラフィックを描くときはなるべく `plotnine` を使ったほうがトラブルが少ない。

また `plotly`, `bokeh` などの `matplotlib` に依存しないモジュールは PDF には対応していないため直接表示できない。一旦画像を保存して、あらためて画像ファイルを埋め込む必要がある。

第 V 部

製本と多様な形式への対応

第 13 章

PDF の文書クラス

HTML は利用者側が見え方をある程度カスタマイズできる。かつて存在した Evernote Clearly やカスタム CSS を使って、そのぶん PDF は作成者側がよりレイアウトに注意を払うことになるだろう。本稿では文章の区切りを章立てにしている。しかし PDF 数十ページしかない文書を大きな文字サイズの見出しで区切るのは少しものものしい感じがする。YAML フロントマターを変更すれば、トップレベルの見出しを変更できる。

`pdf book in Japanese` は “book” ということで書籍の組版をデフォルト設定にしている。もう少し小規模な文書ならば、レポートや論文記事形式のほうが良いかもしれない。例えば、以下のように指定する。

```
documentclass: bxjsreport
```

`documentclass` には LaTeX の文書クラスファイル (`.cls`) ならなんでも与えることができるが、X_ELATEX または LuaLATEXで日本語文書を作成することを想定しているため、以下 2 種類の BXjscls の文書クラス^{*1}の中から選ぶとよい。デフォルトは `bxjsbook` なので、これは明示的に指定する必要はない。

- `bxjsbook`
- `bxjsreport`

このうち、`bxjsbook` が `pdf book in Japanese` のデフォルト設定となっている。`rmdja::texlogo("LaTeX")` の文書クラスは、行間や見出しのレイアウトなどを日本語文書に準じたものにするが、それ以外の細かい調整は `_output.yml` や `_bookdown.yml` の設定を書き換えて調整する。それでも不十分な場合は、`.tex` ファイルや `pandoc` テンプレートを直接編集したり、追加のスタイルファイルを読み込んだりするしかない。

*1 詳細はここにあるドキュメント参照: <https://www.ctan.org/pkg/bxjscls> 但し、スライド用クラスである `bxjsslide` の使用は想定していない。また、`bxjsarticle` を使う場合は後述の `pdf article in Japanese` テンプレートから作成したほうがよい。さらに LuaLATEXを使用するならば `lualatex-ja` で提供される日本語文書クラスも指定することができるが、あまりつかったことがないためレイアウトに不備があるかもしれない。以降は PDF ファイルで出力できる各形式についてこまかく解説する。

しかし、おそらくはこういった細かい調整が必要になることはすぐないだろう。以降では、`rmdja` が用意しているプレゼンテーションや論文形式のテンプレートを紹介する。

13.1 プrezentation資料の作成

`beamer_presentation_ja` は `rmdja` の最初期からあったフォーマットで、そもそも当初はこれを作るのが目的だった。このフォーマットは Beamer を使用してプレゼンテーション用スライドを PDF ファイルで作成する。Beamer は `rmdja::texlogo("LaTeX")` の文書クラスの 1 つで、`rmarkdown::beamer_presentation` はこれを利用しているが、例によって日本語表示は想定されていないため、そのためのもろもろの調整込みのラッパーフォーマットである。ただしスライド資料なので組版の禁則処理のような細かい調整は用意していない。`rmdja` ではスライドは PDF 以外の出力は不可能である^{*2}。

通常の文書と違い、デザインを決めるのは主に `theme` である。デフォルトでは `metropolis`^{*3} である。日本語表示のために調整してあるものの、日本語表示と直接関係ない部分はカスタマイズの余地としていじっていないが、テンプレートには私の好みが反映された調整（プログレスバーの位置調整）が YAML フロントマターに直接書き込まれている。

また、日本語表示と直接関係ないアレンジとして、デフォルトの文献引用のスタイルが変更される。

1. 本文での引用スタイルは番号形式 (`biblatex` の場合は `citestyle=numeric, natbib` の場合は `numeric` オプション)。
2. 「参考文献」というセクションタイトルのみのスライドが冒頭に自動で挿入される
3. 引用された文献の数に応じてフレームが自動分割される
4. これらの参考文献フレームでは上部のタイトルが表示されない
5. 文字サイズが脚注サイズに縮小

という設定になっている。通常のプレゼンテーションでは大量の参考文献を読み上げることは少ないと想定で、紙面の限られたスライドに参考文献のみ羅列したスライドでページ数が増えないように考慮したためこうした。これは既に作成した `my_latex_templates` のテンプレートとほぼ同じである。

さらに、Beamer テンプレート特有の設定をいくつか紹介する。

- プログラムはデフォルトで非表示 (`echo=F`)
- 出力する画像の大きさ `fig_width, fig_height` は `beamer` のデフォルトの大きさに連動している。そして `out_width, out_height` はいずれも "100%" にしているため、概ね `beamer` の画面と同じ大きさになる。

^{*2} HTML 形式のスライドはサポート対象外である。日本語文書特有の処理はあまりないということ、普段と違う環境で表示することの多いであろうスライド資料はなるべく環境に依存しない方法で表示すべきと考えているのが理由である。HTML でスライドを作成したい場合、次のページが参考になる：
https://kazutan.github.io/SapporoR6/rmd_slide.html#/

^{*3} なお `metropolis` テーマ開発者は Fira Sans フォントの使用を想定しており、ビルト時にフォントがないという警告が出ることがあるが無視して良い。（参考：<https://github.com/matze/mtheme/issues/280>）

- プログラムに行番号を表示する `code_rownumber` は `FALSE` している
- テーマは `metropolis` を使っているが、昔ながらのテーマも可能である。昔からあるテーマの比較には `Beamer Theme Matrix` というページが便利である。他にも近年登場したテーマがいくつか存在するが、日本語をうまく表示できなかったり `rmdja::texlogo("XeLaTeX")/rmdja::texlogo("LuaLaTeX")` に対応していなかったりするものも多い。他に日本語に対応したテーマとして、`sakuratheme` が存在する。
- `beamer` のアスペクト比はデフォルトで `4:3` であり、YAML フロントマターで指定できる。例えば `16:9` に変更したい場合

```
classoption:
  - aspectratio=169
```

となる。指定可能なのは `3:2, 4:3, 5:4, 14:1, 14:9, 16:9, 16:10` で、上記のようにコロンを抜いて数字のみで指定する。この `classoption` は LATEX の文書スタイルに対するオプション全般を与えるためにあるため、(beamer スタイル以外にも) 他にもいろいろ存在する。

詳細は [beamer の公式ドキュメント](#) を参考に。

`rmdja` の Beamer 用テンプレートの実際の表示例は `examples` にある。

13.2 (WIP) 卒業論文の作成

卒業論文... というか学術論文での体裁で PDF ファイルを作成することも可能である。`pdf article in Japanese` という名前のテンプレートで論文形式の PDF ファイルを用意している — HTML 形式で論文提出を要求するという話は聞いたことがないので PDF のみ対応している。

書籍形式との違いは、

- 文書の見出しが「X 章」ではなく「1. YYYY」のようになる（したがって、Rmd ファイルで # で記述した見出しへ、PDF ではセクションタイトルとなる）
- 余白のとり方が書籍のように見開きを想定したものでなくなる

など些細である。実際のところ、文書テンプレートの設定を少しいじっている程度のことしかしていない。テンプレートを開いて確認すればわかるように、

```
output:
  rmdja::pdf_book_ja:
    toc: false
    pandoc_args:
```

```
- '--top-level-division=section'
documentclass: bxjsarticle
```

という設定を追加しているだけである^{*4}.

大学によっては論文の体裁が細かく指定されている場合もあるかもしれない。例えば1ページあたりの行数や、1行あたりの文字数とか。例えば1ページあたり50行、1行あたり40字とする場合、以下のような設定を追加する。ただし、行数は図表の挿入などで変動するし、プロポーショナルフォントや字幅の異なる欧文を多用すれば1行あたりの文字数は多くなりうる。

```
classoptions:
- 'number-of-lines=50'
- 'textwidth=40zw'
```

さらに、カラー印刷が許容されない場合もある。`ggplot2` は `scale_*_grey()` などでカラーパレットを簡単に変更できる（図 13.1）。

```
01 ggplot(
02   mutate(mtcars, cyl = factor(cyl)),
03   aes(x = mpg, y = wt, color = cyl)
04 ) +
05   geom_point() +
06   labs(x = "マイル毎ガロン", y = "重量 (1000 ポンド)") +
07   theme_bw() +
08   scale_color_grey() +
09   scale_fill_grey()
```

13.3 (WIP) 小説の執筆

作家の京極夏彦氏は自分の作品を1ページごとに切り取っても作品として成立するようなレイアウトにこだわっているらしいが、すでに説明したように技術文書や学術論文では図表の配置や改行などにこだわることがあまりない。しかし、不可能ではない。HTMLでは難しいが（不可能ではないがHTMLでやるメリットが感じられない）で対応する気がな

^{*4} このテンプレートでは論文形式のフォーマットとして `bxjsarticle` を使用している。LuaLaTeXを使用するならば代わりに `ltjsarticle` クラスも使用可能なはずだが、私は使ったことがないので説明を省く。

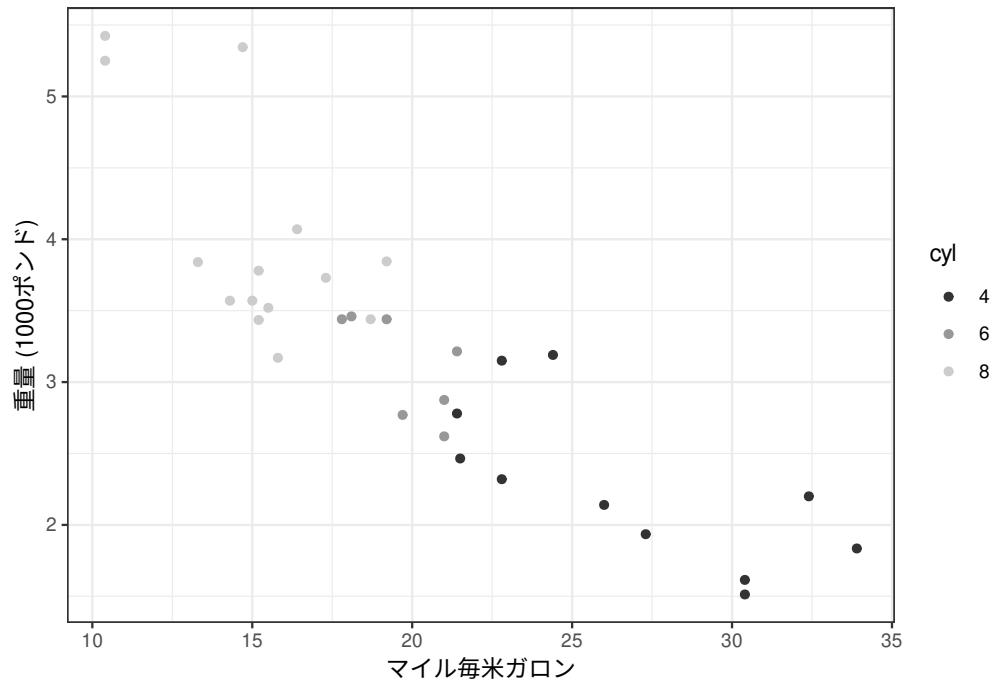


図13.1: グレースケールでのプロット

い), PDF ではある程度のレイアウトの制御が可能である。ただし、本当に厳格な JIS 準拠の組版にこだわるなら、おそらく tex ソースを直接編集しなければならない。

`rmdja` で用意されている縦書き文書テンプレート `pdf vertical writing in Japanese` は、`jlreq` を利用して^{*5} 縦書き文書の PDF を作成する (図: 13.2)。HTML には未対応である。

^{*5} `luatex-jja` にも縦書き文書クラス `ltjt` シリーズが存在するが、公式ドキュメントにすら詳しい解説がなかったため採用しなかった。

第二章 縦書きの例

字下げには全角空白が必要?

雪の結晶の良い顕微鏡写真を撮るには、気温が零度以下になつておいて、別に濡れないように冷しておいた硝子板^{ガラス}に結晶を受けて、普通の顕微鏡写真を撮るようにして写せばよいのである。気温が零下五度以下であると大分楽であるが、零度に近い時はまごまごしていると肝心の結晶がとけたり、冷しておいた硝子板に一面に霜がついたりしてなかなか厄介になる。何よりも大切なことは、硝子板に載った沢山の結晶の中どれを撮るかを決める敏速な決断である。まず眼で見て、次に顕微鏡下で写真を撮る価値があるか否かを調べて、決断をして、暗函をかぶせて、さてシャッターを切るまでの時間が、馴れてくれば二十秒位で出来るようになる。この間が五分位に感じられるようになれば大丈夫である。覗きながら、写真を撮るような便利な器械は、特に寒い所では故障が多くて駄目なよう思われる。一冬にただ一度見るか見ない位の珍しい結晶の時に、得てそのような故障が起りやすいようである。

初めの年は廊下の吹きさらしの寒い所を選んで有り合せの顕微鏡で写真を撮ってみたのであるが、結果はなかなか面白かった。北海道は雪の結晶の種類には極めて恵まれているようで、わずか一冬の

雪の結晶は驚くべく繊細な形をしれども比較にならぬ位の美しさを見、が、丁度開きかけた薔薇の花弁の縁

観測で、いく特別のものを除いては、ほとんどすべての結晶の型が見られた。勝岳の中腹にある白銀荘という山小屋

この小屋は十勝の吹上温泉の近くに在る。周囲は亭々たる蝦夷松^{えぞまつ}と樺^{ひのき}

マスの木のように雪に枝を垂れています。白樺の化けたような巨樹が、細々と空に向って伸している。これらの出している固体の表面はことごとく黒い樹幹を除いては周囲は、た日の青空のみが鮮かな濃い色彩を、れる日は極めて稀れで、冬半年の間、いようである。雪が降り出すと四辺、写真を見るような世界になってしまふ。どのような色彩に対する訓練のない者、らばきっとこの世界のみに見られることがある。もっとも、このであろうと思われる。ヘストックを差し込んで穴を作ると、うな色を呈すること位は誰にも見らう。鮮かな色彩は札幌附近の雪にも、多分積雪の中にも結晶がかなり完全に結晶の面が沢山あるためによるもの

図13.2: 縦書き文書の出力例

第 14 章

製本方法の詳細

冒頭のチュートリアルで行った製本（ビルド）の仕組みをもう少し詳しく解説する。

`bookdown-demo` を念頭に置いた解説。`rmdja` も基本的に同じ。

- `index.Rmd`: デフォルトで最初に読み込まれる Rmd ファイル（名前を変える機能もあるが、現時点では不具合が起こりやすいのでおすすめしない）
- それ以外の Rmd ファイル: 連結して読み込むことが可能
- `_output.yml`: マルチメディア展開のための設定。PDF, HTML, EPUB それぞれの設定を書く
- `_bookdown.yml`: bookdown のレイアウト設定
- その他の設定ファイル: その他製本に必要なもの、画像ファイル、.css ファイル、.bib 等

`_output.yaml`, `_bookdown.yml` は `index.Rmd` のヘッダに書くこともできるが、長くなりすぎるので分割できる。`bookdown::render_book()` 関数は、ルートディレクトリのこれらを自動で読み込んでくれる。

14.1 ファイル構成

これらのファイルの中身を解説する。

14.1.1 `_output.yml`

本来の YAML の `output:` 以下の記述をこの `_output.yml` ファイルに書くことができる。`output:` を複数書くと `rmarkdown::render_site()` やビルドツールでそれぞれの形式に一括作成してくれる。

```
output:
  bookdown::gitbook:
```

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

詳しくは BKD “Ch. 3 Output Formats” の章を.

14.1.2 _bookdown.yml

_bookdown.yml も index.Rmd の YAML ヘッダの bookdown: 以下に対応する内容を書くことができる. 例えばどの Rmd ファイルを読み込むかとか, LaTeX のときだけ, HTML のときだけ読み込むような設定も可能.

<https://ill-identified.hatenablog.com/entry/2020/09/05/202403>

詳しくは, BKD Ch. 4.4 Configuration

Build ページから文書をビルドするには, index.Rmd の YAML ヘッダに site: bookdown::bookdown_site を書く必要がある. さらに, index.Rmd をプロジェクトディレクトリのルートに置いていない場合は, ツールバーの Build -> Configure Build Tools... から index.Rmd を置いているディレクトリを site ディレクトリとする設定が必要になる(図 14.1, 14.2).

または, bookdown::render_book("index.Rmd", "rmdja::pdf_book_ja") などでも実行できるから, コマンドラインからも実行できる. 同時製本は rmarkdown::render_site().

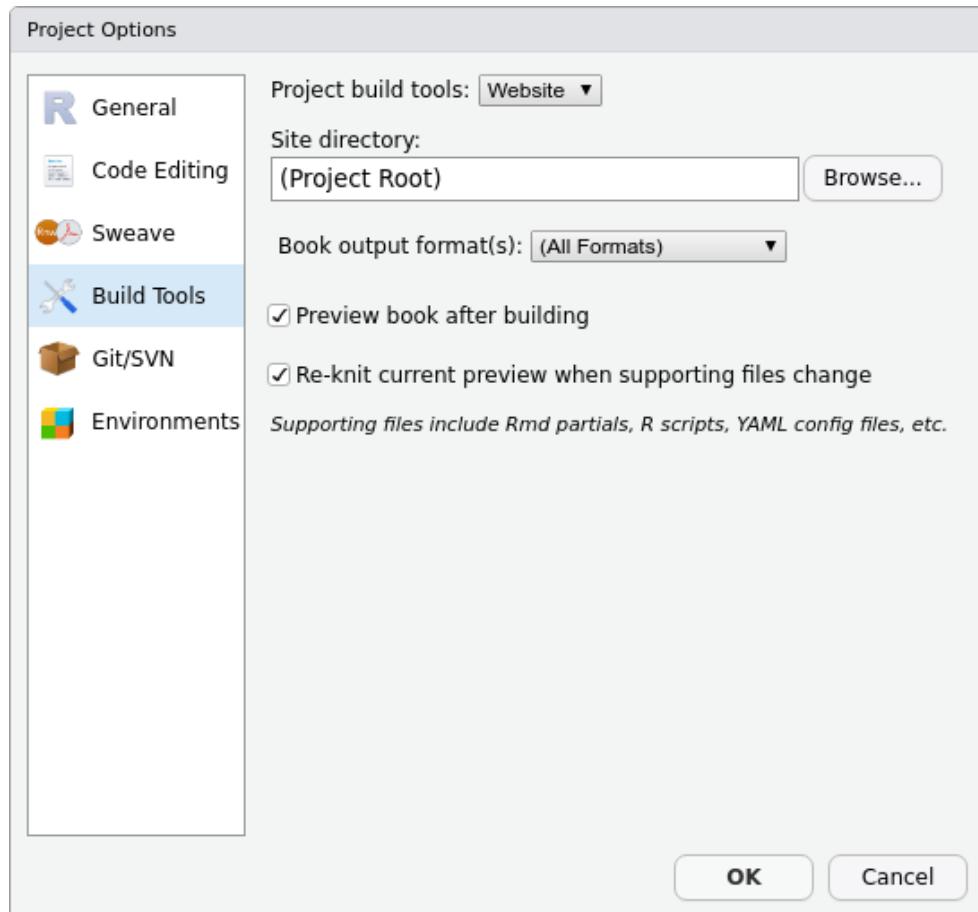


図14.1: Build ペーンの手動設定

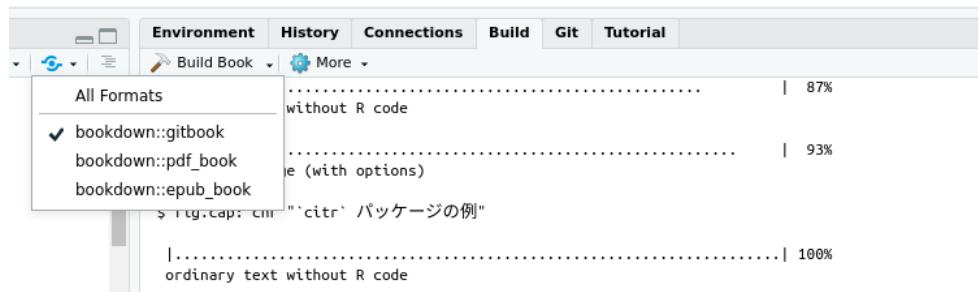


図14.2: Build ペーンの手動設定

第 15 章

出力形式による表現の限界

15.1 HTML と PDF で処理を場合分けする

出力方法で言えば、HTML と PDF に大別できる。Rmd は HTML タグも LaTeX コマンドも受け付けるが、それぞれ HTML と PDF に変換する際にしか反映できない。よって、例えば複雑な図表を LaTeX コマンドでじかに Rmd ファイルに書いてしまった場合、HTML では表示されない。

紙媒体と電子媒体では表現できることに差がある。例えば紙はあらゆる環境で同じような見た目になるが、ハイパーリンクは付けられないし、一度出版してしまうと修正は容易ではない。PDF の見た目も読者の環境に依存しにくいが、やはり更新が容易ではない。

`bookdown` には既に印刷された本の中身を書き換えるする機能はないが、出力ごとに内容を変えることで、PDF にのみ更新履歴を表示することはできる。

`knitr::is_latex_output()`, `knitr::is_html_output()` などは、`knit` 時にどの媒体への変換処理なのかを判定するのに使える。`rmdja::ruby()` もこの機能を利用しているし、本文中の `LATEX` のロゴも HTML と PDF で使い分けている。

また、`_bookdown.yml` の設定、`rmd_files` は、媒体別に設定することができる。

```
rmd_files:
  html:
    - index.Rmd
    - html-only.Rmd
  latex:
    - index.Rmd
    - latex-only.Rmd
```

15.2 絵文字の出力

絵文字を HTML でも PDF でも出力したい場合, `\coloremoji{[]}` のように絵文字を囲む。ただし, RStudio のエディタは一部のマルチバイト文字の表示に対応していないので予期せぬ不具合に注意する。

現在の主要 Web ブラウザでは, 特に設定せずとも Unicode 絵文字をカラー画像に置き換えて表示できるものが多い。しかし PDF 生成時には明示的にフォントを指定するか, 画像に置き換える記述が必要である。その実現のため `bxcoloremoji` という LaTeX パッケージ^{*1}を利用する。このパッケージは CTAN に登録されていないため, 別途インストールする必要がある。

15.3 画像の保存形式

技術文書での画像の多くはプロットなど単純な図形なので, 写真などを掲載するのでない限り, PDF で出力する場合はプロット画像も PDF にするのが望ましい。JPG や PNG などのラスタ画像では拡大すると粗くなるが, PDF などのベクタ画像ならば拡大しても粗くならず, かつ単純な図形ならばファイルサイズも小さく済むことが多い。一方で HTML は通常 Web ブラウザで閲覧するため, PDF に対応していないことが多い。HTML でベクタ画像を掲載したい場合は **SVG 形式**で出力する。

R による SVG への出力は, 従来組み込みの `svg()` で行うことが多かったが, 近年は新たなパッケージが出ている。有力なのは `svglite` と `rsvg` である。

<https://oku.edu.mie-u.ac.jp/~okumura/stat/svg.html>

`rsvg` のほうが高性能だが, `knitr` で対応しているのは `svglite` なので簡単に使いたいならこちらを推奨する。

15.4 デフォルトの保存形式

デフォルトでは, PDF は `cairo_pdf`, HTML では解像度を高めに設定した `PNG` を使用している。これは, 件数の多い散布図など, ベクタ形式ではファイルサイズが大きくなりすぎる場合もありうるための判断である。

画像形式を変更したい場合は, チャンクオプションの `dev` で, オプションは `dev.args=list(...)` で変更できる。

<https://bookdown.org/yihui/rmarkdown-cookbook/graphical-device.html>

^{*1} <https://github.com/zr-tex8r/BXcoloremoji>

第 16 章

製本した文書を配布する

16.1 Wep ページのホスティング

HTML ファイルは様々な配布方法がある。もちろん自分でサーバを立てても良い。特に簡単なのは以下の 2 点である。

1. github pages を利用する
2. bookdown.org に投稿する

(1) の詳細は [github.com の公式ドキュメント](#)を見るのが一番良いだろう。`rmdja` では、`bookdown` の機能である `_bookdown.yml` に文書ファイルの出力場所を指定するオプションをそのまま使えるため、`docs` ディレクトリに出力するよう設定すればあとはリモートリポジトリにプッシュし、`pages` を公開するよう設定するだけである。

既に `bookdown` で作成した文書を公開している例は多数ある。例えば既に何度も言及した公式解説サイトはそれじたいが `bookdown` で作られているし、“R for Data Science” ([Wickham and Grolemund, 2016](#)) *¹は、内容の良さも含め一見に値する。また、“Hands-On Data Visualization: Interactive Storytelling from Spreadsheets to Code” ([Dougherty and Ilyankou, forthcoming](#)) という本*²が来年出るらしい。そして面白いことにこれは R の本ではなく Google スプレッドシートとかの Web 上のサービスの利用法を紹介する文書である。

これらはいずれもソースコードまで公開されている。もちろんここでいうソースコードとは、本文中のプログラムだけでなく文書を生成する `Rmd` ファイルなども含める。

それ以外にも有名無名の多くのドキュメントが公開されているが、一方で日本語はまだまだ少ない。内容が豊富で、かつ `Rmd` のソースコードまで公開されている例として以下が見つかった。

- 『R で計量政治学入門*³』

*¹ <https://github.com/hadley/r4ds>

*² ソース: <https://github.com/handsondataviz/book>

*³ ソース: https://github.com/shohei-doi/quant_polisci

- ・『A レベルの倫理学^{*4}』

さらに以下 2 つは私が作成したものである。

- ・『三國志で学ぶデータ分析 (Japan.R 2019)』^{*5} (Japan.R 2019 の資料)
- ・『経済学と反事実分析接触篇 Economics and Counterfactual Analysis: A Contact』^{*6} (Tokyo.R 第 83 回の資料)

特に私の 2 作品は PDF のレイアウトにも注意を払っているが、当時はまだ rmdja を作成しておらず kazutan 氏作の `bookdown_ja_template` をさらに改良した kenjimyzk 氏の `テンプレート` を元にワンオフで作成したフォーマットを使用しているためあまりスマートでない書き方が見られる。

また、HTML 形式の文書には PDF など他のファイル形式のダウンロードリンクを設置することができる。これは `_bookdown.yml` で表示を指定できる。

16.2 (WIP) 入稿するには

国内の印刷所で PDF 入稿する際のスタンダードは何だろうか？ 紙媒体でやったことがないので全くわからない。ver. 0.3 時点での対応を紹介する。なおこれらの対応は、『Bookdown による技術系同人誌執筆』というブログ記事で言及されていた、入稿に必要な PDF のスタイルだそうである（印刷所ごとに対応は違うと思われる）。

また、現在は `Re:View` という電子・紙媒体書籍作成ソフトがあり、既に出版での使用実績も増えつつあるらしい。R Markdown、ひいては rmdja のモチベーションの 1 つは、R コードを文書に埋め込むことで結果の再現性を確保する、というものである。再現性というと抽象的だが、冒頭に書いたようにコードの埋め込みによってグラフや計算結果の修正差し替え時のミスが減ることはここまで読んだ方は納得いただけるのではなかろうか。しかし、文書に R コードの埋め込みが必要なく、単純に DTP ツールとしての完成度を評価するのなら、`Re:View` のほうが確実に信頼できる。

16.2.1 トンボの表示

`_output.yml` で

```
rmdja::pdf_book_ja:  
  tombow:true
```

^{*4} ソース: <https://github.com/MToyokura/Ethics-for-A-Level-Japanese>

^{*5} ソース: <https://github.com/Gedevan-Aleksizde/Japan.R2019>

^{*6} ソース: https://github.com/Gedevan-Aleksizde/20190125_tokyor

とすると PDF にトンボ (trimming mark) を表示する。これは `gentombow.sty` によるものである。しかし私はこの出力が適切なのか判断することができない。

16.2.2 (TODO) 隠しノンブルとヘッダ・フッタのカスタマイズ

対応中

16.2.3 フォントの埋め込み

少なくとも PDF ではフォントを埋め込みそこなったり、Type 3 フォントが設定されないようにしている。ただし Python 等を利用して描いたグラフを埋め込む場合、個別に設定が必要な場合もあり、完全な保証はできない。

TODO: <https://teastat.blogspot.com/2019/01/bookdown.html> の記述のうち、まだ対応していないものがある。

第 VI 部

デバッグとトラブルシュート

このパートで説明すること

残念ながら、現状 `bookdown` は完全にプログラミング知識のないエンドユーザでも縦横無尽に使用できるかと言うと、まだまだ不安定でそのレベルには達していない。さらに悪いことに、`rmarkdown` および `bookdown` は `knitr`, `pandoc`, `LaTeX` といった様々なプログラムを継ぎ接ぎして実装されているため、R の知識だけではエラーが起った場合や、意図したとおりの出力が得られないときに原因が分かりにくいことがある。そこで、ここではエラーが出た際にどう対処するかのヒントを書いておく。

第 17 章

製本時のエラーへの対処

17.1 エラーがどのタイミングで発生したかを特定する

R Markdown はさまざまな外部プログラムを利用して、数段階のプロセスを経てソースファイルを変換して文書を作成する複雑なプログラムである。逆に言えば、Rmd ファイルを md ファイルに変換 (knitr による処理) するときにエラーが出たのか (= R のプログラムにミスがある可能性が大), md を各ファイルに変換 pandoc する際に起こったのか (= 経験上ほとんどは tex ファイルのコンパイルエラーによるもの) をまず特定するのが重要である。そのためには

1. `keep_md: true / keep_tex: true` を設定する
2. うまくいかないときはキャッシュを削除してから再実行する

という対処法がある。(1) は文字通り中間出力ファイルである .md および .tex を残すことを意味する (tex ファイルの保全はデフォルトで `true` 設定になっている)。これが生成されないなら knitr でのエラーだと分かるし、中身を見て不自然な内容になっているのなら Rmd の書き方が knitr に正しく評価されていないことがわかる。

キャッシュも私の経験上よくエラーの原因となっている。以前に実行していたチャンクの結果が更新されていないせいで、knitr の処理の不整合を起こすことがある。`*_files` には出力に必要な画像ファイルが、`*_cache` にはチャンク実行結果のキャッシュが残っている。後者は `knitr::opts_chunk$set(cache = T)` などでキャッシュを残す設定にできるので、F に設定した上でこれらのファイルを削除する。

処理に時間がかかるチャンクがあってキャッシュを作りたい場合は、別途 rds や RData ファイルに結果を保存するという方法もある。しかしもしプログラムの再現性を重視する場合、この方法は望ましくないだろう。しかし残念ながら現状はこうするか、ひたすら長い時間を待つしかない。

TODO: <https://bookdown.org/yihui/rmarkdown-cookbook/cache.html>

17.2 YAML フロントマターを確認する

以前『[R] R Markdown の YAML ヘッダでハマったおまえのための記事』というブログ記事にも書いたように、YAML フロントマターは慣れない書き間違えやすいのが現状である。もし自分で変更したのなら、改めて確認すべきだろう。特に、製本直後にすぐに、心当たりのない R プログラム関係のエラーが出る場合、チャンクではなく YAML フロントマターの読み取りに失敗している可能性がある。

以下の 4 原則を覚えておこう。以前は bookdown の話を想定してなかったので、さらに条文を 1 つ加えた。

1. `output:` 以下はフォーマット関数への引数
2. トップレベルのオプションは `pandoc` のオプション
3. タイプミスや位置間違えでも動いたり、動かなかったりする
4. `_output.yml` および `_bookdown.yml` を見る。

`output:` には `bookdown::gitbook` など、`bookdown` で提供されているフォーマット関数を指定しており、その配下に記入するのはフォーマット関数に与える引数である。よって、関数ヘルプを確認すれば有効な引数を知ることができる。しかし一方で、`...` が引数になっていることがあるので、タイプミスしてもエラーが出ないことがある。

また、YAML の構文でサポートされている配列は誤評価を引き起こすことがある。

```
output:
  bookdown::gitbook:
    toc_depth: 3
    toc: true
```

```
output:
  bookdown::gitbook:
    - toc_depth: 3
    - toc: true
```

上の例は正しい記法である。一方でハイフン `-` は YAML では配列を記述するために用意されている。下記の場合、キーワード引数ではなく位置引数のような扱いになるため、`toc` に対して `3` を代入することになり、エラーが発生する。逆に言えば、`-` を使う場合、キーワードを書かずに値だけを正しい順番で書けば機能する。

インデントしないトップレベルの引数は、基本的に `pandoc` に与える引数である。これ意味のない引数を与えてもエラーを返さないことが多いので、タイプミスに注意する。

しかし、フォーマット関数に `pandoc_args` という構文をサポートしていることや、フォーマット関数で `pandoc` の同名の引数を上書きする仕様のフォーマットもあるため、上記は絶対ではない。これが原因で、「output: 以下に書くべきものを間違えてトップレベルに書いたが、意図したとおりに機能した」あるいはその逆が発生することがある。また、`pandoc` の構文ではキーワードにハイフンを使うことができるが、フォーマットは R の関数でもあるためハイフンを使えず、アンダースコアで置き換えられる。この違いも書き間違えの原因になる。

17.3 PDF 生成時のエラーを確認する

それでもエラーが出る場合、私の経験上ほとんどが生成した `.tex` ファイルをタイプセットする際にエラーが発生している。`html` との両立を考えると、どうしても `pandoc` が解釈できる構文に限界があるためである。

! LaTeX Error: XXXXX

とか

Error: LaTeX failed to XXXX

といったメッセージが表示されるのですぐ分かる。さらに丁寧なことに、`tinytex` のデバッグ方法への[リンク](#)まで表示される

この場合最も重要なのは、以下に尽きる。

1. `options(tinytex.verbose = TRUE)` を設定する
2. `keep_tex: true` を設定する

これは `keep_md` と同様に、中間ファイルである `.tex` を残すことを意味する。

それでも解決しない場合、改めてこのファイルを手動でタイプセットするのも 1 つの方法だ。もしうまくいったり、異なるエラーが出るのなら、環境の違いが問題かもしれない。そして `upBIBTeX` を使うのなら、後者が唯一のデバッグ方法だ。

17.4 よくあるエラーメッセージ

17.4.1 The File XXX.Rmd Exists.

The file _main.Rmd exists. Please delete it if it was automatically generated. If you are sure it can be safely overwritten or do

多くの場合はファイル名が `_main.Rmd` となるだろう。つまり最終的に出力する PDF と同じ名前である。これは `_bookdown.yml` の `book_filename` で変更することができる。このエラーは文字通り `_main.Rmd` ファイルが既に存在するから処理を続行できない、というものである。製本時に `index.Rmd` と同じフォルダに、中間生成物である `_main.Rmd` が作られるが、前回の製本処理が何らかの理由でエラーが発生し中断しているとこのファイルが残ることがある。よってこのファイルを削除すれば解決する。

17.4.2 No Site Generator Found.

製本処理にあたって、基準となるフォルダの設定が見つけられない際に発生する。`index.Rmd` に `site: bookdown::bookdown_site` が記述されていないか、ビルドペーンでの設定でフォルダを正しく設定できておらず、`index.Rmd` の存在しないフォルダを参照していることがよくある原因である。

第 18 章

その他のトラブルシュート

18.1 コードブロックや出力テキストの折り返し・改行位置 がおかしい

折り返し位置を規定するグローバルオプション `getOptions(width)` を確認する。通常は 80 かそれより大きい値が設定されていることが多いが、何らかの理由で小さく設定されている可能性もある。確実を期すなら、冒頭のチャunkに `options(width = 140)` のように明示的に設定する。

また、コードブロックの折り返しや改行位置がおかしい場合、それはコード自動整形の問題である可能性がある。`rmdja` はスライド用テンプレート以外でデフォルトでコードブロックの自動整形を適用しているが、`black` や `yapf` のある Python などと違い、R のコード自動整形ツールは機能やバリエーションがあまり多くない。コードの整形にこだわるなら、ある程度は手動でやる必要がある。コードの自動整形を無効にするなら、冒頭のチャunkで以下を実行する。

```
01 knitr::opts_chunk$set(tidy = F)
```

コードの自動整形の詳細については付録 [A.3](#) を参照。

18.2 (TODO) Windows 特有の問題

日本ロケールの Windows OS で RStudio を動かす場合によくあるエラーについても対処法を書いておく。これは R-wakalang でもよく訊かれる質問である。これらは Windows の仕様が根本的にアレなことに起因するため、Linux 等の仮想環境上で R を動かせば一切発生しない問題ではあるが、おそらく初心者の多くがハマっているので仮想環境を使わない解決方法を書いておく。

まず、チャunk等のエラーメッセージが文字化けして読めない。これはロケールの問題であることが多い。残念ながら日本語版 Windows は未だに CP932 エンコードを使用して

いるため, CP932 を使うと R の表示で不整合が発生する. よって, CP932 を使用すれば解決できる.

```
01 Sys.setlocale(locale = "Japanese_Japan.932")
```

しかし R の他の部分の多くは UTF-8 を前提として作られているので今度はそちらでいろいろな対処が必要になってしまう. もしこのような「仕様」が気に入らないのなら結局のところ AWS 等のクラウドサービスや仮想環境マシンを使い Linux 系の環境に移行してしまうのが確実である（もちろん Linux でも日本語ロケールの初期設定は必要である）.

付録 A

デフォルト値の自動調整

これはユーザーが通常気にする必要のないような `rmdja` 内部での処理を解説する。`knitr` や `rmarkdown` の仕様に精通している、自分で細かい設定をしたいユーザ向けの解説である。既に R Markdown に慣れていて、かなりトリッキーな使い方をしていたらどうも `rmdja` の機能とは競合するようだ、という場合は参考にしてほしい。

`rmdja` では R Markdown で日本語文書を作成する上で大きな障害の 1 つである、 YAML フロントマターの設定を改善している。`rmdja` の文書フォーマットは YAML フロントマターのデフォルト値などを日本語文書に適したものに変更している。さらに、ユーザーを OS ごとのフォントの違いや煩雑で重複だらけの設定から解放するため、内部処理でも動的に設定変更している。もちろんこれらはユーザーによる YAML フロントマターやチャンクオプションの変更で上書きできる。

A.1 デフォルトのフォント

PDF 出力時のデフォルトフォントは、生成時に OS を判定して設定している。その設定は表 A.1 のようなルールである。

表A.1: OS/エンジン別のデフォルトフォント

engine	Linux	Mac	Windows (>= 8)	Windows (それ以前)
XeLaTeX	Noto	游書体	游書体	MS フォント
LuaLaTeX	Noto	ヒラギノ	游書体	MS フォント

これらは X_ELaTeX ならば `zxjafont`, LuaLaTeX ならば `luatex-jp` で用意されているプリセットを使って設定している。使用 OS の判定は R の基本関数による。なお、Noto フォントを選んだのは Ubuntu 18 以降の日本語用フォントだからである。Ubuntu から派生した OS にはプリインストールされていることが多いようだが、Debian, Cent OS, Fedora 等にはおそらくプリインストールされていないので注意。現時点ではフォントが実際にインストールされているかを確認する機能はない。

フォントのプリセットを指定した場合、個別設定は無効になる。さらに、3種類の和文フォントを全て設定していない場合もデフォルトのプリセットから選ばれる。

A.2 チャンクオプションのデフォルト設定

チャンクオプションのデフォルト設定も R Markdown から多少変更している。

`block`, `block2`, `asis`などのブロックを `echo=F` や `include=F` にするメリットはほぼないため、`knitr::opts_chunk$set(echo = F, include = F)` と一括設定してもこれらは `echo=T`, `include=T` のままである。変更したい場合は、チャンクごとに設定することで有効になる。

デフォルトの R グラフィックデバイスは、HTML では "PNG", PDF では "cairo_pdf" としている。"cairo_pdf" を使う理由は (1) R でよく描画するような単純な図形はベクタ画像が適しているが、件数のとても多いデータの散布図などはベクタ画像にするとファイルサイズが大きくなるため、そのような画像を適度に「劣化」させてファイルサイズを軽減してくれる、(2) 日本語フォントの表示と埋め込みの設定が最も簡単、というものである。そして HTML はそもそもデフォルトの設定で PDF が表示できない Web ブラウザが多いことから、PNG をデフォルトにした。もし HTML でもベクタ画像を表示したいのなら "svglite" を設定して SVG 形式にすると良いだろう。

```
01 knitr::opts_chunk(
02   dev = if (knitr:::is_latex_output()) "cairo_pdf" else "svglite"
03 )
```

ただし、R 以外のプログラムで出力した画像には `cairo_pdf` は使えないため、内部では `pdf` を使用している。これらの画像が日本語フォントを適切に埋め込めるかはそれぞれの設定に依存するため、R 側で制御するのは難しい。

A.3 コードブロックの整形と自動折り返し

HTML はともかく、PDF はコードの自動折り返しが難しい。例えば RCB Ch. 5.3 では、`listings.sty` を使う方法が書かれているが、この方法ではデフォルトのシンタックスハイライトが使えなくなり、R Markdown の大きなメリットの 1つが損なわれてしまう。また、同 Ch. 11.10 では `knitr` のチャンクオプションで `tidy` と `tidy.opts` を設定するという方法が紹介されている。この機能は `formatR::tidy_source()` 関数を利用したコード整形であり、この関数の `width.cutoff` というオプションで自動折り返しを始める位置を指定できる。(たまに勘違いしている人がいるが、ドキュメントをちゃんと読めば分かるように) このようにコード整形機能は自動折り返しを目的としたものではないため、長すぎる関数名や文字列があると `width.cutoff` を超過することも十分ありえる。同章では `styler` パッケージがより機能が豊富だと言及しているが、このパッケージも現時点では 1 行の上

限を指定する機能はない^{*1}. rmdja ではデフォルトで styler を使ったコード整形をするとともに, フォーマット beamer_presentation_ja と pdf_book_ja にコードブロックの自動折り返しを有効にする code_softwrap というオプションを用意した. 前者ではデフォルトで false, 後者では true である.

しかし, これらを使っても「きれいな」コーディングになるとは限らない. 過剰な折り返しで行数が増えてしまう可能性もあるし, 折り返しや改行の位置がふぞろいになる可能性もある. また, トークン単体で非常に長い場合 (たとえば 100 字分の文字列) も, 途中で折り返すことはできない. よって現状では究極的には手動で調整する必要がある

その際のアシストツールとして, RStudio の機能であるマージン位置の表示^{*2} や, WrapRmd パッケージを使うのが良いだろう.

逆に自動コード整形が一切不要という場合, 最初のチャunkで以下のように設定する.

また, 自動折り返しの発生した箇所にはデフォルトでキャリッジターンの記号が表示される. これが不要である場合, 例えば

```
\usepackage{fvextra}
\DefineVerbatimEnvironment{Highlighting}{Verbatim}{commandchars=\{\}}
\begin{Highlighting}[breaklines,breakanywhere,breaksymbolleft={},breaksymbolseleft=0pt,breaksymbolindentleft=0pt]
```

という LATEX コードを includes または header-includes を経由して与える. より細かい設定は fvextra のドキュメントを参照してほしい.

PDF での自動コード整形に関する話題は R Markdown の Issues #646 および Stack Overflow の質問 “pandoc doesn't text-wrap code blocks when converting to pdf” と TeX Stack Exchange の質問 [“Break Lines in minted environment”] (<https://tex.stackexchange.com/questions/200310/break-lines-in-minted-environment>) が参考になるだろう.

^{*1} 参考: Issues #247

^{*2} 参考: Vertical Line in the source editor?

付録 B

PDF の組版に関する細かい話

ここでは pandoc テンプレート等の設定を解説する。PDF の出力は pandoc に大きく依存している。pandoc は PDF を生成する際に、YAML フロントマターの設定を pandoc のテンプレートに代入し、本文を追加した.tex ファイルを作成してタイプセットしている。よってプリアンブル部分は完全に動的に生成されるわけではなく、ある程度の定型が存在する。これを pandoc テンプレートというが、rmdja では日本語表示にあたっていろいろなパッケージ間の競合が見られたこのテンプレートを多少いじっている。

3種類の和文フォントを個別設定をした場合、 \rm XeLaTeX はフォールバックフォントを有効にしている。`j****fontoptions` 以下に、`FallBack=...` というオプションでフォールバックフォントを指定すれば有効になる。

用紙サイズは、デフォルトは `a4paper`, `B5` がよいなら `b5paper` オプションを `classoptions:` に指定する。

PDF を印刷所に持ち込んだことがないため詳しいことはわからないが、『[Bookdownによる技術系同人誌執筆](#)』で指摘されているようなトンボやノンブルは出力されるように作ってある（そしてここで紹介されているような LaTeX のコマンドの多くは rmdja では書く必要がなくなった）。

TODO: PART の扉ページにはまだノンブルが表示されない

B.1 画像の配置

現在、PDF で画像の配置を固定する方法について何も特別なものを用意していない。単純に自分は必要だとおもったことがないため。固定したい場合は R Markdown や Bookdown のドキュメントを参考にしてほしい。ただし、通常は章や部をまたいで表示されることはない（はず）。

B.2 取り消し線

LaTeX の各パッケージのバージョンによっては、和文に取り消し線 (`\sout`) を与えるとタイプセット時にエラーが出ることがある。もともと `ulem.sty` は欧文を前提にしたもの

なので適当に妥協してほしい。

B.3 TODO: しかし英文で書きたい場合

`rmdja` の機能を使いたいが、執筆は英語でしたいと言う場合は最低限以下のようない定変更が必要である。デフォルトは日本語用文書クラスのため、

`documentclass: book / report / article`

など欧文用文書クラスを指定する。

`rmdja` では和文フォントを参照するので、和文フォントの設定も手動で解除する必要がある。

を指定する。そして各種見出しも英文用に調整する。

TODO

付録 C

参考文献リストの書式にこだわる: jecon.bst

0.4.3 以降の `rmdj` は `biblatex` がデフォルトであり、でいちおう和文献のリストが最低限のクオリティで表示できるようになっている。しかし自分で言うのもなんだが、だいぶ稚拙な出来栄えである。そこで和文と欧文を使い分けたスタイルファイルとして、`jecon.bst` を紹介する。`jecon.bst` の公式ではなく、私がカスタマイズしたバージョンでも良い。こちらは本来よりも電子媒体としての利用を重視して、

- 参照 URL を表示せず、ハイパーリンクのみにする
- ArXiv ID の表示とハイパーリンク追加

といった変更をしている。後者は、BibTeX エントリに以下のように `archivePrefix` に `arxiv` と言う値が入っていると、`eprint` の値が ArXiv ID として表示される。これは ArXiv から直接 `.bib` ファイルを取得したり、Zotero などでインポートすれば必ず入力される項目である。

```
archivePrefix = {arXiv},
eprint = {XXXX.XXXXXX},
...
```

このスタイルの使用には `upBIBTEX` が必要である。詳細は 10 章を参照されたい。

TODO: 現在 `jecon.bst` の表示も少しおかしいので確認中。

付録 D

fontregisterer でグラフ用フォントを自動登録

R で描いたグラフに日本語を表示する場合、Linux 系 OS ならばフォント名を適切に設定するだけで表示されるが、Windows や Mac ではフォントをグラフィックデバイスに登録する必要がある。しかし手動登録は面倒なので、インストールされているシステムフォントを全て自動登録するパッケージ、`fontregisterer` を用意している。

```
01 remotes::install_github(  
02   "Gedevan-Aleksizde/fontregisterer", repos = NULL)
```

もちろんこれは R Markdown 以外でも使用できる。このパッケージは読み込まれた時点で登録処理を行うため、

```
01 require(fontregisterer)
```

を最初に実行するだけで良い。詳しい仕組みの解説は『[おまえはもう R のグラフの日本語表示に悩まない \(各 OS 対応\)](#)』に書いている。

参考文献

- Dougherty, Jack and Ilya Ilyankou (forthcoming) “Hands-On Data Visualization Interactive Storytelling from Spreadsheets to Code,” URL: <https://handsondataviz.org/>.
- Wickham, Hadley and Garrett Grolemund (2016) *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*, Sebastopol, CA: O'Reilly, first edition edition, URL: <https://r4ds.had.co.nz/>, (黒川利明・大橋真也訳, 『Rで始めるデータサイエンス』, オライリー・ジャパン, 2017年,) .
- Xie, Yihui (2020) *Bookdown: Authoring Books and Technical Documents with r Markdown*: Chapman & Hall, URL: <https://bookdown.org/yihui/bookdown/>.
- Xie, Yihui, J.J. Allaire, and Garrett Grolemund (2018) *R Markdown: The Definitive Guide*, Boca Raton, Florida: Chapman and Hall/CRC, URL: <https://bookdown.org/yihui/rmarkdown>.
- Xie, Yihui, Christophe Dervieux, and Emily Riederer (2020) *R Markdown Cookbook*, S.l.: CRC PRESS, URL: <https://bookdown.org/yihui/rmarkdown-cookbook/>.