

Rによる作業の自動化レシピ集 -パッケージ活用術-

松村 俊和

2024-07-05

Table of Contents

はじめに	8
0.1 対象とする読者	8
0.2 本書の読み方	9
0.3 R とパッケージのインストール	9
0.4 コードの公開と利用	10
0.5 リンク集(未整理)	11
1 ファイルとディレクトリの操作	14
1.1 ファイルとディレクトリを操作するパッケージ	14
1.2 各種パスの取得と設定	15
1.3 ディレクトリとファイルの生成	17
1.4 ディレクトリとファイル一覧の取得	17
1.5 拡張子の変更・取得・除去	22
1.6 ファイルの移動・コピー・削除	23
1.7 ファイル名の変更	25
1.8 マウスでのファイルやディレクトリの指定	28
2 データの操作	30
2.1 データを操作するパッケージ	30
2.2 集計用のデータ	32
2.3 <code>tibble</code> の生成	34
2.4 整然データへの整形	36
2.5 データフレームの操作	40
2.6 データのグループ化と集計	46

2.7	洗練された作図	51
2.8	繰り返し処理	61
2.9	順次処理	65
3	文字列の操作の自動化	67
3.1	文字列を操作するパッケージ	67
3.2	結合	68
3.3	マッチ箇所の明示	69
3.4	正規表現	69
3.5	置換・削除	73
3.6	検索・抽出	74
3.7	その他の関数	77
3.8	文章の比較	80
4	日付データの操作	83
4.1	日付データを扱うパッケージ	83
4.2	日付クラスへの変換	84
4.3	1月後・1年後の同一日	85
4.4	曜日の取得	85
4.5	日付っぽい文字列	86
4.6	和暦への対応	90
4.7	年の追加	92
4.8	日付っぽい文字列の抽出	94
4.9	西暦と和暦の相互変換	95
4.10	曜日の取り出し	98
4.11	日付と曜日との整合性の確認	99
4.12	1年後の同一位置への更新	102
5	コマンドの実行	106
5.1	アプリケーションの瞬間起動	106
5.2	関連付けアプリでの起動	110

5.3	R のバッチモードでの実行	1 1 1
5.4	OS コマンドの実行	1 1 4
5.5	アプリの終了	1 1 6
5.6	zip ファイルの解凍	1 1 6
6	キーボードとマウスの操作	1 2 0
6.1	キーボードとマウスを操作するパッケージ	1 2 0
6.2	キーボードの操作・文字入力	1 2 0
6.3	マウスの位置取得	1 2 3
6.4	マウスの移動	1 2 4
6.5	マウスのクリック	1 2 4
6.6	画像の位置特定によるマウスの移動・クリック	1 2 5
6.7	USB メモリの取り出しの自動化	1 2 7
7	PDF の操作	1 3 1
7.1	PDF を操作するパッケージ	1 3 1
7.2	作業用の PDF ファイルの用意	1 3 2
7.3	ページ数の取得	1 3 2
7.4	ページの分割・抽出	1 3 3
7.5	ページの結合	1 3 6
7.6	ページの回転	1 3 7
7.7	圧縮・最適化	1 3 8
7.8	文字列の抽出	1 3 8
7.9	画像ファイルへの変換	1 3 9
7.10	PDF の文字認識	1 4 0
7.11	画像の抽出	1 4 2
7.12	重ね合わせ	1 4 3
8	ワードの操作	1 4 7
8.1	ワードを操作するパッケージ	1 4 7
8.2	ワードの読み込み	1 4 8

8.3	文書の書き出し	150
8.4	概要の表示と内容の取り出し	150
8.5	文字列の一括置換	153
8.6	文字の書式変更	155
8.7	ページ設定	155
8.8	文書の相互変換	156
8.9	文書の作成	158
8.10	図表の追加	161
8.11	コメントの追加と抽出	162
8.12	画像の抽出	165
8.13	文書内の日付の修正と更新	168
9	エクセルの操作	170
9.1	エクセルを操作するパッケージ	170
9.2	ファイルの読み込み	171
9.3	ウェブからのデータの読み込み	173
9.4	ファイルの書き込み	174
9.5	ピボットテーブル	175
9.6	列幅の設定	179
9.7	オートフィルタの設定	181
9.8	ウィンドウ枠の固定	183
9.9	罫線	184
9.10	条件付き書式設定による強調表示	191
9.11	ヘッダー・フッター	194
9.12	ページ設定	196
9.13	改ページの設定	197
9.14	PDFへの変換	201
10	パワーポイントの操作	203
10.1	パワーポイントを操作するパッケージ	203

10.2	ファイルの新規作成・読み込み	203
10.3	レイアウトの確認	204
10.4	スライドの追加	205
10.5	文字列の追加	206
10.6	箇条書きの追加	207
10.7	表の追加	209
10.8	画像の追加	211
10.9	ggplot のグラフの追加	217
10.10	概要の表示	218
10.11	文字列の取り出し	219
10.12	表の取り出し	220
10.13	画像ファイルの取り出し	221
10.14	画像・PDF・動画への変換	223
II	画像の操作	228
II.1	画像を操作するパッケージ	228
II.2	作業用の画像ファイルの用意	229
II.3	読み込み	229
II.4	画像の描画	230
II.5	情報の表示	232
II.6	形式変換・書き込み	232
II.7	枠(余白)の追加	234
II.8	文字の追加	235
II.9	画像の結合	237
II.10	背景の塗りつぶし	240
II.11	余白の除去	241
II.12	サイズの変更	241
II.13	画像内の文字認識	244
II.14	傾き追加・補正	248

11.15	部分の切り取り	249
11.16	連続した指定範囲の切り取りと保存	253
11.17	アニメーション化	254
11.18	その他の画像の加工	255
11.19	画像の一覧整理	257
11.20	スクリーンショットの保存	259
11.21	クリップボード画像の自動保存	260
12	メールの操作	264
12.1	Gmail の操作の準備	264
12.2	Gmail の一覧取得	272
12.3	Gmail の新規メール作成とファイルの添付	273
12.4	Gmail でのメールの送信	274
12.5	Gmail の下書きへの保存と下書きメールの送信	274
12.6	Gmail の一斉作成と送信	275
12.7	Gmail での下書きメールの削除	278
12.8	Outlook の操作の準備	279
12.9	Outlook のメールの一覧取得	280
12.10	Outlook の新規メール作成と下書きの保存	281
12.11	Outlook のファイルの添付	282
12.12	Outlook のメールの送信	283
12.13	Outlook のメールの一斉作成と送信	283
12.14	Outlook の下書きメールの削除	285
13	翻訳作業の操作	286
13.1	DeepL の操作の準備	286
13.2	DeepL での翻訳の自動化	289
13.3	DeepL を使った文章の改善	292
13.4	DeepL の利用量の確認	293
13.5	Textra の操作の準備	293

13.6 Textra での自動翻訳.....	295
14 AI の操作	298
14.1 ChatGPT の操作の準備	298
14.2 ChatGPT でのチャット	302
14.3 ChatGPT でのコードの改善.....	303
14.4 ChatGPT のパラメータの調整.....	305
14.5 DALL-E での画像生成.....	308
14.6 Gemini の操作の準備.....	309
14.7 Gemini でのチャット	312
14.8 Gemini での画像の説明	313
14.9 Gemini でのパラメータの調整.....	317
15 ウエブの情報収集の操作(スクレイピング)	320
15.1 HTML 概説.....	320
15.2 スクレイピング時の注意	326
15.3 スクレイピングで使うパッケージ	327
15.4 rvest の基本操作.....	328
15.5 selenide の基本操作	332
15.6 CRAN のパッケージ一覧の取得.....	335
15.7 新刊情報の取得	338
15.8 フォームへの入力	344
15.9 雨雲の動きの画像を取得	346
逆引き一覧.....	350
参考文献.....	351
あとがき	352

はじめに

誰でもそうでしょうが、面倒くさい仕事はしたくありません。 というか、したくないことが面倒なのでしょう。 ちなみに、著者が行動する基準は、面倒くさいかどうかが、かなりの部分を占めています。

ニワトリとタマゴの議論は別にしても、できることなら面倒な作業はしたくありません。 しかし、必須であれば自動化したいものです。もちろん、すべての仕事を自動化できるわけではありませんし、作業内容によっては文章執筆のように自動化すべきでないこともあります。

作業の自動化には、プログラミング言語がよく使われます。 Python は比較的習得しやすい言語らしく、多くの人が使用しています。 私自身も多少は使えるものの、それよりも R の方が慣れています。 R はどちらかといえば統計解析に特化した言語として捉えられています。 実際にそうですが、統計解析以外にも多くの機能があり、自動化のためのパッケージも充実しています。 できることなら、ほぼすべての作業を R で行いたいと思っています。 そこで、本書では R を使った作業の自動化手法についてご紹介します。

R は他の言語と比べるとちょっと独特なところがあります。 個人的見解ですが、R はある意味でパクチーミたいなところがあると思っています。 また、本書では普通の人があまり使っていないような R のパッケージや機能を使っています。 初めて見る人にはとっつきにくいかもしれません。 それでも、一度使ってみると独特なところが結構クセになるかもしれません。

matutosi@gmail.com

0.1 対象とする読者

本書は、以下の人に読者として想定しています。

1. R を使ってパソコンの操作を自動化したい人
2. プログラミングの基礎知識を持っている
3. エラーが出た時にネットで調べて解決できる
4. 自己責任でコードを実行できる

1つ目は言うまでもありませんが、R を使って自動化をしたい人を対象としています。 他の言語をふだん使っている、新たに R を使ってみたいという人も、もちろん対象です。

2つ目の R の基礎知識があれば良いです。 既に仕事や研究などで R を多少使っていれば問題ありません。 本書のほとんどのコードを理解可能でしょう。 R を使ったことはなくとも、他のプログラミング言語を使っていれば大丈夫です。 この際、しっかりと R を習得したい人は、以下のうち1冊読むことをお勧めします。

- ・ R ではじめるデータサイエンス 第2版
- ・ 改訂2版 R ユーザのための RStudio[実践]入門
- ・ RStudio ではじめる R プログラミング入門

3つ目のエラーについては自分で対処する必要があります。OSの違いやその他の環境の違いにより、コードを実行したときにエラーが発生する可能性があるためです。自分のまわりにRの上級者がいれば質問できますが、そうでないときには、自分で解決しなければなりません。そのときには、実行したコードとエラーの内容をもとに検索して解決してください。できれば、日本語よりも英語で検索するのが良いです。圧倒的に情報量が違うからです。

エラーが発生して、検索しても解決しなければGitHubのissueにお送りください。可能な範囲で対応します。ただし、Windows以外のOSはほとんど使ったことがないため対処できない可能性が高いことや、Windowsであっても全てに対処できるわけではないことはご了承ください。

《Tips》 Rをインストールするとき、Message Transrationをオフにするとエラー時の対応が楽になるかもしれません。エラーメッセージが英語で表示されるので、ネットでの検索が楽です。英語が不得意な人はGoogle翻訳や本書で紹介するDeepLやTextraなどを活用してください。

4つ目の自己責任については、プログラムの基本中の基本です。コードを実行して何らかのデータが消失したり、損害を与える可能性があります。そのためコードの実行には注意とともに、自己責任で実行してください。

ファイルの上書きする際や削除の際には、細心の注意が必要でしょう。ファイルの書き込みのときには上書きするのではなく、作業ディレクトリに一旦コピーしてから作業すると良いです。削除のときは、ディレクトリ全体ではなく個別のファイル名を指定すると、「すべて消えてしまった!」という間違いを回避できる可能性が高くなります。

0.2 本書の読み方

各章の内容は独立しているので、読者の興味に合わせて好きな部分をご覧ください。各章では、基本的な関数の説明や自動化のための活用方法を紹介しています。ただし、第1部の内容は他の章でもよく使っているので、ここから読むと理解が早いでしょう。また、必要に応じて関連部分を参照してください。

0.3 Rとパッケージのインストール

Rのインストール方法や基本的な操作は説明しません。これらについては、読者自身で他の書籍などを参考にしてください。

tidyverseは本書を通じて全体的に使用しています。インストールしていない場合はインストールを、また明示していなくても呼び出してください。

```
install.packages("tidyverse") # インストール  
library(tidyverse) # 呼び出し
```

各章で使用する主なパッケージは、章の冒頭でインストール方法を示しています。パッケージのインストール時には、依存するパッケージも合わせてインストールされますが、数が多いと時間がかかります。焦らずにお待ち下さい。また、パッケージのインストール時ではなく、関数を呼び

出したときにパッケージのインストールが必要なことがあります。以下のような表示がされたら、パッケージをインストールしてください。

```
Error in library(xxxx) : there is no package called 'xxxx'
```

【要確認】

```
pkgs <-  
  c("Cairo", "KeyboardSimulator", "Microsoft365R", "ROI.plugin.glpk",  
    "bigrquery", "calendR", "chatgpt", "chromote", "curl", "deeplr",  
    "diffobj", "diffr", "excel.link", "extrafont", "fs", "gmailr",  
    "googledrive", "googlesheets4", "gptstudio", "image.binarization",  
    "magick", "magrittr", "officer", "ompr", "ompr.roi",  
    "openai", "openxlsx", "pdftools", "pivottea", "pivottabler", "polite",  
    "qpdf", "readxl", "remotes", "rsvg", "screenshot", "selenider",  
    "showimage", "tesseract", "textrar", "tidyverse", "viridis",  
    "win.binary", "zipangu")  
install.packages(pkgs)
```

0.4 コードの公開と利用

本文中にコード番号を記載しているコードはすべて GitHub にて公開しています。

<https://github.com/matutosi/r-auto/>

コード番号ごとに分かれたファイルと、章ごとにまとめたファイル、章ごとに関数のみをまとめたファイルがあります。

<https://github.com/matutosi/r-auto/tree/main/R>

コード番号 1.1 は `01_01_file-install.R`、章全体のファイルは `01_file.R`(数字の直後のアンダーバーは 2 つ)などのファイル名です。

他の章や関数のみで使用できるように、各章で定義した関数を `02_analysis_funcs.R`(数字の直後のアンダーバーは 2 つ)のようにまとめています。`source("https://matutosi.github.io/r-auto/R/02_analysis_funcs.R")` とすれば、呼び出せます。ただし、`*_funcs.R` に含んでいる関数が、既存の関数を上書きすることができますので、その点はご注意ください。

本書のコードは、ほぼそのまま実行できます。読者の環境に合わせて、入力するべき箇所は `USERNAME`, `YOUR_DIRECTORY`, `YOURNAME@gmail.com` のように大文字で表記しています。また、API キーや ID などは、`xxxxxx` のような伏せ字で示しています。

本書で使用するデータは以下で公開しています。各章で使用する際には、curl パッケージでのダウンロード方法を記載しています。

<https://github.com/matutosi/r-auto/tree/main/data>

改変しての利用や配布も自由です。ただし、利用や配布によるいかなる損害や不利益についても著者は責任をもちません。読者ご自身の責任でご利用ください。

汚いコードがあるかもしれませんご容赦ください。改善案をご教示いただければ幸いです。

万が一、バグ等があれば電子メールあるいは GitHub の issue でご連絡ください。可能な範囲で対応します。

0.4.1 パッケージ名

R の起動時に読み込まれるパッケージと各章で使用する主なパッケージの関数は、

`function_name()` のように関数名のみを示します。それ以外の関数は、

`package::function_name()` のようにパッケージ名と合わせて記載します。

本書の関数定義の中で使用する関数は `openxlsx::loadWorkbook()` のようにパッケージ名::関数名とパッケージ名を併記します。定義した関数使うときに、`library()` でパッケージを呼び出さなくても使えるようにするためです。

0.4.2 ディレクトリ構成

【補足】 本節は不要かもしれません。

本書で想定している R のインストール先などのディレクトリ構成は次のとおりです。なお、`USERNAME` はユーザ名です。

```
# Windows
c:/Program files/R/R-4.3.3/ # R のインストール先
# 以下は第 5 章で使用
C:/Users/USERNAME/shortcut # ショートカットの保存先(パスを通しておく)
C:/Users/USERNAME/RR # hoge.R などの保存先

# Mac と Linux
/usr/local/bin/ # R のインストール先

# 一時ディレクトリ
# 以下は第 5 章で使用
/Usr/USERNAME/shortcut # ショートカットの保存先(パスを通しておく)
/Usr/USERNAME/RR # hoge.R などの保存先
```

0.5 リンク集(未整理)

【報告】 とりあえず、ここに入れていますが、サポートページにリンク集を作る予定です。

サポートページ <https://github.com/matutosi/r-auto> <https://matutosi.github.io/r-auto/data/>

openai <https://openai.com/>

google aistudio <https://aistudio.google.com/>

学問の自由 寺田寅彦 https://www.aozora.gr.jp/cards/000042/files/43535_24583.html

fs の比較表 <https://cran.r-project.org/web/packages/fs/vignettes/function-comparisons.html>

Google Cloud <https://console.cloud.google.com/>

ggplot <https://ggplot2-book.org/>

r4ds <https://r4ds.hadley.nz/>

CRAN パッケージ https://cran.r-project.org/web/packages/available_packages_by_name.html

気象庁 <https://www.jma.go.jp/bosai/nowc/>

DeepL <https://www.deepl.com/ja/>

TexTra <https://mt-auto-minhon-mlt.ucr.i.jgn-x.jp/>

Rtools <https://cran.r-project.org/bin/windows/Rtools/>

I ファイルとディレクトリの操作

日常業務で次のような経験をしたことはないでしょうか。あるディレクトリ内の多数のサブディレクトリ中に PDF・エクセル・ワードなど多くの種類のファイルがあって、数十分かけて種類別のディレクトリにまとめる作業をした。その作業中にマウスの操作を誤ったため戻す作業に手間取った。多くのファイルを更新日時順で名前をつけ直す作業をしたが、その後でさらにファイルを更新したので、ファイル名を何度も付け直したなどです。fs パッケージを使うと、このようなファイルやディレクトリの操作を自動化できます。

プログラミングでの自動化作業では、データの入出力がつきものです。入出力のファイル名が決まっているときは直接コードに書けますが、必ずしもそうとは限りません。特定の拡張子のファイルを別の形式に変換することができます。ファイルの移動や名称変更もあるでしょう。このような場合にも fs パッケージの関数を使うと簡単に自動化できます。

本章では、各種パスの取得と設定、ディレクトリとファイルの生成、ディレクトリ内の一覧の取得、拡張子の操作、ファイルの移動・コピー・削除、ファイル名の変更について解説します。また、マウスでファイルやディレクトリを指定する方法についても説明します。

I.1 ファイルとディレクトリを操作するパッケージ

R の base パッケージにもファイル操作のための関数が多くあります。しかし、R の発展の中で徐々に関数が追加されたため、関数名が分かりにくく引数の一貫性が無いという難点があります。そこで、本章では fs パッケージを活用します。fs では、命名規則が一貫していて覚えやすく、有用な関数が追加されています。また、ベクトル化した引数を受けることができるため、一括で操作できます。

パッケージをインストールするには、`install.package()` を使います。fs パッケージをインストールする場合、以下を実行します。

コード I.1 (file-install.R) : fs パッケージのインストール

```
install.packages("fs")
```

使用する前には `library()` でパッケージを呼び出します。

コード I.2 (file-library.R) : fs パッケージの呼び出し

```
library(fs)
```

fs パッケージの関数は、ディレクトリ操作(`dir_*`)、ファイル操作(`file_*`)、パス操作(`path_*`)の関数に分かれます。

パッケージのオブジェクト(関数やデータ)の一覧を取得するには, `ls("package:パッケージ名")`を使います。一覧のうち条件に合致したものだけを抽出するには, `stringr::str_subset()`を使います(3.6 節を参照)。

コード I.3 (file-list.R) : fs パッケージの関数一覧

```
# install.packages("tidyverse") # tidyverse をインストールしていないとき
# library(tidyverse) # tidyverse を呼び出していないとき
ls("package:fs") |> stringr::str_subset("^dir")
## [1] "dir_copy"     "dir_create"   "dir_delete"   "dir_exists"   "dir_info"
## [6] "dir_ls"       "dir_map"      "dir_tree"     "dir_walk"
ls("package:fs") |> stringr::str_subset("^file")
## [1] "file_access"   "file_chmod"    "file_chown"
## [4] "file_copy"     "file_create"   "file_delete"
## [7] "file_exists"   "file_info"     "file_move"
## [10] "file_show"    "file_size"    "file_temp"
## [13] "file_temp_pop" "file_temp_push" "file_touch"
ls("package:fs") |> stringr::str_subset("^path")
## [1] "path"          "path_abs"      "path_common"
## [4] "path_dir"      "path_expand"   "path_expand_r"
## [7] "path_ext"      "path_ext_remove" "path_ext_set"
## [10] "path_ext<-'"  "path_file"    "path_filter"
## [13] "path_has_parent" "path_home"    "path_home_r"
## [16] "path_join"    "path_norm"    "path_package"
## [19] "path_real"    "path_rel"     "path_sanitize"
## [22] "path_split"   "path_temp"    "path_tidy"
## [25] "path_wd"
```

fs パッケージには, base や OS のファイル操作のコマンドには存在しない機能があります。拡張子を取り除いた部分を取り出す関数の自作後に, fs で関数を見つけ, 機能の劣る車輪の再発明を後悔しました。なお, 英語ですが fs, base, Windows のコマンド比較が以下にあります。

<https://cran.r-project.org/web/packages/fs/vignettes/function-comparisons.html>

I.2 各種パスの取得と設定

fs パッケージには, 作業ディレクトリなどの特殊なディレクトリのパスを取得する関数があります(表 I.1)。

表 I.1: 各種パスを設定する関数

関数	パス
<code>path_wd()</code>	作業ディレクトリ
<code>path_temp()</code>	一時ファイルのディレクトリ
<code>path_package()</code>	パッケージのディレクトリ
<code>path_home()</code>	ユーザのホームディレクトリ

関数	パス
path_home_r()	R のホームディレクトリ

絶対パスを使わないときは、作業ディレクトリを意識するのが良いでしょう。ファイル一覧の取得でディレクトリを指定しなければ、作業ディレクトリの一覧が表示されるためです。作業ディレクトリを取得するには、`path_wd()`を使います。Rに組み込まれている`getwd()`でも文字列としては同じものを取得できます。

コード I.4 (file-getwd.R) : 作業ディレクトリの取得

```
# OS や使用状況などで表示は異なります
# Windows の場合は、USERNAME にユーザ名が表示されます
path_wd()
## "C:/Users/USERNAME/Documents"
getwd() # 文字列としては同じ
## [1] "C:/Users/USERNAME/Documents"
```

作業ディレクトリを指定する関数は fs パッケージにはないので、`setwd()`を使用します。

コード I.5 (file-setwd.R) : 作業ディレクトリの設定

```
wd <- "c:/NEW/DIRECTORY"
setwd(wd)
```

本節から I.5 節までは、R-4.3.3 の fs パッケージのディレクトリを作業ディレクトリとして説明します。

`path_package()`はパッケージのディレクトリを取得するのに使います。引数にはパッケージ名を文字列で指定します。

コード I.6 (file-setwd-pkg.R) : fs パッケージのディレクトリを作業ディレクトリに設定

```
wd <- path_package("fs")
## C:/Users/USERNAME/AppData/Local/R/win-library/4.3/fs
wd
setwd(wd)
```

`path_temp()`で一時ファイルのディレクトリ名を取得できます。自動化作業で一時的に使用するときに利用します。なお、`path_temp()`で得られるディレクトリは R や RStudio 起動時に自動的に生成され、終了するとディレクトリごと削除されます。

コード I.7 (file-path-temp.R) : 一時パスの取得

```
path_temp()  
## C:/Users/USERNAME/AppData/Local/Temp/Rtmpi8KiHB
```

`file_temp()`を使用すると、一時ファイルの名前が生成されます。拡張子を指定しないと、拡張子のないファイルの名前が自動的に生成されます。`"hoge.txt"`のようにファイル名を指定すると、一時ディレクトリのファイル名が生成されます。複数のファイルを扱うときには、`pattern = "example_"`, `ext = "txt"`のようにするとよいでしょう。`pattern`と`ext`でファイル本体の名前と拡張子をそれぞれ指定可能です。

コード 1.8 (file-file-temp.R) : ファイル名と拡張子を指定した一時パス

```
file_temp()  
## C:/Users/USERNAME/AppData/Local/Temp/Rtmpi8KiHB/file318852404cd1  
path_temp("hoge.txt")  
## C:/Users/USERNAME/AppData/Local/Temp/Rtmpi8KiHB/hoge.txt  
file_temp(pattern = "example_", ext = "txt")  
## C:/Users/USERNAME/AppData/Local/Temp/Rtmpi8KiHB/example_dd07fdc710e.txt
```

【注意】`file_temp()`で生成するのは、あくまでファイル名のパス(文字列)です。ファイル自体が生成されるわけではありません。また、一時ディレクトリや一時ファイルは、作業ディレクトリではなく、ユーザのホームディレクトリ(`C:/Users/USERNAME/`など)の中になります。

1.3 ディレクトリとファイルの生成

ディレクトリを生成するには`dir_create()`を、ファイルを生成するには`file_create()`を使います。引数`path`にディレクトリやファイルの名称を指定します。`dir_create()`は、自動処理で生成したファイルをディレクトリ別で保存する際に活用できます。`file_create()`で生成したファイルは空のファイルですので、使用頻度は高くないかもしれません。

ここでは、`abc`というディレクトリと`hoge.txt`, `fuga.txt`, `piyo.txt`という3つのファイルを生成します。なお、このディレクトリとファイルは1.6節で使用します。

コード 1.9 (file-create.R) : 作業用のディレクトリとファイルの生成

```
dir_create("abc")  
file_create(c("hoge.txt", "fuga.txt", "piyo.txt"))
```

1.4 ディレクトリとファイル一覧の取得

ファイルやディレクトリの一覧を取得するには`dir_ls()`を使います。引数に何も指定しないと、作業ディレクトリの一覧が表示されます。`recurse = TRUE`とすると下位ディレクトリも表示されます。

コード I.10 (file-ls.R) : ディレクトリー一覧の取得

```
dir_ls()  
## COPYRIGHTS DESCRIPTION doc help html  
## INDEX libs LICENSE MD5 Meta  
## NAMESPACE NEWS.md R WORDLIST  
dir_ls(recurse = TRUE) # recurse = TRUE で下位ディレクトリも表示  
## COPYRIGHTS DESCRIPTION  
## doc doc/function-comparisons.html  
## doc/function-comparisons.R doc/function-comparisons.Rmd  
## doc/index.html help  
## (以下省略)
```

引数 type = "file" でファイル、 type = "directory" でディレクトリーの一覧になります。

コード I.11 (file-ls-type.R) : type を指定したディレクトリー一覧の取得

```
dir_ls(type = "file") # ファイルのみ  
## COPYRIGHTS DESCRIPTION INDEX LICENSE MD5  
## NAMESPACE NEWS.md WORDLIST  
dir_ls(type = "directory") # ディレクトリのみ  
## doc help html libs Meta R
```

引数 path を指定すると、指定したディレクトリ内の一覧をフルパスで取得できます。

コード I.12 (file-ls-stringr.R) : path を指定したディレクトリー一覧の取得

```
dir_ls(path = path_package("stringr"))  
## C:/Users/USERNAME/AppData/Local/R/win-library/4.3/stringr/data  
## C:/Users/USERNAME/AppData/Local/R/win-library/4.3/stringr/DESCRIPTION  
## C:/Users/USERNAME/AppData/Local/R/win-library/4.3/stringr/doc  
## (以下省略)
```

作業ディレクトリのファイルのフルパスを入手したいときは、 path = path_wd() とし type = "file" とします (path_wd() については I.2 節を参照)。

コード I.13 (file-ls-fullpath.R) : 作業ディレクトリでの一覧の取得

```
dir_ls(path = path_wd(), type = "file")  
## C:/Users/USERNAME/AppData/Local/R/win-library/4.3//fs/COPYRIGHTS  
## C:/Users/USERNAME/AppData/Local/R/win-library/4.3//fs/DESCRIPTION  
## C:/Users/USERNAME/AppData/Local/R/win-library/4.3//fs/INDEX  
## C:/Users/USERNAME/AppData/Local/R/win-library/4.3//fs/LICENSE  
## C:/Users/USERNAME/AppData/Local/R/win-library/4.3//fs/MD5  
## C:/Users/USERNAME/AppData/Local/R/win-library/4.3//fs/NAMESPACE  
## C:/Users/USERNAME/AppData/Local/R/win-library/4.3//fs/NEWS.md  
## C:/Users/USERNAME/AppData/Local/R/win-library/4.3//fs/WORDLIST
```

引数 `regexp` に文字列を指定すると、結果を絞り込むことができます。文字列には正規表現(3.4節を参照)も使えます。たとえば、下位ディレクトリも含めて拡張子が `doc` か `html` を含むファイルだけを表示させたいときは以下のようにします。

コード I.14 (`file-regexp.R`) : 正規表現を使った一覧の取得

```
dir_ls(type = "file", recurse = TRUE, regexp = "(doc|html)")  
## doc/function-comparisons.html  
## doc/function-comparisons.R  
## doc/function-comparisons.Rmd  
## doc/index.html  
## html/00Index.html  
## html/R.css
```

ディレクトリとファイルをツリー表示させたいときには、`dir_tree()`を使います。`type`でディレクトリを指定できます。引数を指定しないとディレクトリとファイルのすべてが表示されます。次のコードでは、`fs`パッケージの一覧を表示しています。実際のディレクトリ構成(図 I.1)と合わせて見て下さい。

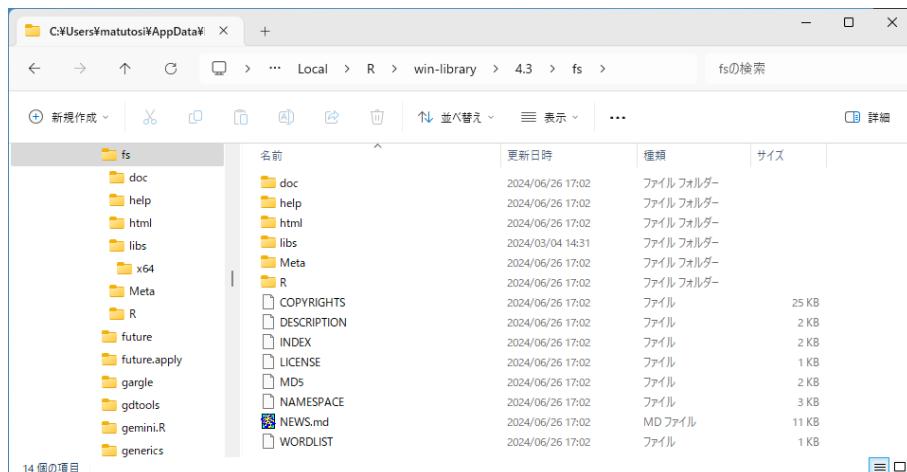


図 I.1: `fs` パッケージのディレクトリ構成

コード I.15 (`file-ls-tree.R`) : ディレクトリのツリー表示

```
dirs <- dir_tree()  
## .  
## └── COPYRIGHTS  
## └── DESCRIPTION  
## └── doc  
##     ├── function-comparisons.html  
##     ├── function-comparisons.R  
##     ├── function-comparisons.Rmd  
##     └── index.html  
## 中略  
## └── INDEX  
## └── libs  
##     └── x64  
##         ├── fs.dll  
##         └── symbols.rds
```

```

## (以下省略)
dirs
## COPYRIGHTS                               DESCRIPTION
## doc                                         doc/function-comparisons.html
## (以下省略)
dir_tree(type = "directory") # ディレクトリのみ
## .
##   doc
##   help
##   html
##   libs
##     └── x64
##   Meta
##   R

```

`reurse = 0` では現在のディレクトリのみが、`reurse = 1` では 1 つ下の段階までが、`reurse = 2` では 2 つ下の段階までが表示されます。`dir_tree()` の返り値は `dir_ls()` と同じでディレクトリやファイルの一覧です。ただし、ツリー表示のために `reurse = TRUE` がデフォルトになっており、サブディレクトリの内容も含みます。

コード I.16 (file-ls-tree-reuse.R) : `reurse` を指定したディレクトリのツリー表示

```

dir_tree(reurse = 0) # 下位ディレクトリを表示しない
## .
##   COPYRIGHTS
##   DESCRIPTION
##   doc
##   help
##   html
##   INDEX
##   libs
## (以下省略)
dir_tree(reurse = 1) # 1 つ下位のディレクトリまで表示
## .
##   COPYRIGHTS
##   DESCRIPTION
##   doc
##     └── function-comparisons.html
##     └── function-comparisons.R
## 中略
##   INDEX
##   libs
##     └── x64
## (以下省略)

```

ファイルやディレクトリが存在するかどうかは `file_exists()` や `dir_exists()` で確認できます。返り値は論理値 (TRUE か FALSE) です。

コード I.17 (file-exests.R) : ファイルとディレクトリ有無を取得

```

file_exists("DESCRIPTION")
## DESCRIPTION
##     TRUE
file_exists("DESCRIPTIONS")
## DESCRIPTIONS
##     FALSE
dir_exists("doc")
## doc
## TRUE
dir_exists("docs")
## docs
## FALSE

```

ファイルの詳細な情報を得たいときには、`dir_info()`を使います。データフレーム形式でパス(path), 種類(type), サイズ(size), 権限(permissions), 更新時刻(modification_time)などを入手できます。引数には、`path_ls()`と同じく path, recurse, type, regexpなどを指定できます。

コード I.18 (file-info.R) : ディレクトリとファイルの詳細な情報の取得

```

dir_info()
## # A tibble: 15 × 18
##   path      type          size permissions modification_time  # 省略
##   <fs::path> <fct>    <fs::byte> <fs::perms> <dttm>
## 1 base      symlink       29  rw-
## 2 COPYRIGHTS file     24.14K  rw-
## 3 DESCRIPTION file    1.71K  rw-
## 4 doc        directory     0  rw-
## 5 help       directory     0  rw-
## (以下省略)

```

`dir_info()`に `dplyr::filter()` や `stringr::detect()` など(3.6 節を参照)を組み合わせると、詳細な情報をもとにファイルを絞り込むことができます。

コード I.19 (file-info-filter.R) : `dir_info()` と `stringr::detect()` を使ったファイルの絞り込み

```

dir_info() |>
  dplyr::filter(size > 10^3) |> # size が 1000 を超えるもの
  dplyr::filter(stringr::str_detect(path, "[A-G]")) # path に A-G を含むもの
## # A tibble: 6 × 18
##   path      type          size permissions modification_time  # 省略
##   <fs::path> <fct>    <fs::byte> <fs::perms> <dttm>
## 1 COPYRIGHTS file     24.14K  rw-
## 2 DESCRIPTION file    1.71K  rw-
## 3 INDEX      file     1.64K  rw-
## 4 MD5        file     1.45K  rw-
## 5 NAMESPACE   file    2.31K  rw-
## 6 NEWS.md    file    10.84K rw-

```

1.5 拡張子の変更・取得・除去

ファイルを形式変換して書き込む時には、ファイル名のうち拡張子だけ変更するでしょう。たとえば、`hoge.txt` から `hoge.docx` のような変更です。単純には、`stringr::str_replace()`で `txt` を `docx` に置換すれば良さそうです(3.6 節を参照)。しかし、`txt` という文字列がディレクトリ名やファイル名に含まれる可能性があります。そのため、置換する場合は

`str_replace("\\.txt$", "\\!.docx$")` と正規表現で拡張子だと明示するのが安全ですが、正直なところ面倒です。そこで、拡張子を設定する `path_ext_set()` を使います。次のコードは `fs` パッケージ内の拡張子が `md` であるファイル名の拡張子を `txt` に変更するものです。

コード 1.20 (file-ext-set.R) : 拡張子の設定

```
md_files <- dir_ls(type = "file", regexp = "\\.md$")
md_files
## NEWS.md
txt_file <- path_ext_set(md_files, "txt")
txt_file
## NEWS.txt
```

【注意】 `path_ext_set()` でファイル名のオブジェクトを変更しても実態のファイル名やファイルの構造は変更されません。あくまで R のオブジェクトの文字列(パス)を修正しただけです。実態のファイル名を変更する方法は 1.6 節を参照してください。もちろん、PDF や Word などのファイル形式の相互変換(8.8 節を参照)もしていません。

`fs` パッケージには、拡張子だけ取り出す `path_ext()` や拡張子を取り除く `path_ext_remove()` という関数もあります。

コード 1.21 (file-ext.R) : 拡張子のみとファイル名のみの抽出

```
path_ext(md_files)
## [1] "md"
path_ext_remove(md_files)
## NEWS

# 下位ディレクトリも対象とするとき
all_files <- dir_ls(type = "file", recurse = TRUE)
path_ext(all_files)
## [1] ""      ""      "html"  "R"     "Rmd"   "html"  "rds"   ""
## [11] "rds"   "html"  "css"   ""     "dll"   "rds"   ""     ""
## [21] "rds"   "rds"   "rds"   "rds"   "rds"   "md"    ""     ""
## [31] ""
path_ext_remove(all_files)
## COPYRIGHTS          DESCRIPTION          doc/function-comparisons
## doc/function-comparisons doc/function-comparisons doc/index
```

1.6 ファイルの移動・コピー・削除

本節では作業ディレクトリに以下のディレクトリとファイルがあるとします。 ファイルを実際に生成して作業するには、コード 1.9 を実行してください。

```
abc # ディレクトリ  
hoge.txt fuga.txt piyo.txt # テキストファイルが3つ
```

ファイルを移動するには `file_move()` を使います。 返り値は移動後のパスですので、移動後のパスを使いたいときは別のオブジェクトに代入します。 なお、この返り値は非表示に設定されています。 そのため、`file_move(files, new_path = "abc")` とするとファイルの移動は実行されますが、返り値は表示されません。

【Tips】 返り値を表示させるには、変数(コード 1.22 では `result`)に代入してその内容を表示させるか、式全体を()で囲んで評価させます。

コード 1.22 (file-move.R) : 指定したディレクトリへのファイルの移動

```
files <- c("hoge.txt", "fuga.txt", "piyo.txt")  
files  
## [1] "hoge.txt" "fuga.txt" "piyo.txt"  
result <- file_move(files, new_path = "abc")  
result  
# abc/hoge.txt abc/fuga.txt abc/piyo.txt
```

これで3つのテキストファイルがディレクトリ `abc` に移動します。 移動後のファイルを作業ディレクトリに戻したいときは、`new_path = ". "` とします。 なお、ここでの`.`(ドット)は現在のディレクトリの意味です。

コード 1.23 (file-move-again.R) : 作業ディレクトリへのファイルの移動

```
result <- file_move(result, new_path = ".")  
result  
## ./hoge.txt ./fuga.txt ./piyo.txt
```

《Tips》 R での`.`(ドット)には複数の意味があり、関数によって評価のされ方が異なります。 `fs` パッケージでの文字列としての`". "`は、現在のディレクトリという意味です。 これは shell でディレクトリを表現するときと同じです。 一方、正規表現(3.4 を参照)では、`". "`は任意の1文字を意味します。 さらに、`magrittr` パッケージのパイプ(`%>%`)では、左辺(前のオブジェクト)という意味で使われます。 他にも別の意味で使われることがありますので、`.`を使う際には注意してください。

ファイルのコピーには、`file_copy()`を使います。返り値は`file_move()`と同様でコピー後のパスです。2つ目の引数にファイル名を指定すると、同じディレクトリにコピーされます(図 1.2)。

コード 1.24 (file-copy.R) : ディレクトリ内のファイルのコピー

```
copy_files <- stringr::str_c("copy_", files) # コピー後のファイル名  
(file_copy(files, copy_files)) # 両端の()で結果を表示  
## copy_hoge.txt copy_fuga.txt copy_piyo.txt
```

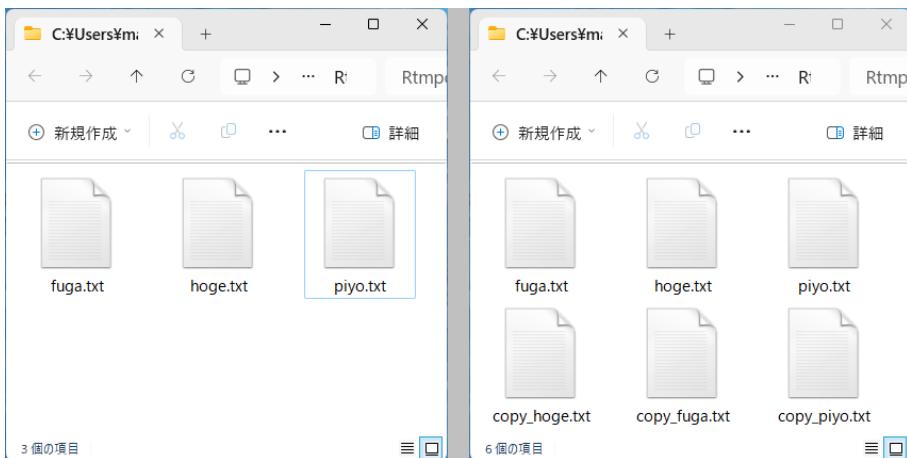


図 1.2: ファイルのコピー前(左)と後(右)のディレクトリ

2つ目の引数にディレクトリを指定すると、そのディレクトリにもとのファイル名でコピーされます。

コード 1.25 (file-copy-abc.R) : 指定したディレクトリへのファイルのコピー

```
(result <- file_copy(files, "abc")) # 両端の()で結果を表示  
## abc/hoge.txt abc/fuga.txt abc/piyo.txt
```

`file_copy()`の上書きを指定する引数はデフォルトが`overwrite = FALSE`のため、コピー先に同名のファイルがあるとエラーになります。

コード 1.26 (file-copy-error.R) : ファイルのコピーでのエラー

```
file_copy(files, "abc") # 上書きできずエラー  
## Error: [EEXIST] Failed to copy 'hoge.txt' to 'abc/hoge.txt':  
## file already exists
```

ファイルの削除には、`file_delete()`を使います。返り値は削除したファイルのパスです。次のコードでは、これまでにコピーして生成した6つのファイル、つまり `copy_` で始まる3つと、`abc` ディレクトリの3つを削除します。

コード 1.27 (file-delete.R) : ファイルの削除

```
(file_delete(copy_files))
## copy_hoge.txt copy_fuga.txt copy_piyo.txt
(file_delete(result))
## abc/hoge.txt abc/fuga.txt abc/piyo.txt
```

1.7 ファイル名の変更

ファイル名を変更するには、`file_move()`を使います。この関数はファイルの移動で使いましたが、名前の変更にも使えます。`file_move()`は引数に対象ファイルの `path` と移動・名称変更先のパス `new_path` を取ります。この引数の指定の仕方でファイルの移動になるか名称変更になるかが決まります(表 1.2)。

表 1.2: `file_move()`の引数と動作

<code>path</code>	<code>new_path</code>	動作
hoge.txt	fuga.txt	名称変更
abc/hoge.txt	abc/fuga.txt	名称変更
hoge.txt	abc/fuga.txt	移動+名称変更
abc/hoge.txt	xyz(ディレクトリ)	移動
hoge.txt	abc(ディレクトリ)	移動

`path` にはファイル名かディレクトリ名のパスを、`new_path` には移動先のパスを指定します。両方にディレクトリを指定しないと、作業ディレクトリ内での移動が実行されます。このため実質的には、ファイル名が変更されるという仕組みです。`path` と `new_path` の両方に同じディレクトリの異なるファイル名を指定した場合もファイル名が変更されます。`path` と `new_path` に異なるディレクトリ名で異なるファイル名を指定すると、移動とともにファイル名が変更されます。なお、`file_move()`では既に存在するファイルやディレクトリは上書きされるので、注意してください。

コード 1.28 (file-rename.R) : ファイル名の変更

```
(files <- dir_ls(regexp = "\\.txt$"))
## fuga.txt hoge.txt piyo.txt
(new_path <- c("foo.txt", "bar.txt", "baz.txt"))
## [1] "foo.txt" "bar.txt" "baz.txt"
new_path
## [1] "foo.txt" "bar.txt" "baz.txt"
(file_move(files, new_path))
## foo.txt bar.txt baz.txt
```

ファイル名の変更方法が分かったところで、少しだけ応用的な作業を考えます。`a.pdf`, `b.pdf`, …, `i.pdf`, `j.pdf` を `01.pdf`, `02.pdf`, …, `09.pdf`, `10.pdf` のように 10 個のファイル名を変更します。

コマンドプロンプトなどで実行するなら次のようなコマンドで変更可能です。dosコマンドの変数やループなどを駆使すると、もっと短く書けるのでしょうか。テキストエディタで書いてもそれほど時間がからないでしょうが、ファイル数が多くなれば大変です。

```
rename a.pdf 01.pdf
rename b.pdf 02.pdf
## 中略
rename i.pdf 09.pdf
rename j.pdf 10.pdf
```

実際のファイルで作業するときには、次のコードでダミーのファイルを生成してください。作業ディレクトリを開いておくと、ファイル名が一瞬で変更されるのを見ることができます。Windowsでは、`shell.exec()`(5.2節を参照)を使って開くことができます。

コード I.29 (file-rename-app-prep.R) : 作業用のダミーファイルの生成

```
paste0(letters[1:10], ".pdf") |>
  file_create()
shell.exec(path_wd()) # 作業ディレクトリを開く(Windowsのみ)
```

`fs`と`stringr`を使うと、以下のように簡潔に書けます。変更前のファイル名は`dir_ls()`で取得し、ファイル数を`len`とします。変更後のファイル名は、`paste0()`と`stringr::str_pad()`でつくります。`str_pad()`は字数合わせをする関数です(3.7節を参照)。

コード I.30 (file-rename-files.R) : 複数のファイル名の変更例

```
old <- dir_ls(regexp = "\\.pdf$")
len <- length(old)
new <-
  paste0(stringr::str_pad(1:len, width = 2, side = "left", pad = "0"), ".pdf")
file_move(old, new)
```

別の例を考えます。自分で完成したつもりのファイルに「完成版」と名前に付けることはよくあります。しかし、上司の修正指示を受けて「修正完成版」とし、さらに上司の上司の指示で「最終版」とさらに名前を変更します。これだけだったら良いのですが、確認していると間違いが見つかって修正しているうちに、ファイル名が混乱します。そんなときは、更新時刻順で連番の数字を付けてファイルを整理したくなります。

ここでは、作業ディレクトリにある`xlsx`を、更新時刻順で連番の数字を付けます。実際のファイルで作業するには、事前にダミーのファイルを生成します。`for`ループと`Sys.sleep()`を使って、60秒ごとにファイルを生成します。なお、`sample()`で生成する乱数でファイル名の順序が決まるので、結果は以下のコードとは異なります。

コード I.31 (file-rename-info-prep.R) : 作業用のダミーファイルの生成

```
olds <- paste0(letters[1:10], ".xlsx")
for(old in sample(olds)){
  file_create(old[i])
  Sys.sleep(60)
}
shell.exec(path_wd()) # 作業ディレクトリを開く(Windowsのみ)
```

まず、変更前のファイル一覧を取得してから、データフレームを並べ替える `dplyr::arrange()` を使って(2.5 節を参照)、更新日時順にしたファイル一覧を作成します。その後、`str_pad()`で字数合わせしたファイル名を作成します。最後に `file_move()`でファイル名を変更します(図 I.3)。

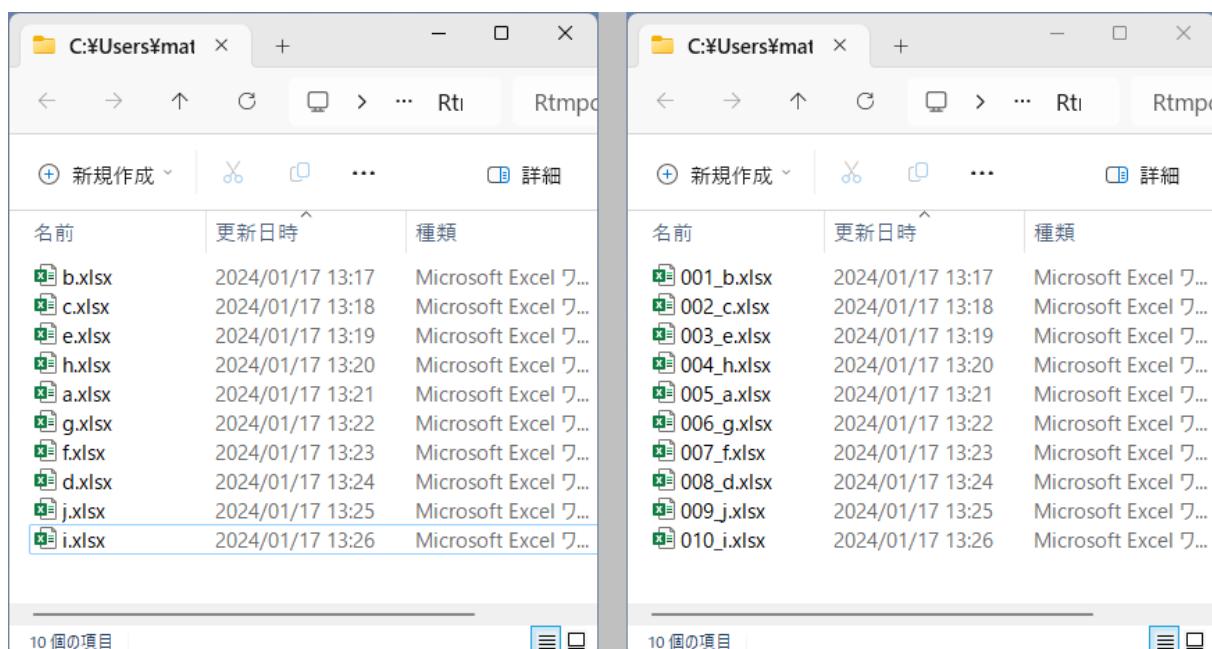


図 I.3: ファイル名の変更前(左)と後(右)

コード I.32 (file-rename-info.R) : 更新時刻順に連番を付けるファイル名の変更例

```
files <-
  dir_info(regexp = "\\.xlsx$") |>
  dplyr::arrange(modification_time) |> # modification_time で並べ替え
  `\$`(_, "path") # _$path と同じ
files
## b.xlsx c.xlsx e.xlsx h.xlsx a.xlsx g.xlsx f.xlsx d.xlsx j.xlsx i.xlsx
no <-
  1:length(files) |> # seq_along(files)でも同じ
  stringr::str_pad(width = 3, side = "left", pad = "0")
new_files <- stringr::str_c(no, "_", files)
new_files
## [1] "001_b.xlsx" "002_c.xlsx" "003_e.xlsx" "004_h.xlsx" "005_a.xlsx"
## [6] "006_g.xlsx" "007_f.xlsx" "008_d.xlsx" "009_j.xlsx" "010_i.xlsx"
file_move(files, new_files)
```

1.8 マウスでのファイルやディレクトリの指定

プログラミングに対して抵抗感のある人にとっては、マウスでファイルやディレクトリを指定できると少し楽に操作できます。 `tcltk` パッケージを使うと、マウスによる操作でディレクトリやファイルを指定できます。`tcltk` パッケージは、R をインストールすると既に入っているので、インストールの必要はありません。ただし、`tcltk` 内の関数を使用するときには、`library(tcltk)` で呼び出すか、以下のように関数名の前に `tcltk::` を付けます。

ディレクトリを選択するには `tcltk::tk_choose.dir()` を、ファイルを選択するには `tcltk::tk_choose.files()` を使います。`tcltk::tk_choose.dir()` を実行すると、ディレクトリの選択画面がでますので、指定したいディレクトリをマウスで指定します(図 1.4)。その結果が返り値として得られます。次のコードでは、`tcltk::tk_choose.dir()` で得たディレクトリを `selected_dir` に代入して、作業ディレクトリに設定します

コード 1.33 (file-tcltk.R) : マウスでのファイルやディレクトリの指定

```
selected_file <- tcltk::tk_choose.files() # ファイルを指定する場合
# ここでマウスによる操作
selected_file
# デスクトップの hoge.xlsx というファイルを指定したとき
## [1] "C:/Users/USERNAME/Desktop/hoge.xlsx"
selected_dir <- tcltk::tk_choose.dir() # ディレクトリを指定する場合
# ここでマウスによる操作
selected_dir
# デスクトップを指定したとき
## [1] "C:/Users/matutosi/Desktop"
setwd(selected_dir)
getwd()
## [1] "C:/Users/matutosi/Desktop"
```



図 1.4: ディレクトリの選択画面

【削除】 `fs` を活用した自動化例で「R のバージョンアップ時の設定ファイル更新」を入れていましたが、バッサリ削除しました。カスタマイズ済みの設定は、ユーザ・ディレクトリ (`c:/Users/USERNAME/Documents`) に `Rconsole` や `.Rprofile` として保存しておけばバージョンアップしてもそのまま使えることがわかったためです。

【相談】 削除したままでも良ければそのままにしますが、何か自動化例があったほうが良ければ追加します。たとえば、ファイル名や拡張子によってファイルを分類して、それぞれにディレクトリにつくって移動するなどです。

【対応状況】 アンケート結果の集計と売上を自動集計して図示するという作業を自動化するということでまとめ直しました。もう少し自動化の例としてもう少し付け足したほうが良い点などがあれば、ご指摘いただければありがとうございます。この内容であれば、基本操作の第1章に入れても違和感がないと思いますが、いかがでしょうか。

2 データの操作

アンケート結果や毎月の売上データについて、エクセルを使って集計があるかもしれません。頻繁にあれば結構な時間になりますので、ぜひ自動化しましょう。あまり頻繁でない作業は集計の方法を忘れてしまうので、作業自体の時間よりも思い出すのに時間がかかるかもしれません。このような作業も自動化すれば、プログラムが実行してくれます。また、集計方法がちょっと異なれば項目の並び順や図の大きさが異なり統一感が失われますが、自動すれば統一可能です。

エクセルなどの表計算ソフトで集計していると、もとのデータを誤って書き換えてしまう恐れがあります。そのため、データと集計作業は分離しておくのが安全です。プログラムを使って集計すれば、データと集計作業を分離できるので、データを書き換えてしまう心配はありません。また、エクセルでの集計では、具体的な作業内容はセルの中を確認する必要があります。一つ一つセルを追いかけて見るのは非常に面倒です。プログラムはコードを見れば、集計の方法は明らかです。

エクセルでは複数の属性のデータで集計や作図するには、属性の数だけ作業を繰り返す必要があります。プログラムを使えば、1回の実行で完了しますし、属性ごとに別々のファイルや図として出力することも簡単です。

`tidyverse` には、データ集計の前処理から集計作業、さらには作図までのパッケージが揃っています。そのため、一連のものとして作業を完了させることができます。また、可読性の高いコードを書くことができるため、データの形式が変わっても柔軟に対応できます。いつもと違う方法で集計・作図したくなったときにも対応できます。

2.1 データを操作するパッケージ

本章で主に使用するのは、`tidyverse` というパッケージ群です。コアパッケージ(表 2.1)と非コアのパッケージがあります。

表 2.1: `tidyverse` に含まれるパッケージ

パッケージ 操作対象など	
<code>tibble</code>	データフレームの拡張型
<code>tidyr</code>	整然データ
<code>dplyr</code>	データフレーム
<code>ggplot2</code>	作図
<code>purrr</code>	繰り返し処理
<code>lubridate</code>	日付・時間データ

パッケージ 操作対象など

readr	ファイル入出力
stringr	文字列
forcats	ファクター

本章では、コアパッケージのうちデータフレームの拡張型のクラスを扱う `tibble`、データに前処理を施して整然データにする `tidyr`、データフレームを操作する `dplyr`、作図する `ggplot2`、繰り返し処理をする `purrr`について説明します。`lubridate`は、日付や時間データを扱いますが、日付データの自動化(第4章)で使い方を説明します。`readr`は、データの入出力で活用しますが、ファイルの読み込み(9.2節)と書き込み(9.4節)で説明します。`stringr`は、文字列の処理で重要な役割を果たしますので、文字列の操作(第3章)で詳しく紹介します。本書ではファクターをほとんど扱っていないため、`forcats`は説明しません。`tidyverse`のさらなる活用方法は、Wickham, Çetinkaya-Rundel, and Grolemund (2024)で学んでください。

`tidyverse`に含まれるパッケージ群は一括でインストール・呼び出しが可能です。

コード 2.1 (analysis-install.R) : tidyverse のインストール

```
install.packages("tidyverse")
```

`library(tidyverse)`でコアパッケージをまとめて呼び出します。また、`library(tidyr)`のように個々のパッケージを呼び出すこともできます。

コード 2.2 (analysis-library.R) : tidyverse の呼び出し

```
library(tidyverse)
## Loading required package: grDevices
## Loading required package: graphics
## Loading required package: utils
## — Attaching core tidyverse packages ————— tidyverse 2.0.0 —
## ✓ dplyr     1.1.4    ✓ readr     2.1.5
## ✓ forcats   1.0.0    ✓ stringr   1.5.1
## ✓ ggplot2   3.5.0    ✓ tibble    3.2.1
## ✓ lubridate 1.9.3    ✓ tidyr     1.3.1
## ✓ purrr    1.0.2
## — Conflicts ————— tidyverse_conflicts() —
## ✘ dplyr::filter() masks stats::filter()
## ✘ dplyr::lag()    masks stats::lag()
## ✘ dplyr::select() masks MASS::select()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

`tidyverse`を呼び出すと、`dplyr::filter()` masks `stats::filter()`としてRの基本パッケージの関数を覆い隠したことが表示されます。上記の場合であれば、`filter()`と入力すれば`dplyr::filter()`と同じ意味になります。`stats`の同名の関数を呼び出すには、`stats::filter()`のようにパッケージ名を明示します。

2.2 集計用のデータ

データの読み込みには、`readxl` パッケージと `openxlsx` パッケージの関数を使います。これらの関数の使い方は、9.2 節を参照してください。

同じ書式で店舗ごとや月ごとなどでデータが入っているときなど、多くのシートが入っているエクセルのファイルのときは、一気にデータを読み込むと便利です。そのための関数を定義します。

まず、`getSheetNames()`でエクセルのパスを指定してシート名を取得します。`map()`を使って取得したシート名で繰り返して、`read.xlsx()`でエクセルのファイルからデータを読み込み、`tibble`に変換します。その後、シート名を読み込んだデータフレームの名前として設定します。シート名を追加するときは、`map2()`と `mutate()`で追加します。なお、`map()`や `map2()`は、2.8 節を参照してください。

コード 2.3 (`analysis-read-all-sheets-fun.R`) : エクセルの全シートを読み込む関数

```
read_all_sheets <- function(path, add_sheet_name = TRUE){
  sheets <- openxlsx::getSheetNames(path) # シート名の一覧
  xlsx <-
    sheets |>
    purrr::map(\(x){ # シートごとに
      openxlsx::read.xlsx(path, sheet = x) # データの読み込み
    }) |>
    purrr::map(tibble::tibble) # tibble に変換
  names(xlsx) <- sheets # シート名
  if(add_sheet_name){ # シート名を tibble に追加するか
    xlsx <- purrr::map2(xlsx, sheets,
      \(.x, .y){
        dplyr::mutate(.x, sheet = .y) # シート名の列を追加
      }
    )
  }
  return(xlsx)
}
```

`answer` と `attribute` はアンケートの結果と回答者の属性のデータ、`sales` と `unit_price` は売上と単価のデータです。

コード 2.4 (`analysis-read.R`) : 集計用データの読み込み

```
wd <- fs::path_temp()
setwd(wd)
```

```

files <- c("answer.xlsx", "attribute.xlsx", "sales.xlsx", "unit_price.xlsx")
urls <- paste0("https://matutosi.github.io/r-auto/data/", files)
curl::multi_download(urls)
answer <- readxl::read_excel(files[1])
attribute <- readxl::read_excel(files[2])
sales <- read_all_sheets(files[3]) |> dplyr::bind_rows()
unit_price <- readxl::read_excel(files[4])

```

読み込んだデータの概要は以下のとおりです。

コード 2.5 (analysis-answer.R) : answer の概要

```

head(answer, 5)
## # A tibble: 5 × 3
##   id    item    ans
##   <chr> <chr>   <chr>
## 1 1     satisfy 1
## 2 1     apps    <NA>
## 3 1     comment コメントコメント自由記述コメント自由記述
## 4 2     satisfy 1
## 5 2     apps    HAD;JASP;S-PLUS;SPSS

```

answer はアンケートの回答で、id は回答者の識別番号、item は質問項目、ans は回答内容です。

コード 2.6 (analysis-attribute.R) : attribute の概要

```

head(attribute, 5)
## # A tibble: 5 × 3
##   id    area    period
##   <chr> <chr>    <dbl>
## 1 1     関東      5
## 2 2     関東      19
## 3 3     中部      12
## 4 4     九州・沖縄 15
## 5 5     中国・四国  6

```

attribute はアンケートの回答者の属性で、id は回答者の識別番号、area は居住地域、period は経験年数です。なお、id は answer の id と対応します。

コード 2.7 (analysis-sales.R) : sales の概要

```

head(sales, 5)
## # A tibble: 5 × 14
##   period うどん オムライス カツ丼 カレー サンドウィッチ スパゲティ
##   <chr>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 2023-01    47       61       45       46      12       12
## 2 2023-02    32       55       41       31      19       21
## 3 2023-03    44       42       47       46      16       21

```

```

## 4 2023-04      48       40     38     46      11      23
## 5 2023-05      37       49     47     44      14      19
## # i 7 more variables: チャーハン <dbl>, ドリア <dbl>, ピザ <dbl>,
## # 蕎麦 <dbl>, 洋定食 <dbl>, 和定食 <dbl>, sheet <chr>

```

`sales` は売上データで、`period` うどんから和定食までの料理名の項目が並び、最後に `sheet` という列があります。`period` は販売年月で、`yyyy-mm` 形式です。うどんから和定食までの料理名は全部で 12 あり、販売数が記録されています。`sheet` は店舗名で、読み込み時のエクセルのシート名から取り込んでいます。

コード 2.8 (analysis-unit-price.R) : `unit_price` の概要

```

head(unit_price, 5)
## # A tibble: 5 × 2
##   item    price
##   <chr>   <dbl>
## 1 和定食   650
## 2 洋定食   700
## 3 カレー    600
## 4 カツ丼    650
## 5 うどん   500

```

`unit_price` は料理ごとの単価です。`item` は `sales` と対応しており、`price` はそれぞれの単価です。

2.3 `tibble` の生成

データ集計では、データをエクセルのファイルから読み込むこともありますが、プログラム内の変数からデータフレームを作ることもあります。変数からデータフレームを作るには `tibble()` を使います。たとえば、アンケート結果の `answer` の元となるデータを生成するには次のようにします。なお、`set.seed(12)` と乱数種を指定することでサンプルのデータと同じものが生成されます。

コード 2.9 (analysis-tibble-tibble.R) : `tibble()` による `tibble` の生成

```

set.seed(12)
n <- 100
id <- seq(n)
area <- sample(
  c("北海道・東北", "関東", "中部", "近畿", "中国・四国", "九州・沖縄"),
  n, replace = TRUE)
period <- sample(1:30, n, replace = TRUE, prob = 30:1)
tibble::tibble(id, area, period) |>
  head(5)
## # A tibble: 5 × 3
##       id   area period
##   <dbl> <chr>  <dbl>
## 1     1 北海道・東北  1.00
## 2     2 関東        2.00
## 3     3 中部        3.00
## 4     4 近畿        4.00
## 5     5 中国・四国  5.00

```

```

## <int> <chr>      <int>
## 1   関東          5
## 2   関東          19
## 3   中部          12
## 4   九州・沖縄    15
## 5   中国・四国    6

```

簡単なデータを手入力するときには、`tribble()`を使います。既にテキストデータとして表形式のものがあれば、それを編集して`tribble()`を使うのが良いかもしれません。

コード 2.10 (analysis-tibble-tribble.R) : `tribble()`による `tibble` の生成

```

tibble::tribble(
  ~item, ~price,
  "和定食" , 650,
  "洋定食" , 700,
  "カレー" , 600,
  "カツ丼" , 650,
  "うどん" , 500,
  "蕎麦" , 500,
  "オムライス" , 600,
  "チャーハン" , 630,
  "ドリア" , 700,
  "ピザ" , 740,
  "スペゲティ" , 710,
  "サンドウィッチ" , 450 ) |>
  head(3)
## # A tibble: 3 × 2
##   item    price
##   <chr>   <dbl>
## 1 和定食    650
## 2 洋定食    700
## 3 カレー    600

```

`tibble()`や`tribble()`の返り値は、`tibble`という形式です。これは、データフレーム(`data.frame`)を使いやく拡張したものです。`tibble`では、行数と列数を表示するとともに、データのタイプとして`int`(整数)や`chr`(文字列)などが表示されます。また、画面の幅に表示を合わせてくれます。

なお、`tibble`でないデータフレームを`tibble`に変換するには、`as_tibble()`を使います。

コード 2.11 (analysis-tibble-as-tibble.R) : データフレームへの`tibble`の変換

```

data(iris)
class(iris)
## [1] "data.frame"

```

```

iris # 全部表示される
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1        3.5         1.4        0.2  setosa
## 2          4.9        3.0         1.4        0.2  setosa
## 3          4.7        3.2         1.3        0.2  setosa
## 4          4.6        3.1         1.5        0.2  setosa
## 5          5.0        3.6         1.4        0.2  setosa
## (中略)
## 149         6.2        3.4         5.4        2.3 virginica
## 150         5.9        3.0         5.1        1.8 virginica
iris_tibble <- tibble::as_tibble(iris)
class(iris_tibble)
## [1] "tbl_df"     "tbl"        "data.frame"
iris_tibble # 最初の部分が表示される
## # A tibble: 150 × 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##       <dbl>      <dbl>      <dbl>      <dbl> <fct>
## 1          5.1        3.5         1.4        0.2  setosa
## 2          4.9        3.0         1.4        0.2  setosa
## 3          4.7        3.2         1.3        0.2  setosa
## 4          4.6        3.1         1.5        0.2  setosa
## 5          5.0        3.6         1.4        0.2  setosa
## 6          5.4        3.9         1.7        0.4  setosa
## 7          4.6        3.4         1.4        0.3  setosa
## 8          5.0        3.4         1.5        0.2  setosa
## 9          4.4        2.9         1.4        0.2  setosa
## 10         4.9        3.1         1.5        0.1  setosa
## # i 140 more rows
## # i Use `print(n = ...)` to see more rows

```

2.4 整然データへの整形

データ集計の前には、集計しやすい形式にデータを整形することが重要です。一般的には `tidy` なデータ、つまり整然データであれば、データの集計や作図が楽にできます。

データフレームや `tibble` であっても整然であるとは限りません。整然データは、次の4つを満たすことが必要です。

1. 1つの変数が1つの列を構成する
2. 1つの観測が1つの行を構成する
3. 1つのタイプの観測群が1つの表を構成する
4. 1つの値が1つのセルを構成する

入手できるものが整然データではないことはよくあります。また、関数によっては非整然データが必要なことがあります。さらに、データの内容を人間の目で確認するときは非整然データの方が見やすいことがあります。そのため、整然データと非整然データを相互に変換できることが重要です。

`tidyverse` パッケージには、整然データに変換するための関数が多く含まれています。`tidyverse` の関数は、一般的には第1引数に対象とするデータフレームを指定します。そのため、以下の説明で第1引数がデータフレームのときは、説明を省略します。

`answer` は、`item` と `ans` の各列に複数の変数が混在しており、1 行が 1 つの観測ではありません。`satisfy` と `apps` と `comment` は、それぞれが質問項目(満足度と使用しているアプリと自由記述のコメント)で、`ans` はそれぞれに対する回答です。これらは、本来は別々の列に配置するべきです。整然データに変換するには、`pivot_wider()` で横長形式に変換します。引数 `names_from` には変換後の列名にする `item` を、`values_from` には変換後の値にする `ans` を指定します(図 2.1)。

コード 2.12 (analysis-tidyr-pivot-wider.R) : 横長形式への変換

```
head(answer, 5)
## # A tibble: 5 × 3
##   id     item     ans
##   <chr> <chr>    <chr>
## 1 1     satisfy 1
## 2 1     apps     <NA>
## 3 1     comment  コメントコメント自由記述コメント自由記述
## 4 2     satisfy 1
## 5 2     apps     HAD;JASP;S-PLUS;SPSS
answer <- 
  tidyr::pivot_wider(answer, names_from = item, values_from = ans)
head(answer, 5)
## # A tibble: 5 × 4
##   id     satisfy apps           comment
##   <chr> <chr>    <chr>        <chr>
## 1 1     1         <NA>        コメントコメント自由記述コメント...
## 2 2     1         HAD;JASP;S-PLUS;SPSS 自由記述コメント自由記述コメント
## 3 3     5         <NA>        コメント自由記述コメントコメント
## 4 4     2         excel;JMP;S-PLUS   自由記述コメント自由記述コメント...
## 5 5     4         <NA>        コメント自由記述
```

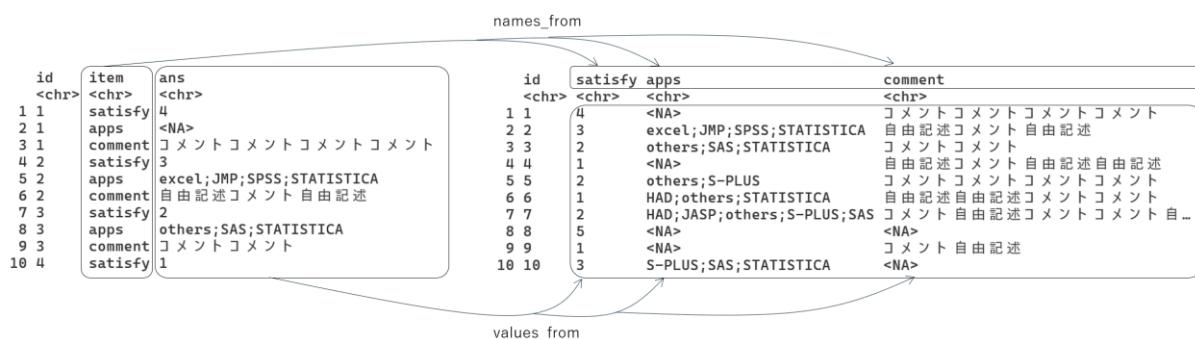


図 2.1: `pivot_wider()` の動作イメージ(左: 前, 右: 後)

`sales` は 1 行に複数の観測が入っています。各行には、うどん、オムライスなどの販売数が入っていますが、これらは別々の観測です。これを整然な縦長形式に変換するには、`pivot_longer()` を使います。引数 `cols` に変換対象の列として `period` と `sheet` 以外を指定します。`names_to` に新たな列名 "item" を、`values_to` に新たな列名 "count" を指定します(図 2.2)。

コード 2.13 (analysis-tidyr-pivot-longer.R) : 縦長形式への変換

```

head(sales, 5)
## # A tibble: 5 × 14
##   period うどん オムライス カツ丼 カレー サンドウィッチ スパゲティ
##   <chr>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 2023-01     47      61      45      46      12      12
## 2 2023-02     32      55      41      31      19      21
## 3 2023-03     44      42      47      46      16      21
## 4 2023-04     48      40      38      46      11      23
## 5 2023-05     37      49      47      44      14      19
## # i 7 more variables: チャーハン <dbl>, ドリア <dbl>, ピザ <dbl>,
## # 蕎麦 <dbl>, 洋定食 <dbl>, 和定食 <dbl>, sheet <chr>
sales <-
tidy::pivot_longer(sales, cols = !c(period, sheet), # ! は以外の意味
                   names_to = "item", values_to = "count")
head(sales, 5)
## # A tibble: 5 × 4
##   period sheet item      count
##   <chr>   <chr> <chr>    <dbl>
## 1 2023-01 芦屋 うどん      47
## 2 2023-01 芦屋 オムライス    61
## 3 2023-01 芦屋 カツ丼      45
## 4 2023-01 芦屋 カレー      46
## 5 2023-01 芦屋 サンドウィッチ    12

```

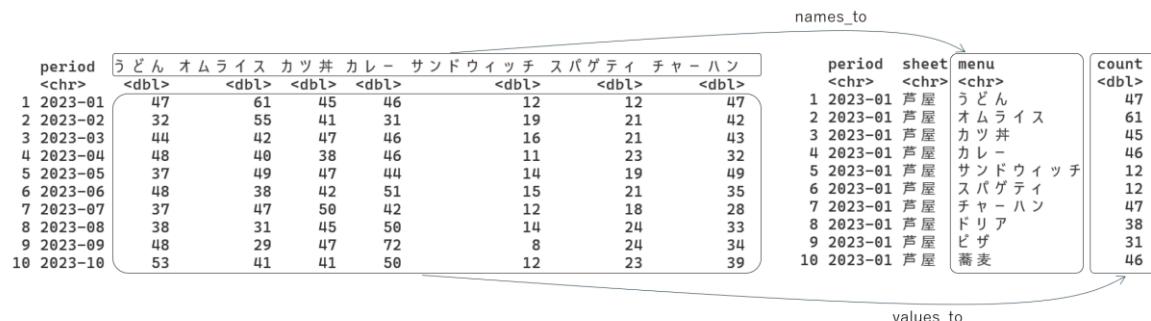


図 2.2: pivot_longer()の動作イメージ(左：前, 右：後)

sales の period のように 1列に年と月が含まれている場合、これらを分割してから集計することができます。文字列を分割するには、separate()を使います。引数 col には分割する前の列名を指定します。into には分割後の列名を文字列で指定します。sep には区切り文字を指定します。sep を指定しないときは、既定値の "[[:alnum:]]+" が使われ、半角の英数字以外の文字が1つ以上という意味の正規表現です。つまり、文字列に半角の英数字以外のものが含まれていると、その文字列で区切られます。なお、日本語の文字列も区切り文字として解釈されます。

コード 2.14 (analysis-tidyr-separate.R) : 列の分割

```

tidy::separate(sales, col = period, into = c("year", "month"), sep = "-") |>
head(3)
## # A tibble: 3 × 5
##   year month sheet item      count
##   <chr> <chr> <chr> <chr>    <dbl>
## 1 2023 01 芦屋 うどん      47
## 2 2023 02 芦屋 オムライス    61
## 3 2023 03 芦屋 カツ丼      45

```

```

## 1 2023 01 芦屋 うどん 47
## 2 2023 01 芦屋 オムライス 61
## 3 2023 01 芦屋 カツ丼 45

```

列を統合するには、`unite()`を使います。統合後の列名を`col`で指定します。統合したい列をその後ろに指定します。`sep`を指定しないと`"_"`(アンダーバー)で結合します。既定値では統合前の列は削除されますが、`remove = FALSE`とすると削除されません。

コード 2.15 (analysis-tidyr-unite.R) : 列の統合

```

tidyr::unite(sales, col = "shop_item", sheet, item, sep = "-") |> head(3)
## # A tibble: 3 × 3
##   period shop_item     count
##   <chr>   <chr>       <dbl>
## 1 2023-01 芦屋-うどん     47
## 2 2023-01 芦屋-オムライス   61
## 3 2023-01 芦屋-カツ丼     45

tidyr::unite(sales, "shop_item", sheet, item, remove = FALSE) |> head(3)
## # A tibble: 3 × 5
##   period shop_item     sheet item     count
##   <chr>   <chr>       <chr> <chr>   <dbl>
## 1 2023-01 芦屋_うどん    芦屋 うどん     47
## 2 2023-01 芦屋_オムライス 芦屋 オムライス   61
## 3 2023-01 芦屋_カツ丼    芦屋 カツ丼     45

```

データフレームの1つのセルに複数のデータが入っていることがあります。たとえばアンケートでの選択肢の複数回答がこれにあてはまることが多く、今回のデータでは`answer`の`apps`の列です。このときには、`separate_rows()`で行(縦)方向に分割できます。引数に分割したい列名を指定します。区切り文字`sep`の既定値は`"[[:alnum:]]+"`ですが、明示的に指定可能です。

コード 2.16 (analysis-tidyr-separate-rows.R) : 列の縦方向への分割

```

answer <- tidyr::separate_rows(answer, apps, sep = ";")
head(answer, 5)
## # A tibble: 5 × 4
##   id   satisfy apps   comment
##   <chr> <chr>   <chr> <chr>
## 1 1     1       <NA>  コメントコメント自由記述コメント自由記述
## 2 2     1       HAD    自由記述コメント自由記述コメント
## 3 2     1       JASP   自由記述コメント自由記述コメント
## 4 2     1       S-PLUS 自由記述コメント自由記述コメント
## 5 2     1       SPSS   自由記述コメント自由記述コメント

```

データに`NA`があるときは、`"-"`や`0`に変換したほうがよいことがあります。`NA`を置換するには`replace_na()`を使います。引数にはリスト形式で、置換対象の列 = 置換後の値を指定します。

コード 2.17 (analysis-tidyr-replace-na.R) : NA の置換

```
answer <- replace_na(answer, list(apps = "-", comment = "-"))
head(answer, 5)
## # A tibble: 5 × 4
##   id   satisfy apps  comment
##   <chr> <chr>   <chr> <chr>
## 1 1     1       -      コメントコメント自由記述コメント自由記述
## 2 2     1       HAD    自由記述コメント自由記述コメント
## 3 2     1       JASP   自由記述コメント自由記述コメント
## 4 2     1       S-PLUS 自由記述コメント自由記述コメント
## 5 2     1       SPSS  自由記述コメント自由記述コメント
```

2.5 データフレームの操作

dplyr はデータフレームを操作するためのパッケージです。列の追加や選択、行の抽出や並べ替え、グループ化、集計などができます。dplyr の関数名や機能の多くは SQL と似ています。なお、tidyr と同様に第 1 引数は対象とするデータフレームのため、説明では省略します。

データ集計では、グループ化して集計することが目的であることが多いです。ただし、その前にデータフレームの結合、列の選択、行の抽出、変換などの操作が必要です。

データフレームを結合するには、`left_join()`などを使用します。前節で整形した `answer` と `attribute` を `id` 列で結合し、`sales` と `unit_price` を `item` 列で結合します。

結合に使う列の指定には、`by` を使用します。指定しなければ、同じ列名で結合します。`by = join_by(foo == bar)` と指定すると、第 1 引数の `foo` 列と第 2 引数の `bar` 列の内容で結合します。`sales` と `unit_price` の結合の例では明示していますが、本来は `by` を指定する必要はありません。また、同じ列名が複数あれば、全ての列が同じ行を結合します。

コード 2.18 (analysis-dplyr-join.R) : データフレームの結合

```
answer <- dplyr::left_join(attribute, answer) # id 列で結合
## Joining with `by = join_by(id)`
head(answer, 3)
## # A tibble: 3 × 6
##   id   area  period satisfy apps  comment
##   <chr> <chr> <chr>   <chr> <chr> <chr>
## 1 1     関東  5      1       -      コメントコメント自由記述コメント自...
## 2 2     関東  19     1       HAD    自由記述コメント自由記述コメント
## 3 2     関東  19     1       JASP   自由記述コメント自由記述コメント
sales <- dplyr::left_join(sales, unit_price, by = join_by(item == item))
head(sales, 3)
## # A tibble: 3 × 5
##   period sheet item      count price
##   <chr>   <chr> <chr>    <dbl> <dbl>
## 1 2023-01 芦屋 うどん      47    500
```

```

## 2 2023-01 芦屋 オムライス   61   600
## 3 2023-01 芦屋 カツ丼       45   650

```

`left_join()`以外にも、第2引数をもとのデータとして結合する `right_join()`、全結合する `full_join()`などがあります。

一方、マッチしない行を返す `anti_join()`などもあります。`anti_join()`は第1引数のデータフレームにあって、第2引数のデータフレームにはない行を返します。そのため、全体の中から漏れているもの、つまり欠落したデータの抽出に使えます。

ここでは `answer` と `attribute` を使います。両方とも `id` は 1 から 100 まで全て揃っていますが、`answer` からわざと一部のデータを欠落させてから、欠落した部分を抽出します。

コード 2.19 (`analysis-dplyr-anti-join.R`) : 漏れ(欠落データ)の抽出

```

lost <- dplyr::filter(answer, apps != "-") # apps == "-" を欠落させる
print(answer, n = 3)
## # A tibble: 275 × 6
##   id    area  period satisfy apps  comment
##   <chr> <chr> <chr>   <chr> <chr> <chr>
## 1 1     関東   5      1      -    コメントコメント自由記述コメント自...
## 2 2     関東   19     1      HAD   自由記述コメント自由記述コメント
## 3 2     関東   19     1      JASP  自由記述コメント自由記述コメント
## # 272 more rows
print(lost, n = 3)
## # A tibble: 260 × 6
##   id    area  period satisfy apps  comment
##   <chr> <chr> <chr>   <chr> <chr> <chr>
## 1 2     関東   19     1      HAD   自由記述コメント自由記述コメント
## 2 2     関東   19     1      JASP  自由記述コメント自由記述コメント
## 3 2     関東   19     1      S-PLUS 自由記述コメント自由記述コメント
## # 257 more rows
dplyr::anti_join(attribute, lost) # attribute にあって、lost にないもの
## Joining with `by = join_by(id, area, period)`
## # A tibble: 15 × 3
##   id    area      period
##   <chr> <chr>      <chr>
## 1 1     関東        5
## 2 3     中部       12
## 3 5     中国・四国  6
## 4 15    近畿        6
## 5 25    九州・沖縄  8
## 6 35    中部       20
## 7 45    中国・四国 18
## 8 49    中部       19
## 9 58    関東       30
## 10 60   中部       4
## 11 61   中国・四国 26
## 12 87   関東       16

```

```

## 13 89 九州・沖縄 2
## 14 98 九州・沖縄 9
## 15 100 中国・四国 17

```

特定の列を選択あるいは除外するには、`select()`を使います。引数には、選択する列名を列挙します。除外するときは、`-か!`を使います。

コード 2.20 (analysis-dplyr-select.R) : 列の選択

```

dplyr::select(answer, id, area, period) |> head(3)
## # A tibble: 3 × 3
##   id    area  period
##   <chr> <chr> <chr>
## 1 1     関東  5
## 2 2     関東  19
## 3 2     関東  19

dplyr::select(sales, -c(period, item)) |> head(3)
## # A tibble: 3 × 3
##   sheet count price
##   <chr> <dbl> <dbl>
## 1 芦屋     47   500
## 2 芦屋     61   600
## 3 芦屋     45   650

```

行を抽出するときは、`filter()`を使います。`satisfy == "5"`のように抽出する条件を指定します。条件には、論理演算の`&`や`|`を使えます。`,`(カンマ)で区切って複数の条件を指定すると、`&`と同様にすべての条件に合致するものが抽出されます。

コード 2.21 (analysis-dplyr-filter.R) : 行を抽出

```

dplyr::filter(answer, satisfy == "5") |> head(3)
## # A tibble: 3 × 6
##   id    area  period satisfy apps  comment
##   <chr> <chr> <chr> <chr> <chr> <chr>
## 1 3     中部  12     5     -     コメント自由記述コメントコメント
## 2 10    関東  8      5     excel 自由記述コメント
## 3 10    関東  8      5     HAD   自由記述コメント

dplyr::filter(sales, 600 < price & price < 700) |> head(3)
## # A tibble: 3 × 5
##   period sheet item       count price
##   <chr>   <chr> <chr>     <dbl> <dbl>
## 1 2023-01 芦屋  カツ丼      45   650
## 2 2023-01 芦屋  チャーハン  47   630
## 3 2023-01 芦屋  和定食      42   650

```

重複を除去するには、`distinct()`を使います。引数で指定した列について重複のないデータが得られます。たとえば、`answer`で`area`だけを指定すると`area`の一覧が得られます。複数の列を指定することも可能です。

コード 2.22 (analysis-dplyr-distinct.R) : 重複の除去

```
dplyr::distinct(answer, area)
## # A tibble: 6 × 1
##   area
##   <chr>
## 1 関東
## 2 中部
## 3 九州・沖縄
## 4 中国・四国
## 5 近畿
## 6 北海道・東北
dplyr::distinct(sales, period, sheet) |>
  print(n = 3)
## # A tibble: 216 × 2
##   period sheet
##   <chr>   <chr>
## 1 2023-01 芦屋
## 2 2023-02 芦屋
## 3 2023-03 芦屋
## # 213 more rows
```

列の内容で並べ替えるには、`arrange()`を使います。複数の列を指定することもできます。また、逆順には`desc()`を使います。

コード 2.23 (analysis-dplyr-arrange.R) : 並べ替え

```
dplyr::arrange(answer, period) |> head(3)
## # A tibble: 3 × 6
##   id    area      period satisfy apps  comment
##   <chr> <chr>     <chr>   <chr>  <chr> <chr>
## 1 54   中国・四国 1       1     excel -
## 2 54   中国・四国 1       1     HAD   -
## 3 54   中国・四国 1       1     SAS   -
dplyr::arrange(sales, desc(count)) |> head(3)
## # A tibble: 3 × 5
##   period sheet item  count price
##   <chr>   <chr> <chr> <dbl> <dbl>
## 1 2024-05 三宮  カレー  92    600
## 2 2023-12 三宮  和定食  89    650
## 3 2023-11 岡本  洋定食  88    700
```

追加した列を移動したいときは、`relocate()`を使います。指定した列だけが前方(左側)に移動します。引数`.before`や`.after`で特定の列の前や後ろへ移動もできます。

コード 2.24 (analysis-dplyr-relocate.R) : 列の順序変更

```

dplyr::relocate(answer, apps) |> head(3)
## # A tibble: 3 × 6
##   apps id    area period satisfy comment
##   <chr> <chr> <chr> <dbl> <chr>   <chr>
## 1 -    1    関東     5      1    コメントコメント自由記述コメント自...
## 2 HAD   2    関東    19      1    自由記述コメント自由記述コメント
## 3 JASP  2    関東    19      1    自由記述コメント自由記述コメント

dplyr::relocate(answer, comment, .before = apps) |> head(3)
## # A tibble: 3 × 6
##   id    area period satisfy comment           apps
##   <chr> <chr> <dbl> <chr>   <chr>   <chr>
## 1 1    関東     5      1    コメントコメント自由記述コメント自... -
## 2 2    関東    19      1    自由記述コメント自由記述コメント     HAD
## 3 2    関東    19      1    自由記述コメント自由記述コメント     JASP

```

列名を変更するには、`rename()`を使います。新しい列名 = 既存の列名として指定します。

コード 2.25 (`analysis-dplyr-rename.R`) : 列名の変更

```

sales <- dplyr::rename(sales, shop = sheet)
head(sales, 3)
## # A tibble: 3 × 5
##   period shop  item      count price
##   <chr>  <chr> <chr>     <dbl> <dbl>
## 1 2023-01 芦屋 うどん      47    500
## 2 2023-01 芦屋 オムライス    61    600
## 3 2023-01 芦屋 カツ丼       45    650

```

列の追加には `mutate()`を使います。既存の列名を指定した場合は、列の元の位置のままでです。新しい列を追加するときに位置を指定しないと、新しい列の場所は最後(右端)になります。また、`relocate()`と同様に、`.before` や `.after` での指定が可能です。

コード 2.26 (`analysis-dplyr-mutate.R`) : 列の追加

```

dplyr::mutate(answer, id = as.numeric(id), period = as.numeric(period))
## # A tibble: 275 × 6
##   id    area      period satisfy apps   comment
##   <dbl> <chr>     <dbl> <chr>   <chr>   <chr>
## 1 1    関東      5     1    -    コメントコメント自由記述...
## 2 2    関東     19     1    HAD   自由記述コメント自由記述...
## 3 2    関東     19     1    JASP   自由記述コメント自由記述...
## 4 2    関東     19     1    S-PLUS 自由記述コメント自由記述...
## 5 2    関東     19     1    SPSS   自由記述コメント自由記述...
## 6 3    中部      12     5    -    コメント自由記述コメント...
## 7 4    九州・沖縄 15     2    excel  自由記述コメント自由記述...
## 8 4    九州・沖縄 15     2    JMP    自由記述コメント自由記述...
## 9 4    九州・沖縄 15     2    S-PLUS 自由記述コメント自由記述...
## 10 5   中国・四国  6     4    -    コメント自由記述
## # i 265 more rows

```

```

answer |>
  dplyr::mutate(ap = stringr::str_sub(apps, 1, 2), .before = 2) |> # 2列目の前
  dplyr::mutate(co = stringr::str_sub(comment, 1, 2), .after = ap)
## # A tibble: 275 × 8
##   id    ap    co    area      period satisfy apps    comment
##   <chr> <chr> <chr> <chr>     <chr>  <chr> <chr> <chr>
## 1 1     -     コメ  関東      5       1     -     コメントコメン...
## 2 2     HA    自由  関東      19      1     HAD   自由記述コメン...
## 3 2     JA    自由  関東      19      1     JASP  自由記述コメン...
## 4 2     S-    自由  関東      19      1     S-PLUS 自由記述コメン...
## 5 2     SP    自由  関東      19      1     SPSS  自由記述コメン...
## 6 3     -     コメ  中部      12      5     -     コメント自由記...
## 7 4     ex    自由  九州・沖縄 15      2     excel 自由記述コメン...
## 8 4     JM    自由  九州・沖縄 15      2     JMP   自由記述コメン...
## 9 4     S-    自由  九州・沖縄 15      2     S-PLUS 自由記述コメン...
## 10 5    -     コメ  中国・四国 6       4     -     コメント自由記述
## # 265 more rows

```

新しい列の追加だけでなく、既存の列の内容をもとにした演算や文字列の操作もできます。

コード 2.27 (analysis-dplyr-mutate-new-col.R) : 列の追加

```

sales <- dplyr::mutate(sales, amount = count * price)
head(sales, 3)
## # A tibble: 3 × 6
##   period shop  item      count  price  amount
##   <chr>   <chr> <chr>     <dbl> <dbl>   <dbl>
## 1 2023-01 芦屋  うどん      47    500   23500
## 2 2023-01 芦屋  オムライス    61    600   36600
## 3 2023-01 芦屋  カツ丼      45    650   29250

```

`mutate_at()`を使えば、指定した列を一括で変換できます。たとえば、`id`, `period`, `satisfy` の各列を数値に変換するには以下のようにします。

コード 2.28 (analysis-dplyr-mutate-at.R) : 指定列の追加

```

answer <- dplyr::mutate_at(answer, c("id", "period", "satisfy"), as.numeric)

```

条件に合致した列のみを変換するには `mutate_if()`を、全ての列を対象に変換するには `mutate_all()`を使います。以下では、`mutate_if()`で `is.numeric` を条件として数値の列のみを100倍します。

コード 2.29 (analysis-dplyr-mutate-if.R) : 指定列の追加

```

dplyr::mutate_if(answer, is.numeric, magrittr::multiply_by, 100) |> head(3)
## # A tibble: 3 × 6
##   id    area  period satisfy apps    comment
##   <dbl> <chr> <dbl>   <dbl> <dbl> <chr>
## 1 1     関東  19      1     100  コメントコメン...
## 2 2     関東  19      1     100  HAD   自由記述コメン...
## 3 2     関東  19      1     100  JASP  自由記述コメン...

```

```

##   <dbl> <chr>  <dbl> <dbl> <chr> <chr>
## 1 100 関東      500     100 -    コメントコメント自由記述コメント自...
## 2 200 関東      1900    100 HAD  自由記述コメント自由記述コメント
## 3 200 関東      1900    100 JASP 自由記述コメント自由記述コメント

```

`transmute()`は `mutate()`と似ていますが、指定しない列は削除されます。なお、`mutate_*`()と同様の関数として、`transmute_at()`や`transmute_if()`があります。

コード 2.30 (analysis-dplyr-transmute.R) : 列の追加と選択

```

dplyr::transmute(sales, item = stringr::str_sub(item, 1, 2), count) |> head(5)
## # A tibble: 5 × 2
##   item  count
##   <chr> <dbl>
## 1 うど     47
## 2 オム     61
## 3 カツ     45
## 4 カレ     46
## 5 サン     12

```

2.6 データのグループ化と集計

データ集計の多くの場合で、区分ごとにグループ化してデータを集計します。グループ化するには`group_by()`を、集計には`summarise()`を使います。`group_by()`でグループ化すると、グループ化した列(Groups)とグループ数が表示されます。

コード 2.31 (analysis-dplyr-group-by.R) : グループ化

```

dplyr::group_by(answer, area) |> print(n = 3)
## # A tibble: 275 × 6
## # Groups:   area [6]
##       id area period satisfy apps  comment
##   <dbl> <chr> <dbl>   <dbl> <chr> <chr>
## 1     1 関東     5       1 -    コメントコメント自由記述コメント自...
## 2     2 関東     19      1 HAD  自由記述コメント自由記述コメント
## 3     2 関東     19      1 JASP 自由記述コメント自由記述コメント
## # 272 more rows
dplyr::group_by(sales, item) |> print(n = 3)
## # A tibble: 2,592 × 6
## # Groups:   item [12]
##   period shop  item      count  price amount
##   <chr>   <chr> <chr>    <dbl> <dbl>  <dbl>
## 1 2023-01 芦屋  うどん     47    500  23500
## 2 2023-01 芦屋  オムライス   61    600  36600
## 3 2023-01 芦屋  カツ丼     45    650  29250
## # 2,589 more rows

```

`group_by()`と`summarise()`を合わせて使うと真価を發揮します。各グループの平均値や最大値などの集計が可能です。

コード 2.32 (analysis-dplyr-summarise.R) : 平均や最大値などの集計

```
dplyr::group_by(answer, area) |>
  dplyr::summarise(m_period = mean(period), m_satisfy = mean(satisfy))
## # A tibble: 6 × 3
##   area      m_period m_satisfy
##   <chr>        <dbl>     <dbl>
## 1 中国・四国    9.91     2.33
## 2 中部          8.6      2.6
## 3 九州・沖縄    8.17     3.24
## 4 北海道・東北    7       2.38
## 5 近畿          7.98     2.75
## 6 関東         12.3     2.95
```

なお、`summarise()`で引数`.by`を指定することで、`goup_by()`と同じことができます。1回限りの集計でグループを使うときには、この方が簡潔な記載ができます。

コード 2.33 (analysis-dplyr-summarise-by.R) : .by を使った平均や最大値などの集計

```
dplyr::summarise(answer, m_period = mean(period), m_satisfy = mean(satisfy),
                  .by = area)
## # A tibble: 6 × 3
##   area      m_period m_satisfy
##   <chr>        <dbl>     <dbl>
## 1 関東         12.3     2.95
## 2 中部          8.6      2.6
## 3 九州・沖縄    8.17     3.24
## 4 中国・四国    9.91     2.33
## 5 近畿          7.98     2.75
## 6 北海道・東北    7       2.38
```

`summarise()`に似た関数に、`summarise_at()`, `summarise_if()`, `summarise_all()`があります。これらを使えば、個別に列を指定しなくても複数の列を一度に集計できます。たとえば、数値の列の最大値を求めるには次のようにします。

コード 2.34 (analysis-dplyr-summarise-if.R) : 数値の列の集計

```
dplyr::group_by(sales, item) |>
  dplyr::summarise_if(is.numeric, max)
## # A tibble: 12 × 4
##   item      count price amount
##   <chr>     <dbl> <dbl>  <dbl>
## 1 うどん      80    500  40000
```

```

## 2 オムライス      75  600  45000
## 3 カツ丼          85  650  55250
## 4 カレー         92  600  55200
## 5 サンドウィッチ   35  450  15750
## 6 スパゲティ     45  710  31950
## 7 チャーハン      65  630  40950
## 8 ドリア         59  700  41300
## 9 ピザ           54  740  39960
## 10 和定食        89  650  57850
## 11 洋定食        88  700  61600
## 12 蕎麦          77  500  38500

```

グループごとの個数を数える集計は `n()`で可能です。

コード 2.35 (analysis-dplyr-n.R) : 個数を数える

```

dplyr::group_by(answer, area) |>
  dplyr::summarise(n = n())
## # A tibble: 6 × 2
##       area     n
##   <chr>   <int>
## 1 中国・四国     69
## 2 中部        30
## 3 九州・沖縄    41
## 4 北海道・東北    21
## 5 近畿        56
## 6 関東        58

```

`n()`は頻繁に使うので、ショートカットがあります。単純なショートカットは `tally()`で、`summarise(n = n())`と同じ機能です。`count()`は `group_by()`も含めたショートカットで、引数で指定した列をグループ化して個数を数えます。また、引数 `wt` を指定すると、指定した列の値を合計した結果が得られます。

コード 2.36 (analysis-dplyr-tally.R) : 個数を数えるショートカット

```

dplyr::group_by(answer, area) |>
  dplyr::tally() |> head(3)
## # A tibble: 3 × 2
##       area     n
##   <chr>   <int>
## 1 中国・四国     69
## 2 中部        30
## 3 九州・沖縄    41
dplyr::count(answer, area) |> head(3)
## # A tibble: 3 × 2
##       area     n
##   <chr>   <int>
## 1 中国・四国     69

```

```

## 2 中部      30
## 3 九州・沖縄   41
dplyr::count(sales, shop, wt = count) |> head(3)
## # A tibble: 3 × 2
##   shop     n
##   <chr> <dbl>
## 1 三宮  15396
## 2 三田   7317
## 3 元町  14319

```

グループ別で集計して、さらに `pivot_wider()`などで見やすく表示することもできます。たとえば、`sales` データの売り上げを店舗と月ごとで集計してから、店舗を横方向に展開します。すると縦方向に年月、横方向に店舗名が並びます。

コード 2.37 (`analysis-dplyr-summarise-pivot-wider.R`) : 集計後の表示変更

```

sales |>
  dplyr::summarise(sum = round(sum(amount) / 1000), .by = c(period, shop)) |>
  tidyr::pivot_wider(names_from = shop, values_from = sum)
## # A tibble: 24 × 10
##   period 芦屋 岡本 元町 御影 三宮 三田 西宮 姫路 六甲
##   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 2023-01  297   310   402   341   386   187   219   130   311
## 2 2023-02  269   294   378   387   377   190   226   124   308
## 3 2023-03  293   309   390   372   401   191   215   114   294
## 4 2023-04  275   307   379   375   412   202   233   105   317
## 5 2023-05  285   344   345   370   401   176   232   114   343
## 6 2023-06  288   318   390   350   402   202   236   131   346
## 7 2023-07  254   309   382   365   433   201   256   124   345
## 8 2023-08  274   311   376   369   369   190   239   118   320
## 9 2023-09  283   292   377   358   432   196   257   128   363
## 10 2023-10 277   285   384   383   394   192   227   125   323
## # i 14 more rows

```

`dplyr` の関数を使えば、アンケートの複数回答を集計する関数をつくれます。複数回答は1つのセルの中に、`;`(セミコロン)などの区切り文字で結合した回答が入っていることが多いので、`separate_rows()`で縦方向に区切れます。また、`filter()`で空("")や NA は除外します。その後、グループ化して個数を数えます。

コード 2.38 (`analysis-dplyr-count-multi-ans-fun.R`) : 複数回答を集計する関数

```

count_multi <- function(df, col, sep = "[^[:alnum:]]+", group_add = TRUE){
  df |>
    tidyr::separate_rows(tidyselect::all_of(col), sep = sep) |> # 縦に分割
    dplyr::filter({{ col }} != "") |>                                # 空を除去
    dplyr::filter(!is.na({{ col }})) |>                                # NA を除去
    dplyr::group_by(dplyr::pick({{ col }}), .add = group_add) |> # グループ化
    dplyr::tally() |>                                                 # 個数
}

```

```
dplyr::filter({{ col }} != "") # 空を除去
}
```

《Tips》 filter()とgroup_by()の中で{{ col }}やpick({{ col }})として列名を指定しています。これはdplyrでの独特な表現です。{{ }}は整理された選択と言われるもので、データフレームの変数を選択する際に使います。通常のプログラミング時には、入力や表示を簡潔にするためにfilter(apps != "")のappsのようにダブルクオーテーションでは囲まずに列名を指定します。逆に、列名に文字列を使うとうまく動作しません。これへの対応が{{ }}です。さらにgroup_by()では、pick()を使っています。これはデータマスキングされた変数を計算するときに使います。詳細は Wickham, Çetinkaya-Rundel, and Grolemund (2024) を参考にしてください。

整然データであれば、アンケートの単数回答や複数回答の単純集計やクロス集計はすぐにできます。

コード 2.39 (analysis-dplyr-straight-summary.R) : 単純集計

```
dplyr::count(answer, area) # 単数回答・単純集計
## # A tibble: 6 × 2
##   area          n
##   <chr>      <int>
## 1 中国・四国    69
## 2 中部        30
## 3 九州・沖縄    41
## 4 北海道・東北    21
## 5 近畿        56
## 6 関東        58
count_multi(answer, "apps") # 複数回答・単純集計
## # A tibble: 11 × 2
##   apps          n
##   <chr>      <int>
## 1 ""         30
## 2 "HAD"      34
## 3 "JASP"     25
## 4 "JMP"      16
## 5 "PLUS"     28
## 6 "S"        28
## 7 "SAS"      39
## 8 "SPSS"     36
## 9 "STATISTICA" 31
## 10 "excel"    38
## 11 "others"   13
```

クロス集計した表は縦に長いので、pivot_wider()で見やすくします。

コード 2.40 (analysis-dplyr-cross-summary.R) : クロス集計

```
dplyr::count(answer, area, satisfy) |> # 単数回答・クロス集計
tidyverse::pivot_wider(names_from = area, values_from = n, values_fill = 0)
## # A tibble: 5 × 7
```

```

##   satisfy `中国・四国` 中部 `九州・沖縄` `北海道・東北` 近畿 関東
##   <dbl>     <int> <int>     <int>     <int> <int> <int>
## 1 1          18    7       3        9    17    17
## 2 2          15    8      11       6    7     5
## 3 3          31    7       7       0    10    15
## 4 4          5     6      13       1    17     6
## 5 5          0     2       7       5     5    15
answer |> # 複数回答・クロス集計
dplyr::group_by(area) |>
count_multi("apps", sep = ";") |>
tidyr::pivot_wider(names_from = area, values_from = n, values_fill = 0)
## # A tibble: 10 × 7
##   apps   `中国・四国` 中部 `九州・沖縄` `北海道・東北` 近畿 関東
##   <chr>     <int> <int>     <int>     <int> <int> <int>
## 1 -          4     4       3       0     1     3
## 2 HAD         9     5       4       2     8     6
## 3 JASP        3     3       4       2     6     7
## 4 JMP         4     1       2       1     3     5
## 5 S-PLUS      7     2       4       0     7     8
## 6 SAS         9     5       7       4     7     7
## 7 SPSS        9     5       4       3     7     8
## 8 STATIST...   8     1       6       4     7     5
## 9 excel       11    3       5       5     7     7
## 10 others     5     1       2       0     3     2

```

集計した結果をファイルとして保存するときは、CSVであれば `readr` の `write_csv()` を、エクセルであれば `openxlsx` の `write.xlsx()` を使うと良いでしょう。 詳細は 9.4 節を参照してください。

2.7 洗練された作図

集計した結果をわかりやすく図示して、その図を保存することができます。 そのときには、`ggplot2` を活用します。`ggplot2` は、作図環境を提供するパッケージで、統一的な操作で洗練された作図が可能です。 同一のデータをもとに異なる形式のグラフを楽に作図できます。 グループ分けや全体の見栄えの変更も簡単です。 また、`ggplot2` を拡張するパッケージも多くあります。

本節では `ggplot2` の基本的な使い方を説明しますが、詳細は Wickham (2012) を参照してください。

2.7.1 基本的な作図

`ggplot2` で作図するときには、`ggplot()` を最初に使います。`ggplot()` は作図に使用するデータフレームを第1引数にとります。 次のコード 2.41 ではパイプを使っているため、第1引数を省略しています。 第2引数の `mapping` と `aes()` の説明は長くなるので、とりあえずおまじないとして捉えてください。 詳しくはコードの後ろで説明します。 x 軸に `item` を、y 軸に `count` を使正在することが分かれば大丈夫です。

ggplot2では、小さな命令を積み重ねて作図していきます。そのときに命令を追加するのが+です。次のコードでは `ggplot()`でデータ、x軸、y軸を指定するだけで、具体的な描画方法は示しません。`geom_boxplot()`で箱ひげ図を指定しています。コードを実行すると箱ひげ図を作図できます(図 2.3)。

コード 2.41 (analysis-ggplot-ggplot.R) : 基本的な描画(箱ひげ図)

```
sales |>
  ggplot2::ggplot(ggplot2::aes(x = item, y = count)) +
  ggplot2::geom_boxplot()
```

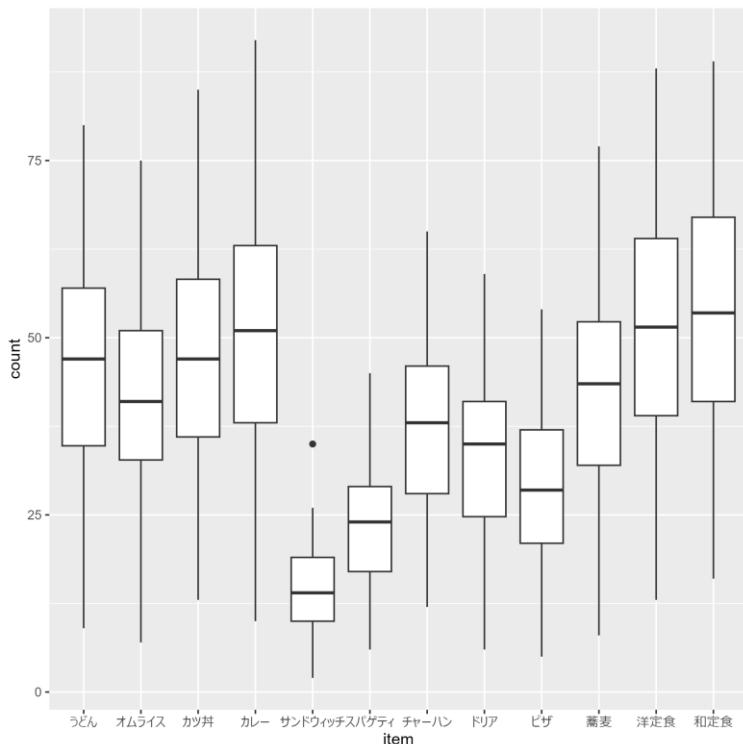


図 2.3: 基本的な描画(箱ひげ図)

`ggplot()`の第 2 引数には、`mapping` として `aes()` 関数の返り値を使います。`mapping` とは作図でのデータの割り当て方法です。データを x 軸、y 軸、色、凡例のような作図要素のどれに割り当てるのかを決めます。この割り当ての指定では、`aes()`を使います。`aes` は `aesthetic` の略で「審美的」と翻訳されますが、いわゆる見た目をどのようにするかという意味です。つまり、`mapping` で作図の見た目を決めます。上のコードでは引数を省略せずに書いていますが、`mapping` や `aes()`の中の `x` と `y` は省略して、以下のように記載するのが一般的です。

```
ggplot(aes(item, count)) # 一般的な記載方法
```

2.7.2 描画方法の変更

ggplot2 には多くの作図関数があり、`geom_boxplot()`の部分を変更すれば、別の形式で作図可能です(表 2.2)。帯グラフ・棒グラフは `geom_bar()`で、折れ線グラフは `geom_line()`で描画できます。他にも多くの描画手法があります。

表 2.2: ggplot2 の主な作図関数

関数	役割
geom_bar()	棒グラフ
geom_density()	密度分布
geom_histogram()	ヒストグラム
geom_jitter()	ジッター・プロット
geom_line()	線
geom_point()	散布図
geom_quantile()	分位点回帰
geom_smooth()	平滑化線
geom_text()	文字列
geom_violin()	バイオリン・プロット

ここでは、ばらつきを与える geom_jitter() で作図します(図 2.4).

コード 2.42 (analysis-ggplot-geom-jitter.R) : ジッター・プロット

```
sales |>
  ggplot2::ggplot(ggplot2::aes(item, count)) +
  ggplot2::geom_jitter()
```

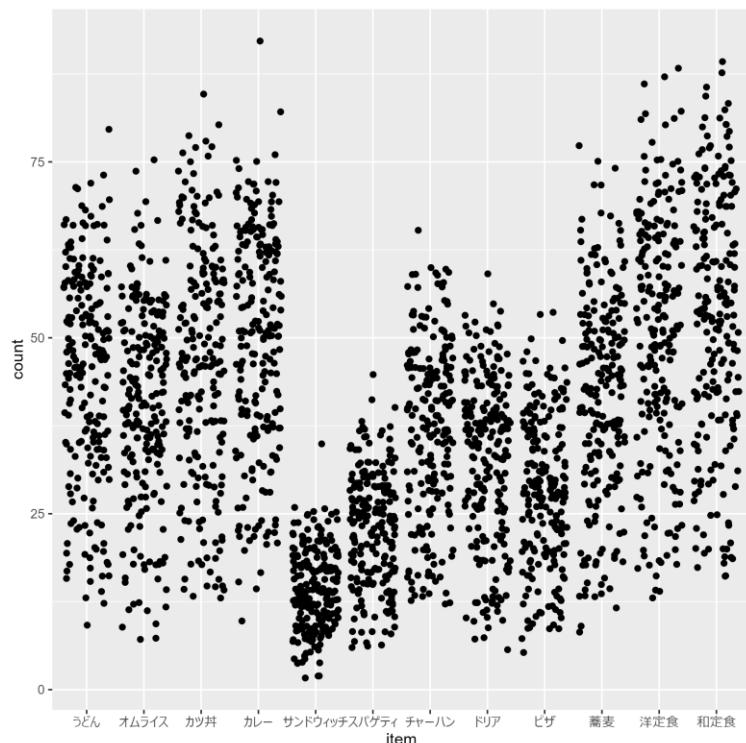


図 2.4: ジッター・プロット

作図方法の変更だけでなく、追加も可能です。`geom_boxplot()`に`geom_jitter()`を追加すると、全体の要約とともに個々のデータもわかります。なお、外れ値の重複を防ぐため、次のコードの`geom_boxplot()`では引数を追加しています。 x 軸(width)と y 軸(height)のばらつきの程度、個々の点の大きさ(size)を設定します。さらに、 x 軸のラベルの重なりを`guides(x = guide_axis(n.dodge = 2))`で防ぎます。次のコードを実行すると、箱ひげ図とジッター・プロットを重ね合わせたものが作図できます(図 2.5)。

コード 2.43 (analysis-ggplot-geom-boxplot-geom-jitter.R) : 箱ひげ図とジッター・プロットの重ね合わせ

```
gg_sales <-
  sales |>
  ggplot2::ggplot(ggplot2::aes(item, count)) +
  ggplot2::geom_boxplot(outlier.color = NA) + # 外れ値の重複を防止
  ggplot2::geom_jitter(width = 0.3, height = 0, size = 0.3) +
  ggplot2::guides(x = ggplot2::guide_axis(n.dodge = 2)) # x 軸の重なり防止
gg_sales
```

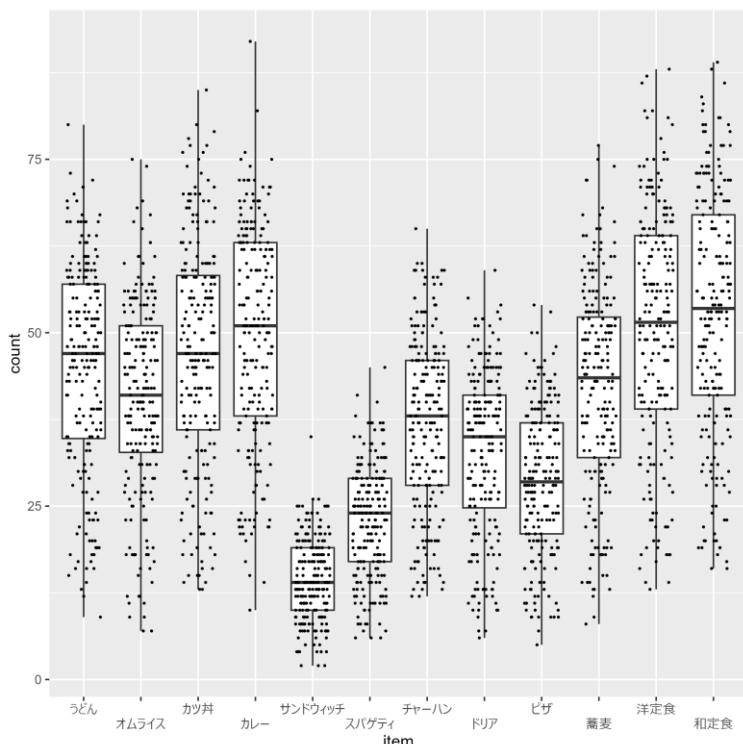


図 2.5: 箱ひげ図とジッター・プロットの重ね合わせ

ggplotで作図したものは作図パネルに表示されますが、オブジェクトとして保存しておくことができます。オブジェクト名そのもので、作図したオブジェクトを図示できます。また、次項で説明するように、保存したオブジェクトの変更あるいはpngやPDFとしての保存が可能です。

2.7.3 テーマの変更

作図全体のテーマを変更したいときには、`theme_*`()を使います。テーマとは、タイトル、軸ラベル、凡例、背景などのデータには直接関係しない部分です。`ggplot2`には、組み込みテーマとして`theme_gray()`, `theme_bw()`, `theme_linedraw()`, `theme_light()`, `theme_dark()`などがあります。たとえば、`theme_bw()`はシンプルな白黒の描画をします(図 2.6)。

コード 2.44 (`analysis-ggplot-theme.R`) : テーマの変更

```
gg_sales + ggplot2::theme_bw()
```

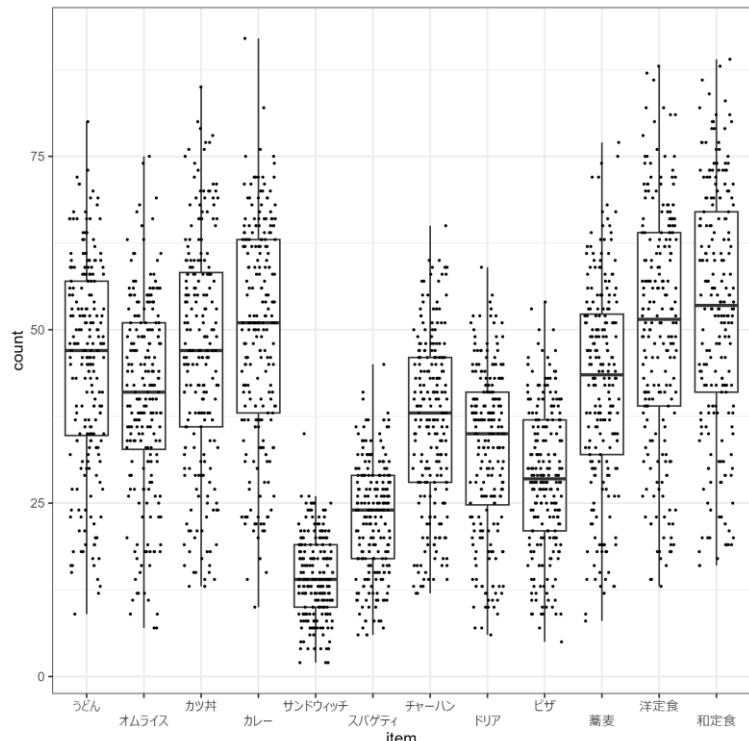


図 2.6: テーマを変更した図

2.7.4 facet で分割

`facet` は、もともとは宝石をカットしたときの面を指すことばですが、`ggplot2` では全体のデータを分割して表示する意味で使われます。つまり、1つのデータを色々な側面から分析しようという意図です。たとえば、`sales` の全店舗を1つのプロットではなく、店舗別に作図するのが `facet` です。

`plot()` 関数で実行するなら `for` と `subset()` で以下のようにするでしょう。なお、`plot()` で x 軸を要素にすると箱ひげ図が作成されます(図 2.7)。

コード 2.45 (`analysis-ggplot-facet-for.R`) : `for` と `subset()` を使った散布図の分割

```
par(mfcol = c(3, 3))
par(mar = rep(0.1, 4))
```

```

par(oma = rep(0.1, 4))
for(s in unique(sales$shop)){
  sales_sub <- subset(sales, shop == s)
  plot(factor(sales_sub$item), sales_sub$count)
}
## png
## 2

```

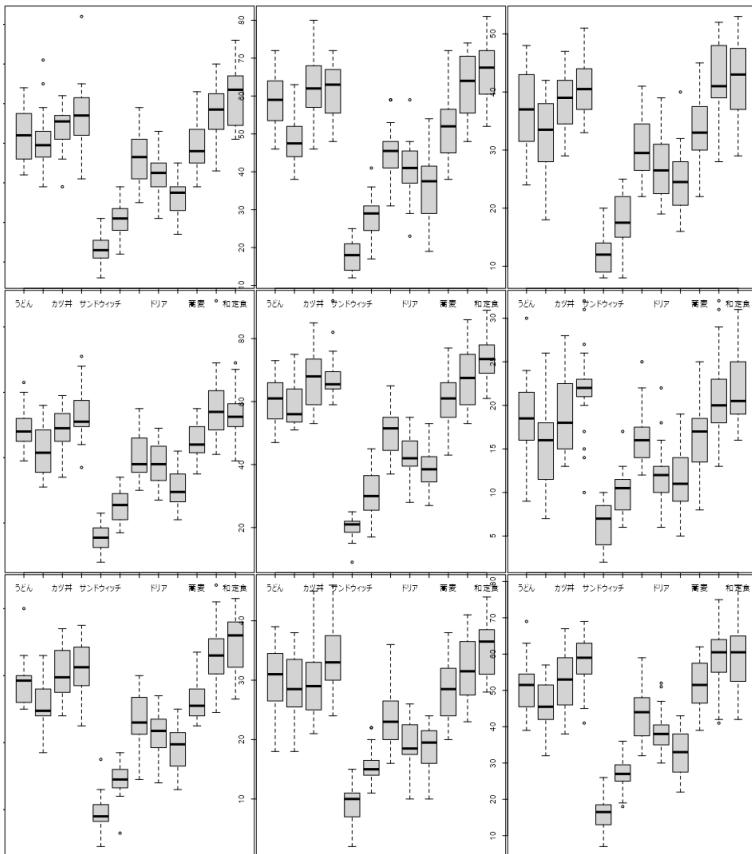


図 2.7: for と subset()で分割した散布図

ggplot2 では、以下のように ggplot2::facet_wrap(vars(shop))を追加するだけです。次のコードで店舗別に作図できます(図 2.8)。

コード 2.46 (analysis-ggplot-facet-wrap.R) : facet による分割して作図

```

sales |>
  ggplot2::ggplot(ggplot2::aes(item, count)) +
  ggplot2::geom_boxplot() +
  ggplot2::facet_wrap(vars/shop))

```

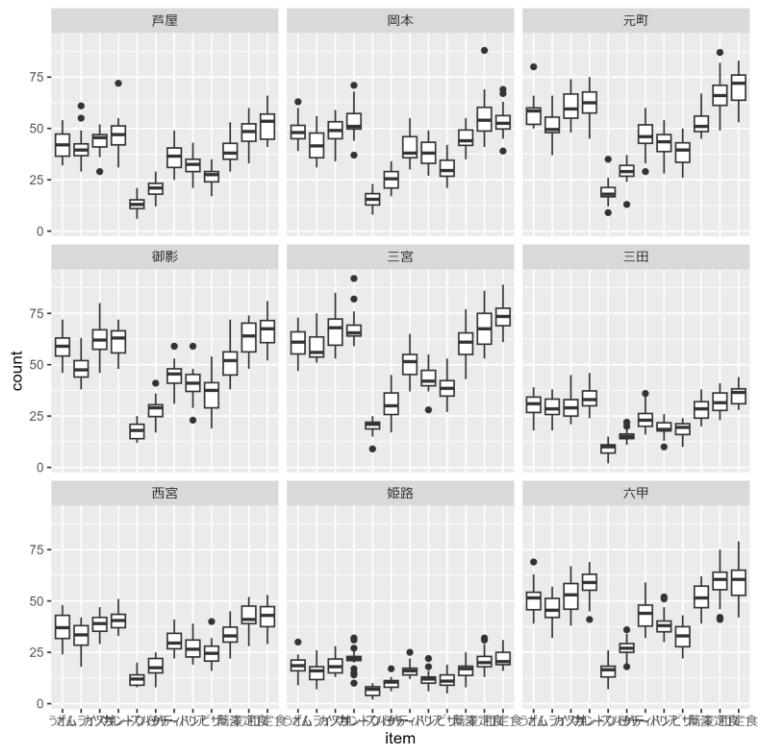


図 2.8: facet で分割した散布図

コードの量はほぼ同じですが、ggplot2の方が変更は容易です。たとえば、既に作成した ggplot オブジェクトのテーマを変更して、さらにファセットを追加するには次のようにします。テーマの変更とファセットが簡単にできます(図 2.9)。

コード 2.47 (analysis-ggplot-facet-wrap-theme.R) : テーマの変更とファセットの追加

```
gg_sales +
  ggplot2::theme_bw() +
  ggplot2::facet_wrap(vars(shop))
```

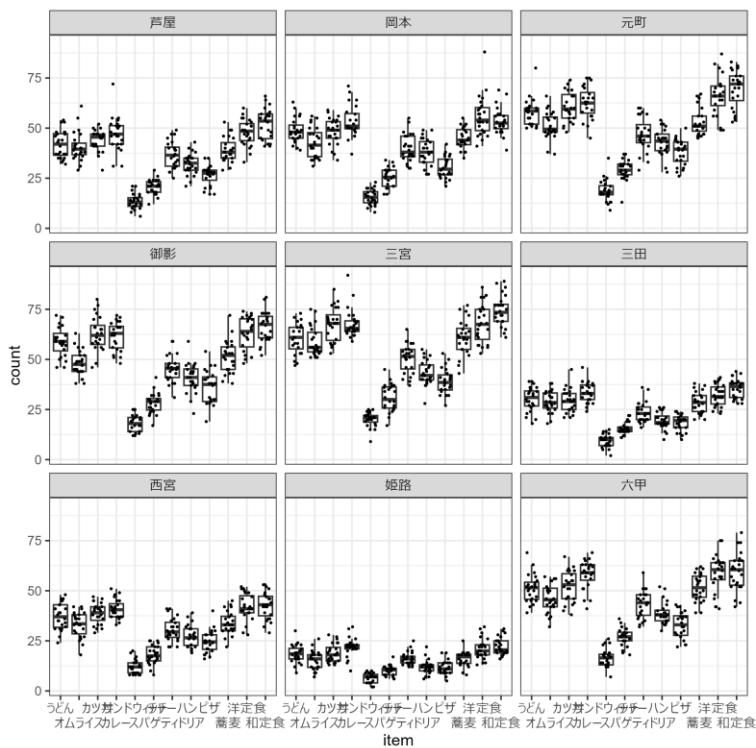


図 2.9: テーマの変更とファセットを追加した作図

2.7.5 作図のファイルへの保存

ggplot2で作図したオブジェクトは、`ggsave()`でファイルに書き込みます。第1引数にパス、第2引数に作図オブジェクトを指定します。作図オブジェクトを指定しなければ、直前の作図が書き込まれます。`png`と`PDF`が使用可能で、形式はパスの拡張子で自動判別します。

使用環境によっては、`PDF`で日本語が文字化けすることがあります。その場合は、次項の文字化け対策を参考にするか、`png`で保存してください。

コード 2.48 (`analysis-ggplot-ggsave.R`) : 作図のファイルへの保存

```
path <- fs::file_temp(ext = "png")
ggplot2::ggsave(path, gg_sales)
## Saving 6 x 4 in image
```

2.7.6 文字化け対策

作字したものを`PDF`で保存すると、Windowsでは日本語が文字化けした状態で表示されることがあります。たとえば、`gg_sales`を保存して図 2.10 のようになるときです。

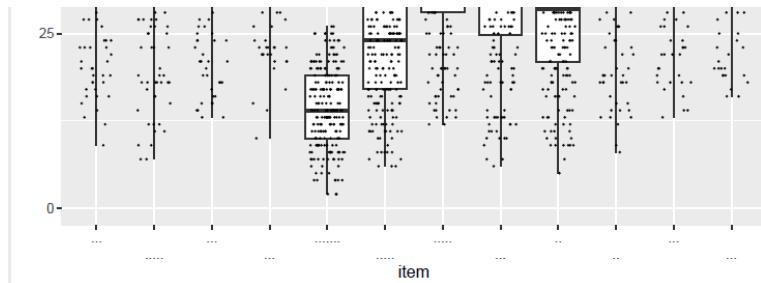


図 2.10: 文字化けした x 軸の日本語

このときは、extrafont パッケージと Cairo パッケージを使います。

コード 2.49 (analysis-extrafont.R) : extrafont と Cairo のインストールと呼び出し

```
install.packages("extrafont")
install.packages("Cairo")
library(extrafont)
library(Cairo)
```

まずは、`font_import()`でフォントをインポートします。Continue?とのメッセージに `y` を入力するとインポート作業が始まります。数分から 10 分程度かかりますので、気長に待ちます。

コード 2.50 (analysis-extrafont-font-import.R) : フォントのインポート

```
extrafont::font_import()
## Importing fonts may take a few minutes, depending on the number of fonts and the speed of the system.
## Continue? [y/n] y
## Scanning ttf files in C:\Windows\Fonts, C:\Users\USERNAME\AppData\Local\Microsoft\Windows\Fonts ...
## Extracting .afm files from .ttf files...
## (以下省略)
```

インポートが完了したら、`loadfonts()`でフォントを登録します。

コード 2.51 (analysis-extrafont-loadfonts.R) : フォントの登録

```
extrafont::loadfonts()
## Registering font with R using windowsFonts(): Agency FB
## Registering font with R using windowsFonts(): Algerian
## Registering font with R using windowsFonts(): Arial Black
## Registering font with R using windowsFonts(): Arial
## Registering font with R using windowsFonts(): Arial Narrow
## (以下省略)
```

フォントのインポートと登録は 1 回実行するだけで大丈夫です。ただし、R を再起動したときには、`library(extrafont)`を実行する必要があります。

登録できれば、利用可能なフォントを確認します。使用環境によって表示されるフォントは異なります。多くのフォントが表示されるときは、`str_subset()`で絞り込んでください。

コード 2.52 (`analysis-extrafont-font.R`) : 利用可能なフォントの確認

```
extrafont::fonts()
## [1] "Agency FB"          "Algerian"        "Arial Black"
## [4] "Arial"              "Arial Narrow"     "Arial Rounded MT Bold"
## (中略)
## [205] "ZWAdobeF"
extrafont::fonts() |>
  stringr::str_subset("Yu")
## [1] "Yu Mincho Demibold" "Yu Mincho Light"    "Yu Mincho"
```

日本語フォントを使うには、`theme(text = element_text("フォント名"))`のように利用可能なフォントを指定します。また、`ggsave(device = cairo_pdf)`とします。

コード 2.53 (`analysis-ggplot-notofu.R`) : PDF ファイルの文字化け対策

```
# library(extrafont) # 再起動時には必要
library(Cairo)
gg_sales_cairo <-
  gg_sales +
  ggplot2::theme(text = ggplot2::element_text(family = "Yu Mincho"))
path <- fs::file_temp(ext = "pdf")
ggplot2::ggsave(path, gg_sales_cairo,
  device = cairo_pdf, width = 7, height = 7)
# shell.exec(path)
```

正しいフォントを指定すれば、文字化けしなくなります(図 2.11)。

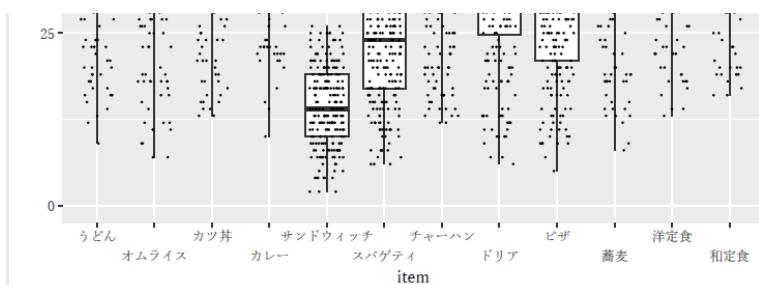


図 2.11: 文字化け対策した PDF ファイル

`geom_text()`でプロット内に日本語を入れるときには、`family = "フォント名"`のようにします。その後、`ggsave()`で `device = cairo_pdf` とすれば文字化けしません(図 2.12)。

コード 2.54 (`analysis-geom-text.R`) : プロット内に日本語を入れる

```
tibble::tibble(x = 1:5, y = 1:5, label = c("あ", "い", "う", "え", "お")) |>
  ggplot2::ggplot(aes(x, y, label = label)) +
  ggplot2::geom_text(family = "Yu Mincho", size = 10)
path <- fs::file_temp(ext = "pdf")
ggplot2::ggsave(path, device = cairo_pdf)
# shell.exec(path)
```

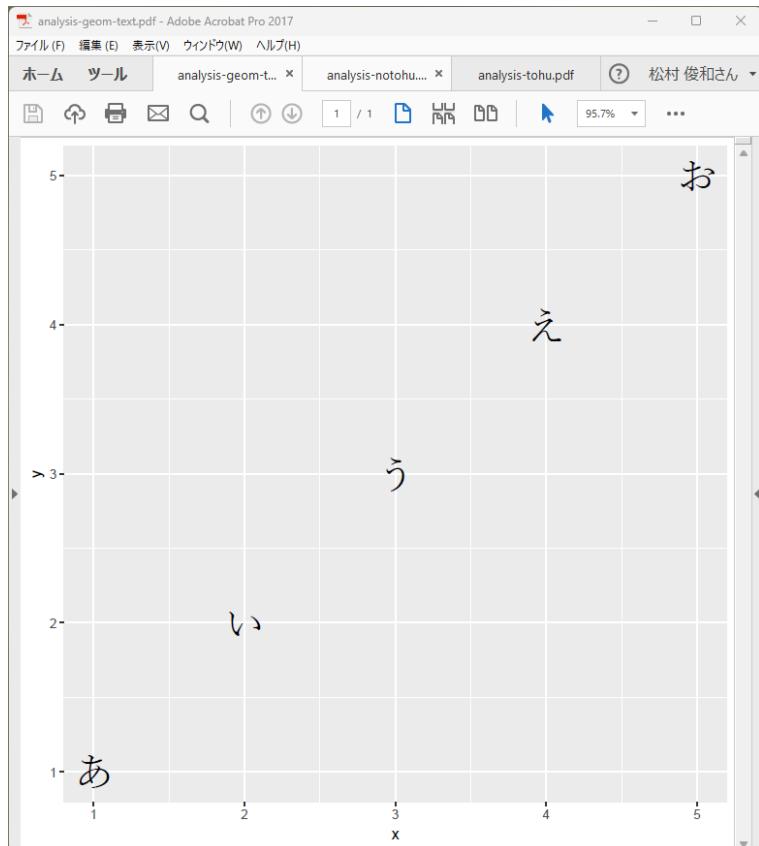


図 2.12: プロット内に日本語を入れた図

《Tips》 RStudio での作図時に文字化けするときは、以下の設定を変更すると文字化けしなくなる可能性があります。メニューの[Tools] - [Global Options] - [General] - [Graphics] - [Backend] (図 2.13)で、AGG を選択すると著者の環境では文字化けしなくなりました。

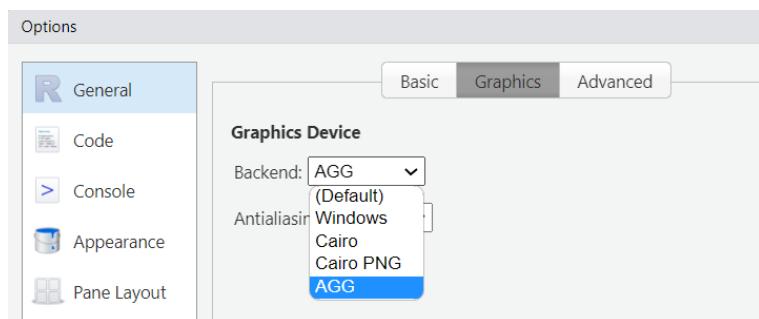


図 2.13: RStudio の Global Option の設定

2.8 繰り返し処理

アンケート結果を属性ごとに分けて集計したり、売上げデータの店舗ごとに図示したりという繰り返し作業をすることがあります。2-3回程度の繰り返してあれば、似たようなコードを複写して

から修正すれば構いませんが、何度も同じ作業をするのは面倒です。また、パイプの途中で `for` や `while` によるループは使えません。簡潔な記述でないことも難点です。

パイプで繰り返しをするには `purrr` パッケージの `map()` を使います。`map()` を使うと、連続的かつ簡潔なコードが書けます。

データフレームを使うときには、特定の列の値で分割したリストにすることが多いです。たとえば、アンケート結果を地域で区分するには、`split()` を使います。

コード 2.55 (`analysis-purrr-split-area.R`) : データフレームのリストへの分割

```
split(answer, answer$area)
## $関東
## # A tibble: 22 × 6
##       id area  period satisfy apps      comment
##   <dbl> <chr> <dbl>    <dbl> <chr>
## 1     1 関東      5        1 -
## 2     2 関東     19        1 HAD;JASP;S-PLUS;SPSS
## 3     8 関東     20        4 SAS
## # (中略)
## $近畿
## # A tibble: 20 × 6
##       id area  period satisfy apps      comment
##   <dbl> <chr> <dbl>    <dbl> <chr>
## 1     7 近畿      8        4 excel;JASP;S-PLUS;SPSS
## 2    15 近畿      6        4 -
## 3    19 近畿      8        2 excel;STATISTICA
## # (以下省略)
```

ただし、`answer$area` のように指定するのは面倒なので、パイプ中でも使いやすくする関数を定義します。

コード 2.56 (`analysis-purrr-split-by-fun.R`) : リストに分割する関数の定義

```
split_by <- function(df, group){
  split(df, df[[group]])
}
```

`map()` は、第1引数にベクトルかリストを、第2引数に関数を指定します。第2引数の関数に引数がある場合は、第2引数のうしろに記載します。次のコードでは、アンケートの結果の複数回答を地域ごとに集計して、回答を降順で並べ替えます。

コード 2.57 (`analysis-purrr-answer.R`) : 集計の繰り返し

```
answer |>
  split_by("area") |>
```

```

purrr::map(count_multi, "apps", ";") |>
purrr::map(dplyr::arrange, desc(n))
## $関東
## # A tibble: 10 × 2
##   apps      n
##   <chr>    <int>
## 1 S-PLUS      8
## 2 SPSS        8
## 3 JASP        7
## (中略)
## $近畿
## # A tibble: 10 × 2
##   apps      n
##   <chr>    <int>
## 1 HAD        8
## 2 S-PLUS      7
## 3 SAS        7
## (以下省略)

```

作図を繰り返すこともできます。たとえば、売上データを店舗ごとに分割し、その後に作図します。作図では、`x`軸に `period` を、`y`軸に `count` を、グループに `item` を指定します。折れ線グラフでは、`item` を色別で描画するように指定します。白黒のテーマとフォントとタイトルの設定もします。

また、分割したときの `shop` の名前を必要とするため、`map()`の派生形である `imap()`を使っています。`imap()`の関数内では、`.x` と `.y` は特別な意味を持ちます。`imap()`の第1引数を `arg` とすると、`.x` は `arg[[i]]`(`i` は序数)、`.y` は `names(arg)[[i]]` の意味です。つまり次のコードでは、`.x` は分割したデータフレーム、`.y` は"関東"などの地域名です。

コード 2.58 (analysis-purrr-split-imap.R) : 作図の繰り返し

```

gg_sales_split <-
sales |>
split_by("shop") |>
purrr::imap(
  \(.x, .y){
    ggplot2::ggplot(.x, ggplot2::aes(period, count, group = item)) +
    ggplot2::geom_line(aes(color = item)) +
    ggplot2::theme_bw() +
    ggplot2::theme(text = ggplot2::element_text(family = "Yu Mincho")) +
    ggplot2::labs(title = .y)
  }
)

```

コード 2.59 (analysis-purrr-split-imap-gg.R) : リストになったグラフの表示

```
gg_sales_split[[2]] # 2番目のグラフ
```

`gg_sales_split` には 9 つの店舗のグラフが入っています。 RStudio では、`gg_sales_split` とすると、全てのグラフが順番に表示されます。 2 番目のグラフのみを表示するには、`gg_sales_split[[2]]` とします(図 2.14)。

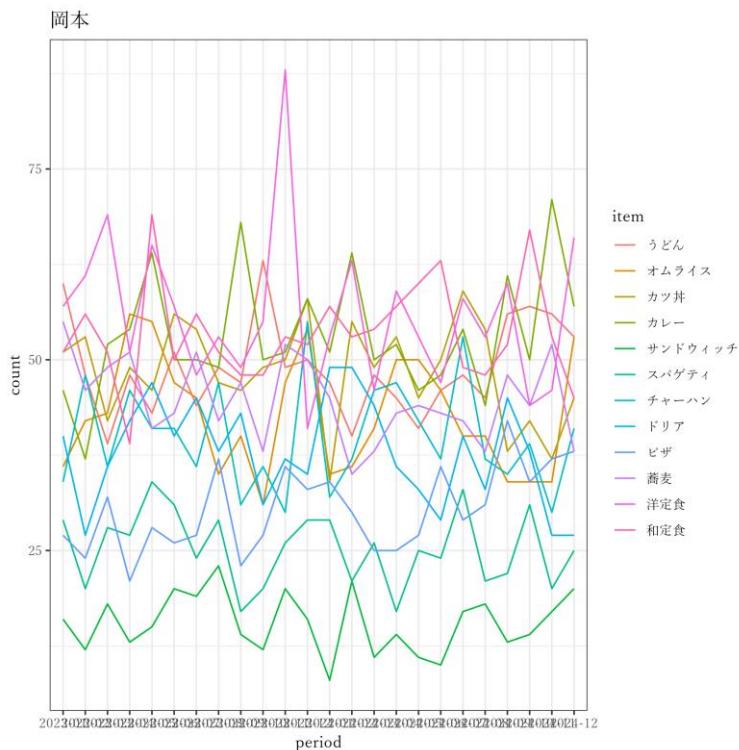


図 2.14: 2 番目のグラフ

作図したオブジェクトを一気に保存することも可能です。`ggsave()` はパスと作図オブヘクトの 2 つの引数を指定するため、`map()` の派生形である `map2()` を使います。`map2()` の第 1 引数と第 2 引数には、繰り返す関数の第 1 引数と第 2 引数、つまりここではパスと作図オブジェクトを指定します。`ggsave()` の第 3 引数以降は、関数の後ろに書きます。なお、ファイルのパスは `path_temp()` でファイル名を生成しています。

コード 2.60 (analysis-purrr-split-map2.R) : 複数の図の保存

```
pdfs <-
  paste0(names(gg_sales_split), ".pdf") |>
  fs::path_temp()
purrr::map2(pdfs, gg_sales_split, ggsave,
            device = cairo_pdf, width = 7, height = 7)
## [[1]]
## C:/Users/USERNAME/AppData/Local/Temp/RtmpSmq99m/芦屋.pdf
## [[2]]
## C:/Users/USERNAME/AppData/Local/Temp/RtmpSmq99m/岡本.pdf
## (以下省略)
# shell.exec(pdfs[1])
```

大量の繰り返し処理をするときには、途中でエラーが発生することがあります。また、通信状況などによるエラーも考えられます。エラーの発生が予想されるときには、`safely()` あるいは `possibly()` で関数をエラー対応のものに変換しておくと良いでしょう。`safely()` はエラー時も止

まらずに実行して、結果とエラーを返します。`possibly()`はエラー時も止まらずに実行して、エラー時には設定した値を返します。

準備として、2のときにエラーになる関数を定義します。

コード 2.61 (analysis-purrr-safely-prep.R) : 2のときにエラーになる関数

```
error_if_two <- function(x){  
  if(x == 2){  
    stop("エラーです")  
  }else{  
    return(x)  
  }  
}
```

`map()`で `error_if_two()` を実行するときと、`possibly()` でエラー対応の関数にしたときを比較します。

コード 2.62 (analysis-purrr-safely.R) : 繰り返し処理のエラー対応

```
purrr::map(1:3, error_if_two) # そのままのとき  
## Error in `purrr::map()`:  
## i In index: 2.  
## Caused by error in `f()`:  
## ! エラーです  
## Run `rlang::last_trace()` to see where the error occurred.  
error_if_two Possibly <- possibly(error_if_two, otherwise = 0) # エラー時は0  
purrr::map_dbl(1:3, error_if_two_Possibly)  
## [1] 1 0 3
```

エラーのときには、0を返してくれるとともに、実行が継続されます。

2.9 順次処理

各要素の内容を順次処理するときには `purrr` パッケージの `reduce()` や `accumulate()` を使います。これは階乗のようなもので、1:4 のベクトルがあったときに `1*2` をして、その結果の2に対して `*3` を、次に `*4` をして最終的に 24を得ます。途中の結果は不要で、最終的な結果だけを得るのが `reduce()` です。途中の結果も得るのが `accumulate()` です。

コード 2.63 (analysis-purrr-reduce-add.R) : 順次処理の関数の基本動作

```
accumulate(1:4, `*`)  
## [1] 1 2 6 24
```

```
reduce(1:4, `*`)  
## [1] 24
```

本来は、`distinct()`で処理すればよいのでしょうが、ここでは説明用に`reduce()`を使います。`paste_if_new()`は、`x`と`y`を比較して、`x`に`y`が含まれなければ、つまり新規であれば追加します。既存の場合は、そのまま`x`を返します。

コード 2.64 (`analysis-purrr-paste-if-new-fun.R`) : 新しいものを追加する関数

```
paste_if_new <- function(x, y){  
  pattern <- paste0("(^|;)+", y, "(;|$)+")  
  if(stringr::str_detect(x, pattern)){  
    x  
  }else{  
    paste0(x, ";", y)  
  }  
}
```

次のコードを実行すると、`answer`について`area`と`satisfy`のグループ別で`apps`の文字列を重複なく結合できます。

コード 2.65 (`analysis-purrr-reduce.R`) : 順次処理の関数

```
answer |>  
dplyr::summarise(apps = reduce(apps, paste_if_new),  
  .by = c(area, satisfy))  
## # A tibble: 28 × 3  
##   area      satisfy apps  
##   <chr>     <dbl> <chr>  
## 1 関東          1 -;HAD;JASP;S-PLUS;SPSS;others;SAS;STATISTICA;ex...  
## 2 中部          5 -;HAD  
## 3 九州・沖縄      2 excel;JMP;S-PLUS;HAD;SAS;STATISTICA;others;SPSS  
## 4 中国・四国      4 -;excel;JASP;SAS;STATISTICA  
## 5 中国・四国      1 excel;others;HAD;SAS;SPSS;JMP;S-PLUS;STATISTICA...  
## 6 近畿          4 excel;JASP;S-PLUS;SPSS;-;others;STATISTICA;SAS;...  
## 7 関東          4 SAS;S-PLUS;SPSS;excel  
## 8 中部          3 HAD;JASP;JMP;S-PLUS;SPSS;-  
## 9 関東          5 excel;HAD;JASP;S-PLUS;STATISTICA;JMP;SAS;SPSS  
## 10 中国・四国     2 excel;HAD;JMP;SAS;SPSS;S-PLUS;STATISTICA;others  
## # 18 more rows
```

3 文字列の操作の自動化

文書を作成したあとに、ことばを言い換えたいことはよくあります。ワードでは検索や置換の機能を使いますが、複数のファイルが対象のときには「ファイルを開く・検索や置換する・ファイルを書き込む・ファイルを閉じる」という一連の作業を繰り返さなければなりません。作業中に「このファイルは作業済みだったっけ」ということもしばしばあります。

検索の対象文字列が決まっているれば作業は簡単です。しかし、日付のように形式は決まっていても具体的な表現が多様なことがあります。目視での確認では漏れが出てくることがあるでしょう。そんなときには、一定の形式にマッチする正規表現を用いると効率的な検索と置換ができる、自動化作業で役立ちます。

さらに、多数のファイルを生成するとき、ファイル名を 001_xxxx.pdf のように連番を付けたいこともあります。このようなファイル名の自動的な生成は、プログラムが得意とするところです。もちろん、既存のファイル名の文字列の置換や削除、ディレクトリやファイルの名前の検索や変更も自動化できます。

本章では、`stringr` パッケージを使った文字列の結合・置換・削除・検索・抽出など基本的な文字列の操作方法と正規表現について、例を示しながら説明します。また、自動化で活用できる `stringr` のその他の関数の使い方を簡単に紹介します。

3.1 文字列を操作するパッケージ

`stringr` は `tidyverse` に含まれています。そのため、`tidyverse` をインストール済みなら、インストールは不要です。`tidyverse` が未インストールの場合は、インストールをお勧めします。

なお、`stringr` は文字列を操作する `stringi` パッケージを使いやくしたものです。`stringi` だけの関数がありますので、より高度な文字列の操作を自動化するには、`stringi` の使用を検討してください。

コード 3.1 (string-install.R) : `tidyverse` のインストール

```
install.packages("tidyverse") # stringr を含むパッケージ群をインストール  
# install.packages("stringr") # 単独でのインストール
```

コード 3.2 (string-library.R) : `tidyverse` の呼び出し

```
library(tidyverse) # tidyverse を呼び出し(tidyverse は stringr を含む)  
# library(stringr) # 単独で呼び出すとき
```

3.2 結合

複数の文字列を結合するときには、`str_c()`を使います。引数の指定の仕方によって挙動が異なります。Rに組み込まれている`paste0()`もほぼ同じ機能ですが、動作が異なることがあります。

コード 3.3 (string-str-c.R) : 文字列の結合

```
str_neko <-  
  c("吾輩は猫である。", "名前はまだない。",  
    "I am a cat.", "I don't have any name yet.")  
str_c(str_neko, "◆") # 各文字列に"◆"を追加  
## [1] "吾輩は猫である。◆"           "名前はまだない。◆"  
## [3] "I am a cat.◆"                 "I don't have any name yet.◆"  
# paste0(str_neko, "◆")でも同じ  
  
# collapse で 1 つの文字列に結合する  
str_c(str_neko[1:3], collapse = "◆") # [1:3] : 出力を短くするため  
## [1] "吾輩は猫である。◆名前はまだない。◆I am a cat."  
# paste0(str_neko[1:3], collapse = "◆")も同じ  
  
# 複数の文字列を引数にしたとき  
str_c("吾輩は", "猫である。")  
## [1] "吾輩は猫である."  
# paste0("吾輩は", "猫である。")も同じ  
  
# 注意：動作が異なる  
str_c("吾輩は", "猫である。", sep = "◆")  
## [1] "吾輩は◆猫である."  
paste0("吾輩は", "猫である。", sep = "◆")  
## [1] "吾輩は猫である。◆"
```

引数に文字列のベクトルを指定したときには特に注意してください。エラーや想定と異なる結果になることがあります。

コード 3.4 (string-str-c-caution.R) : 文字列の結合での注意点

```
str_c(1:5, str_neko, "◆") # エラー：文字列のベクトルの要素数が不一致  
## Error in `str_c()`:  
## ! Can't recycle `..1` (size 5) to match `..2` (size 4).  
str_c(1:4, str_neko, "◆") # ベクトルで指定するときの出力に注意  
## [1] "1 吾輩は猫である。◆"           "2 名前はまだない。◆"  
## [3] "3I am a cat.◆"                 "4I don't have any name yet.◆"
```

《Tips》 プログラミングで一番困るのは、希望する動作と違う動作で作業が完結してしまうことです。エラーが出ればうまく行かなかったことがわかります。一方で作業が完結すると、思い込

みどおりに作業できているものとみなして、「成功した」と勘違いしてしまいます。これがプログラミングで一番怖いことです。動作確認は必ずしましょう。

3.3 マッチ箇所の明示

【変更点：独立化】

`stringr` の関数で文字列を検索したときに、検索文字列とのマッチした箇所を明示するには、`str_view()`を使います。マッチした箇所は<マッチした箇所>のように<>で囲って表示されます。

コード 3.5 (str-view.R) : マッチ箇所の明示

```
str <- c("今日はいい天気です。")
pattern <- "いい天気"
str_view(str, pattern)
## [1] | 今日は<いい天気>です。
c(str, "明日もいい天気かな。明後日はいい天気でしょう。") |>
  str_view(pattern)
## [1] | 今日は<いい天気>です。
## [2] | 明日も<いい天気>かな。明後日は<いい天気>でしょう。
```

3.4 正規表現

正規表現を使うと、文字列を操作するときの自動化の幅が広がります。プログラミングだけでなく、テキストエディタでも活躍します。ただし、正規表現はプログラミング言語やエディタによって仕様が若干異なります。`vignette("regular-expressions")`とすると解説を見ることができます。正規表現をさらに詳しく知りたい人は Wickham, Çetinkaya-Rundel, and Grolemund (2024)などを参考にしてください。

正規表現という言葉を初めて見聞きしたときは、意味が分からぬと思います。言葉だけでは意味を想像できないからです。正規表現は `regular expression` という英語の訳語です。`regular` を「正規」と訳しているのですが、これだけでは分かりにくい気がします。`regular` には「手続きに従った」との意味があり、`regular expression` は「ある手続きに従った表現」と理解したほうが分かりやすいと思います。つまり、文字列を具体的な表現ではなく、ある手続きに従った表現として抽象化したものです(表 3.1)。実例で考えたほうが分かりやすいでしょう。

表 3.1: 主な正規表現と意味(以下の m や n は数字)

正規表現	意味
[]	範囲内の文字
[^]	範囲外の文字

正規表現 意味	
^	行頭
\$	行末
{n}	直前パターンが n 回
{m,n}	直前パターンが m-n 回
*	直前パターンが 0 回以上
+	直前パターンが 1 回以上
?	直前パターンが 0-1 回
.(ドット)	任意の 1 文字
	または
()	グループ
\n	参照タグ : n 番目のマッチ

たとえば、文書の中から "2007_05_26" や "2010-07-23" のような年月日の表現の全てを抽出したいとします。すべての年月日の組み合わせを書き出すことは無意味です。そこで正規表現の出番です。2000 年から 2020 年代まで区切りが "-" (ハイフン) か "_" (アンダーバー) であれば、次のように表現できます。

```
"20[0-2][0-9][_-][01][0-9][_-][0-3][0-9]"
```

ここでは [] で囲んだ文字でその範囲内の文字を表現しています。[0-2] は「0 から 2 まで」、[01] は「0 か 1」です。年月日は 0 から 9 までの数字が入るはずです。すべて [0-9] としても構いませんが、この場合は実在しない "2010-30-59" もマッチします。

そのような文字列もマッチさせるのであれば、[0-9]{4}[_-][0-9]{2}[_-][0-9]{2} とするほうが良いでしょう。ここでは、{n} は「直前パターンが n 回」という意味です。つまり 4 つ、2 つ、2 つの数字が「_か-のどちらか」で繋がっている文字列がマッチします。

【追加】 大文字・小文字の区別、`regex()`, `fixed()` の部分を追加しました。

`stringr` の関数での正規表現では、特に指定しないとアルファベットの大文字と小文字を区別して扱います。ただし、大文字と小文字の違いを無視して、"A" にも "a" にもマッチさせたいことがあります。その時には、`regex(pattern, ignore_case = TRUE)` とすると "A" と "a" のどちらにもマッチします。

コード 3.6 (string-regex.R) : 大文字・小文字の区別の有無

```
str <- c("A B C a b c")
pattern <- "A|B"
pattern_ic <- regex(pattern, ignore_case = TRUE)
```

```

str_view(str, pattern)    # 大・小の区別あり
## [1] | <A> <B> C a b c
str_view(str, pattern_ic) # 大・小の区別なし
## [1] | <A> <B> C <a> <b> c

```

また、`stringr` の関数の引数 `pattern` には正規表現を使いますが、文字そのものをマッチさせたいときは `fixed()` を使います。`regex()` と同様に引数で `ignore_case = TRUE` とすれば、大文字と小文字を区別しません。

コード 3.7 (string-fixed.R) : 正規表現ではなく文字そのものでマッチ

```

str <- c("abc a.c ABC A.C")
pattern <- fixed("a.c")
pattern_ic <- fixed("a.c", ignore_case = TRUE)
str_view(str, pattern)    # 文字列そのもの、大・小の区別あり
## [1] | abc <a.c> ABC A.C
str_view(str, pattern_ic) # 文字列そのもの、大・小の区別なし
## [1] | abc <a.c> ABC <A.C>

```

正規表現では()でグループ化して結合する優先順位を指定できます。数式で足し算よりも掛け算が優先されるように、正規表現でも結合する優先順位があります。たとえば、`abc+`は「ab の次にcが1回以上の繰り返し」とマッチします。つまり、`abcccc` の文字列全体に、`abcabcabc` では最初の `abc` とマッチしますが、`abcabcabc` の文字列全体とはマッチしません。`abcabcabc` の全体、つまり「`abc` の1回以上の繰り返し」とマッチさせるには`(abc)+`とします。

グループ化は置換時にマッチした文字列を参照するときにも活用します(表 3.1)。

`str_replace()` による置換(3.5 節を参照)では、「AをBに置換」のように単純に置き換えたいときばかりではありません。マッチした部分をそのまま残したいこともあるでしょう。たとえば年月日で「2023-12-13」を「2023_12_13」に置換するときに文章全体の「-」を「_」に置換すると、年月日とは関係のない「-」も「_」に置換してしまいます。そのときには、検索文字列に`([0-9]{4})[-]([0-9]{2})[-]([0-9]{2})`を、置換文字列に`\1_\2_\3`を指定します。()でグループ化した部分についてマッチした順に`\1`, `\2`, `\3`として参照できます。つまり、`2023`が`\1`, `12`が`\2`, `13`が`\3`になり、置換後の文字列が「2023_12_13」になります。このように、マッチした部分を参照して置換します。

.や+のように正規表現で特殊な意味を持つ文字そのものをマッチさせたいこともあります。その場合は、エスケープ文字といって\とその文字そのものを組み合わせて使います(表 3.2)。たとえば、+そのものとマッチさせたければ\+とします。

また、エスケープ文字と似たような表現としてメタ文字があります。メタ文字を使うことで、改行やタブなどの特殊な文字や数字などを指定できます。

表 3.2: エスケープ文字とメタ文字

正規表現	意味
\	\

正規表現	意味
\	\

正規表現 意味

\	
\.	.
\^	^
\\$	\$
*	*
\+	+
\?	?
\[]	[]
\{\}	{}
\()	()
\n	改行
\t	タブ
\s	空白文字(\t,\n など)
\d	数字([0-9])
\D	\d 以外の文字
\w	_を含む英数文字([a-zA-Z0-9_])
\W	\w 以外の文字

【追加】 正規表現の注意を追加しました。

正規表現のエスケープ文字とメタ文字を扱うときには、ややこしい問題があります。普通の文字列として"."(ドット)があるときに、正規表現でマッチさせるには\\.(のように\を2つ使わなければなりません。正規表現のときには、\\が\の意味になるからです。

コード 3.8 (string-regexp-meta.R) : エスケープ文字とメタ文字の例

```
str <- "Hello. "
str_view(str, ".") # 全てにマッチ
## [1] | <H><e><l><l><o><.>>< >
str_view(str, "\.") # エラー
## Error: '\.' is an unrecognized escape in character string (<input>:1:17)
str_view(str, "\\.") # .(ドット)にマッチ
## [1] | Hello<.>
```

3.5 置換・削除

文字列の置換には `str_replace()` や `str_replace_all()` を使います。 `str_replace()` は検索したものにマッチした最初の文字列だけを、 `str_replace_all()` はマッチしたすべての文字列を置換します。

コード 3.9 (string-str-replace.R) : 文字列の置換

```
str_replace(str_neko, " ", "◆")      # 最初の半角スペースを◆に置換
## [1] "吾輩は猫である。"           "名前はまだない。"
## [3] "I◆am a cat."            "I◆don't have any name yet."
str_replace_all(str_neko, " ", "◆") # 全部を置換
## [1] "吾輩は猫である。"
## [2] "名前はまだない。"
## [3] "I◆am◆a◆cat."
## [4] "I◆don't◆have◆any◆name◆yet."
```

`str_replace()` は引数として対象文字列の `string`、置換前の文字列の `pattern`、置換後の文字列の `replacement` をとります。 このうち、`pattern` には正規表現(3.4 節を参照)を用いることができます。 たとえば、"は"・"猫"・"。" "・" a のどれかに当てはまれば、"◆"に置換するとします。 そのときは、[]で囲み"[は猫。 a]"とします。

コード 3.10 (string-str-replace-regrep.R) : 正規表現を使った文字列の置換

```
pattern <- "[は猫。 a]"
str_replace(str_neko, pattern, "◆")
## [1] "吾輩◆猫である。"           "名前◆まだない。"
## [3] "I ◆m a cat."            "I don't h◆ve any name yet."
str_replace_all(str_neko, pattern, "◆")
## [1] "吾輩◆◆である◆"          "名前◆まだない◆"
## [3] "I ◆m ◆ c◆t."            "I don't h◆ve ◆ny n◆me yet."
```

文字列の削除には `str_remove()` や `str_remove_all()` を使います。 違いは `str_replace()` や `str_replace_all()` と同様です。 次のコードは、"[a-z]"で小文字のアルファベット全部、"[あ-ん]"でひらがな全部を対象に除去します。 ただし、`str_remove()` では最初の 1 つだけを除去し、`str_remove_all()` ではマッチした全部を除去します。

コード 3.11 (string-str-remove.R) : 正規表現を使った文字列の削除

```
str_remove(str_neko, "[a-z]")      # [a-z]：小文字のアルファベット全部
## [1] "吾輩は猫である。"           "名前はまだない。"
## [3] "I m a cat."              "I on't have any name yet."
str_remove_all(str_neko, "[a-z]")
## [1] "吾輩は猫である。" "名前はまだない。" "I ."
## [4] "I ' ."
```

```

str_remove(str_neko, "[あ-ん]")    # [あ-ん]：ひらがな全部
## [1] "吾輩猫である。"           "名前まだない。"
## [3] "I am a cat."            "I don't have any name yet."
str_remove_all(str_neko, "[あ-ん]")
## [1] "吾輩猫。"                 "名前。"
## [3] "I am a cat."            "I don't have any name yet."

```

ところで、`str_remove()`や`str_remove_all()`は`str_replace()`や`str_replace_all()`と似ていることに気づくかもしれません。これらの違いは引数に`replacement`がないぐらいです。関数の中身を見てみると、`replacement`を`""`として`str_replace()`を呼び出しているだけです。

コード 3.12 (string-str-remove-fun.R) : `str_remove()`の中身

```

str_remove # 関数の中身
## function (string, pattern)
## {
##   str_replace(string, pattern, "")
## }
## <bytecode: 0x000001615c10cf50>
## <environment: namespace:stringr>

```

3.6 検索・抽出

文字列に特定の文字列が含まれるかどうかなどを検索するには、`str_detect()`をはじめとする三姉妹を使います。`str_detect()`は文字列の有無で、`str_starts()`は最初の文字列で、`str_ends()`は最後の文字列でマッチするかどうかを論理値(TRUE / FALSE)で返します。最初や最後の文字列ということは、正規表現で`^`や`$`を使うのと同じです。実際に`str_starts()`と`str_ends()`は、内部で`^`や`$`を追加しています。ただし、単に`^`や`$`を追加するだけではないので、詳細は関数の中身を見てください。

コード 3.13 (string-str-brothers.R) : 文字列の検索

```

str_detect(str_neko, "猫")
## [1] TRUE FALSE FALSE FALSE
str_starts(str_neko, "名前")
## [1] FALSE TRUE FALSE FALSE
str_ends(str_neko, "t.")
## [1] FALSE FALSE TRUE TRUE

```

`str_detect()`三姉妹は、データフレームにおける要素の抽出で活躍します。論理値でマッチした結果を返すので、`dplyr::filter()`と組み合わせるとマッチするものだけを抽出できます。また、`str_detect(str, pattern, negate = TRUE)`でマッチしないものを抽出できます。

コード 3.14 (string-filter-detect.R) : 文字列の検索

```

# library(tidyverse) # tidyverse を呼び出していないとき
(mpg <- mpg[,1:5]) # 自動車の燃費データ(dplyrに含まれる)のうち5列だけ
## # A tibble: 234 × 5
##   manufacturer model      displ  year   cyl
##   <chr>        <chr>     <dbl> <int> <int>
## 1 audi         a4          1.8  1999    4
## 2 audi         a4          1.8  1999    4
## 3 audi         a4          2    2008    4
## 4 audi         a4          2    2008    4
## 5 audi         a4          2.8  1999    6
## 6 audi         a4          2.8  1999    6
## 7 audi         a4          3.1  2008    6
## 8 audi         a4 quattro  1.8  1999    4
## 9 audi         a4 quattro  1.8  1999    4
## 10 audi        a4 quattro  2    2008    4
## # i 224 more rows
dplyr::filter(mpg, # pickupとマッチするもの
  str_detect(model, "pickup")) |> print(n = 5)
## # A tibble: 26 × 5
##   manufacturer model      displ  year   cyl
##   <chr>        <chr>     <dbl> <int> <int>
## 1 dodge        dakota pickup 4wd   3.7  2008    6
## 2 dodge        dakota pickup 4wd   3.7  2008    6
## 3 dodge        dakota pickup 4wd   3.9  1999    6
## 4 dodge        dakota pickup 4wd   3.9  1999    6
## 5 dodge        dakota pickup 4wd   4.7  2008    8
## # i 21 more rows
dplyr::filter(mpg, # pickupとマッチしないもの
  str_detect(model, "pickup", negate = TRUE)) |> print(n = 5)
## # A tibble: 208 × 5
##   manufacturer model displ  year   cyl
##   <chr>        <chr> <dbl> <int> <int>
## 1 audi         a4       1.8  1999    4
## 2 audi         a4       1.8  1999    4
## 3 audi         a4       2    2008    4
## 4 audi         a4       2    2008    4
## 5 audi         a4       2.8  1999    6
## # i 203 more rows

```

`str_starts()`と`str_ends()`はデータフレーム内の文字列のうち、最初と最後のものにマッチするものを抽出するときに使います。

コード3.15 (string-filter-starts.R) : 最初と最後にマッチする文字列の検索

```

dplyr::filter(mpg, str_starts(model, "m")) |> print(n = 5) # mで始まる
## # A tibble: 21 × 5
##   manufacturer model displ  year   cyl
##   <chr>        <chr> <dbl> <int> <int>
## 1 chevrolet    malibu  2.4  1999    4
## 2 chevrolet    malibu  2.4  2008    4
## 3 chevrolet    malibu  3.1  1999    6
## 4 chevrolet    malibu  3.5  2008    6
## 5 chevrolet    malibu  3.6  2008    6
## # i 16 more rows
dplyr::filter(mpg, str_ends(model, "4wd")) |> print(n = 5) # 4wdで終わる

```

```

## # A tibble: 74 × 5
##   manufacturer model      displ  year    cyl
##   <chr>        <chr>     <dbl> <int> <int>
## 1 chevrolet    k1500 tahoe 4wd     5.3  2008     8
## 2 chevrolet    k1500 tahoe 4wd     5.3  2008     8
## 3 chevrolet    k1500 tahoe 4wd     5.7  1999     8
## 4 chevrolet    k1500 tahoe 4wd     6.5  1999     8
## 5 dodge        dakota pickup 4wd    3.7  2008     6
## # i 69 more rows

```

抽出したい要素の文字列が複数あるときは、 "subaru|toyota" というように、 |を使った正規表現で指定できます。

(#thm:string-filter-code) (string-filter-.R) : 正規表現を使った複数文字列の検索

```

str <- "subaru|toyota" # subaru か toyota
dplyr::filter(mpg, str_detect(manufacturer, str)) |> print(n = 5)
## # A tibble: 48 × 5
##   manufacturer model      displ  year    cyl
##   <chr>        <chr>     <dbl> <int> <int>
## 1 subaru       forester awd    2.5  1999     4
## 2 subaru       forester awd    2.5  1999     4
## 3 subaru       forester awd    2.5  2008     4
## 4 subaru       forester awd    2.5  2008     4
## 5 subaru       forester awd    2.5  2008     4
## # i 43 more rows
# dplyr::filter(mpg, manufacturer %in% c("subaru", "toyota")) # 上と同じ

```

多くの文字列を含むベクトルから特定の文字列を含む要素を抽出するとします。 そのときはマッチする文字列を抽出する `str_subset()`を使います。

コード 3.16 (string-str-subset.R) : 文字列の抽出

```

str_stringr <- ls("package:stringr") # パッケージのオブジェクト一覧
length(str_stringr) # 要素数
## [1] 62
head(str_stringr, 15) # 最初の 15 個
## [1] "%>%"          "boundary"       "coll"          "fixed"
## [5] "fruit"          "invert_match"   "regex"         "sentences"
## [9] "str_c"           "str_conv"       "str_count"     "str_detect"
## [13] "str_dup"         "str_ends"       "str_equal"
str_subset(str_stringr, "str_s") # 最初が str_s
## [1] "str_sort"        "str_split"      "str_split_1"
## [4] "str_split_fixed" "str_split_i"    "str_squish"
## [7] "str_starts"      "str_sub"        "str_sub_all"
## [10] "str_sub<-"       "str_subset"
str_subset(str_stringr, "t$") # 末尾が t
## [1] "fruit"          "str_count"     "str_detect"   "str_extract"
## [5] "str_sort"        "str_split"     "str_subset"

```

なお、 `str_subset()`に似た関数に `str_sub()`があります。 名前が似ているので混同しやすいですが、 動作は全く違います。 `str_sub()`は文字列の最初と最後の位置を指定して抽出します。 複数

の文字列に対して個別に位置を指定するには、`start` と `end` の一方か両方に文字列と同じ長さのベクトルを指定します。 ただ、そのような使い方は少ないでしょう。

コード 3.17 (`string-str-sub.R`) : 文字列の抽出

```
str_123 <-  
  c(paste0(1:9, collapse = ""),  
    "abcdefg",  
    "あいうえおかきくけこ") |>  
  print()  
## [1] "123456789"           "abcdefg"  
## [3] "あいうえおかきくけこ"  
str_sub(str_123, start = 2, end = 6) # 全て 2-6 を抽出  
## [1] "23456"      "bcdef"      "いうえおか"  
str_sub(str_123, 1:3, 3:5)          # 前から順に 1-3, 2-4, 3-5 を抽出  
## [1] "123"        "bcd"        "うえお"
```

3.7 その他の関数

`stringr` には有用な関数が多くあります(表 3.3)。 簡単に使用法を紹介します。 詳しい使い方は、関数のヘルプを参照してください。

表 3.3: `stringr` のその他の関数

関数	内容
<code>str_count()</code>	マッチ箇所の数
<code>str_locate()</code> , <code>str_locate_all()</code>	マッチ箇所の位置(<code>start</code> , <code>end</code>)
<code>str_pad()</code>	字数合わせ
<code>str_trunc()</code>	切り捨てて省略
<code>str_flatten()</code>	文字列ベクトルの結合
<code>str_split()</code>	分割
<code>str_split_1()</code>	分割(長さが 1 の文字列)
<code>str_which()</code>	マッチ要素のインデックス
<code>str_length()</code>	文字列の長さ(個数)
<code>str_width()</code>	文字列の表示幅
<code>str_unique()</code>	重複除去
<code>str_trim()</code>	端の空白文字の除去
<code>str_squish()</code>	端と重複の空白文字の除去

コード 3.18 (string-others.R) : stringr のその他の関数

```
# 文字列
str_123
## [1] "123456789"           "abcdefg"
## [3] "あいうえおかきくけこ"

str_neko
## [1] "吾輩は猫である。"          "名前はまだない。"
## [3] "I am a cat."            "I don't have any name yet."
# アルファベットの小文字かひらがなが1つ以上
pattern <- "[a-z]+|[あ-ん]+"

# マッチ箇所の確認
str_view(str_neko, pattern)
## [1] | 吾輩<は>猫<である>。
## [2] | 名前<はまだない>。
## [3] | I <am> <a> <cat>.
## [4] | I <don>'<t> <have> <any> <name> <yet>.

# マッチ箇所の数
str_count(str_neko, pattern)
## [1] 2 1 3 6

# 1つ目のマッチ箇所の位置(start, end)
str_locate(str_neko, pattern)
##      start end
## [1,]     3   3
## [2,]     3   7
## [3,]     3   4
## [4,]     3   5

# 全てのマッチ箇所の位置(start, end)
str_locate_all(str_neko, pattern)
## [[1]]
##      start end
## [1,]     3   3
## [2,]     5   7
##
## [[2]]
##      start end
## [1,]     3   7
##
## [[3]]
##      start end
## [1,]     3   4
## [2,]     6   6
## [3,]     8  10
##
## [[4]]
##      start end
## [1,]     3   5
## [2,]     7   7
## [3,]     9  12
```

```

## [4,]    14  16
## [5,]    18  21
## [6,]    23  25

# 字数合わせ
str_pad(1:10, width = 2, side = "left", pad = "0")
## [1] "01" "02" "03" "04" "05" "06" "07" "08" "09" "10"

# 切り捨てて省略
str_trunc(str_neko, 7, "right") # 右を切り捨て
## [1] "吾輩は猫..." "名前はま..." "I am..."     "I do..."
str_trunc(str_neko, 7, "center") # 中央を切り捨て
## [1] "吾輩...る。" "名前...い。" "I ...t."     "I ...t."
str_trunc(str_neko, 7, "left")  # 左を切り捨て
## [1] "...である。" "...だない。" "...cat."     "...yet."

# 分割
(splitted <- str_split(str_neko, "は| "))      # 「は」か半角スペース
## [[1]]
## [1] "吾輩"       "猫である。"
##
## [[2]]
## [1] "名前"       "まだない。"
##
## [[3]]
## [1] "I"          "am"        "a"         "cat."
##
## [[4]]
## [1] "I"          "don't"    "have"    "any"      "name"    "yet."
str_flatten(str_neko) |> str_split_1("." |\\.) # 「。」か「.」
## [1] "吾輩は猫である"           "名前はまだない"
## [3] "I am a cat"              "I don't have any name yet"
## [5] ""

# 文字列ベクトルの結合
str_flatten(splitted[[1]], collapse = "◆")
## [1] "吾輩◆猫である。"

# マッチ要素のインデックス
str Which(str_neko, "猫|cat")
## [1] 1 3

# 文字列の長さ(個数)
str_length(str_123)
## [1] 9 7 10

# 文字列の表示幅(半角は 1, 全角は 2)
str_width(str_123)
## [1] 9 7 20

```

```

# 重複除去
(str_number <- letters[c(1:5, 3:7)])
## [1] "a" "b" "c" "d" "e" "c" "d" "e" "f" "g"
str_unique(str_number)
## [1] "a" "b" "c" "d" "e" "f" "g"

# 空白文字の除去
str_trim(" a b c ") # 端のみ除去
## [1] "a b c"
str_squish(" a b c ") # 重複も除去
## [1] "a b c"

```

3.8 文章の比較

【追加】 文章の比較の節を追加しました。

文章の変更部分がわからなくなることがあります。 そのようなときは、 `diffr` パッケージが便利です。

`diffr` を CRAN からインストールして呼び出します。

コード 3.19 (`string-diffr-install.R`) : `diffr` のインストールと呼び出し

```

install.packages("diffr")
library(diffr)

```

`diffr()` で文字単位の比較が可能です。 比較するときは、 あらかじめファイルに保存する必要があります。

コード 3.20 (`string-compare-diffr.R`) : 文章の比較

```

f1 <- fs::file_temp()
f2 <- fs::file_temp()
writeLines("日本語での比較実験\n今日は晴れです。 \n同じ文章", con = f1)
writeLines("英語での比較の実験\n今日は天気です。 \n同じ文章", con = f2)
diffr::diffr(f1, f2, before = fs::path_file(f1), after = fs::path_file(f1))

```

文字単位で比較した結果が、 HTML の左右対照形式で表示されます(図 3.1)。 変更のある行はやや薄い桃色(緑色)で表示され、 変更文字はやや濃い色で表示されます。

file13c425e3163	file13c425e3163
1 日本語での比較実験	英語での比較の実験
2 今日は晴れです。	今日は天気です。
3 同じ文章	同じ文章

図 3.1: `diffr` で文字列を比較した結果

変更文字が分かりにくいときは、以下の path の codediff.css を修正すると色の設定を変更できます。

コード 3.21 (word-css-path.R) : 比較結果の HTML の設定ファイル

```
path <- fs::path(fs::path_package("diffR"),
                 "htmlwidgets/lib/codediff/codediff.css")
shell.exec(path)

# codediff.css の 72 行目以降
# 変更行(左)
.before.replace {
  background-color: #fee;
}

# 変更行(右)
.after.replace {
  background-color: #efe;
}

# 変更文字(左)
.before .char-replace, .before .char-delete {
  background-color: #fcc;
}

# 変更文字(右)
.after .char-replace, .after .char-insert {
  background-color: #cfc;
}
```

CSS(Cascading Style Sheet)は、HTML の表示方法を指定するものです。 色は#ffffff(白)のように#と 6 衡の 16 進数で指定します。 6 衡のうち 2 衡ずつが赤、緑、青に割り当てられており、各色の数字が 00 や ff のようにすべて同じときは、それぞれの 2 衡の部分を 1 衡に短縮できます。つまり、#0088ff と#08f は同じ意味になります。また、特定の色は white や red のように文字列でも指定可能です。文字列での指定可能な色は、silver, gray, white, maroon, red, purple, fuchsia, green, lime, olive, yellow, navy, blue, teal, aqua などです。

以下のように#fcc と#cfc の両方を#fcc(黄)に変更するとともに、font-size: 150%; としてフォントサイズを 1.5 倍に設定すると、図 3.2 の表示になります。

```
.before .char-replace, .before .char-delete {
  background-color: yellow;
  font-size: 150%;
}

.after .char-replace, .after .char-insert {
  background-color: yellow;
  font-size: 150%;
}
```

	file13c451f355cd	file13c451f355cd
1	日本語での比較実験	英語での比較の実験
2	今日は晴れです。	今日は天気です。
3	同じ文章	同じ文章

図 3.2: `diff` で文字列を比較した結果(色とフォントサイズを変更)

【追加・修正点】 tidyverse にあった lubridate を独立させました。ワードなどで使うことを考えて、日付っぽい文字列を取り出して、置換する作業の例を入れました。曜日の間違いのチェックもできるようにしました。最後の方の曜日の修正・1年後の日付などのところは、実際の文書から取り出して保存するところまでの全体的なコードは、ワードの章で書いています。

4 日付データの操作

年月日などの日付データは、様々な文書に出てきます。文書を再利用のときに日付データを正しく修正すれば問題ありません。しかし、定期的に実施するイベントの年月日だけ修正して曜日が修正されていないときは、混乱のもとです。

年月日や曜日の修正を自動化すれば、定期的な更新が楽になります。また、文書の中で日付と曜日があわない箇所の一覧があれば、確認の手間を大幅に減らせます。

日本語の文書では、西暦と和暦が出てくるのも頭を悩ませます。西暦と和暦の変換は難しい作業ではありませんが、西暦と昭和・平成・令和を全て正しく変換するには確認が必要です。これでも自動化できます。

本章では日付データを操作するパッケージの基本手な使い方を紹介します。また、自動化例として年月日と曜日との矛盾箇所の抽出、曜日固定での日付の変更、西暦と和暦との相互変換や併記の方法をご紹介します。

4.1 日付データを扱うパッケージ

年月日や曜日を簡単に扱うには、lubridate パッケージを利用します。lubridate は tidyverse に含まれているため、tidyverse が既に入っている場合は、単独でのインストールは不要です。なお、lubridate は日付だけでなく、時刻データも扱えます。

また、和暦を使用するために zipangu パッケージと stringi パッケージを利用します。zipangu は日付だけでなく、日本で扱う文字などのデータ操作にも対応しています。stringi は、tidyverse をインストールすれば stringr と一緒にインストールされます。

さらに、日付の確認用にカレンダーを表示するパッケージの calendR をインストールします。

コード 4.1 (date-install.R) : zipangu と calendR のインストール

```
# install.packages("lubridate") # 個別にインストールするとき
# install.packages("stringi") # 個別にインストールするとき
install.packages("zipangu")
install.packages("calendR")
```

コード 4.2 (datte-library.R) : tidyverse, zipangu, calendR の呼び出し

```

library(tidyverse)
# library(lubridate) # 個別に呼び出すとき
# library(stringi) # 個別に呼び出すとき
library(zipangu)
library(calendR)

```

4.2 日付クラスへの変換

日本語の表記でよく出てくる年・月・日の日付表記の文字列は、関数 `ymd()` で日付クラスに変換できます。`ymd()` は、日付っぽい文字列を日付クラスにします。以下の文字列は、普通に変換してくれます。日付の後ろの"(水)"などの曜日は無視されます。

コード 4.3 (date-ymd.R) : 文字列を日付データに変換

```

c("2024 年 4 月 10 日", "2024-4-10", "20240410", "2024/4/10") |>
  ymd()
## [1] "2024-04-10" "2024-04-10" "2024-04-10" "2024-04-10"
c("2024 年 4 月 10 日(水)", "2024-4-10(水)", "20240410(水)", "2024/4/10(水)") |>
  ymd()
## [1] "2024-04-10" "2024-04-10" "2024-04-10" "2024-04-10"

```

年が入っていない場合はエラーになります。その場合は、年を追加してください。

コード 4.4 (date-ymd-add-yr.R) : 年のないときはエラー

```

c("4 月 10 日", "4/10") |>
  ymd()
## Warning: All formats failed to parse. No formats found.
## [1] NA NA
paste0("2024-", c("4 月 10 日", "4/10")) |>
  ymd()
## [1] "2024-04-10" "2024-04-10"

```

本章では扱いませんが、時刻の計算もうまくできます。

コード 4.5 (date-ymd-hms.R) : 時刻の計算

```

d <- ymd_hms("2024-4-10-9-00-00")
magrittr::add(d, 105 * 60) # 105 分後
## [1] "2024-04-10 10:45:00 UTC"
magrittr::subtract(d, 3 * 60) # 3 分前
## [1] "2024-04-10 08:57:00 UTC"

```

なお、`today()`で実行日の日付を得ることができます。引数 `tzone` を指定するとタイムゾーンを指定でき、日本の場合は `today("Asia/Tokyo")`、グリニッジ標準時の場合は、`today("GMT")`とします。

コード 4.6 (date-today.R) : 実行時の日付

```
today() # 実行日によって結果は異なる
## [1] "2024-07-05"
```

4.3 1月後・1年後の同一日

1月後や1年後の同一の日付を得たいとします。月の大小や閏年がややこしいです。1月後の日付は`+ months(1)`とします。1年後の日付には`+ years(1)`とします。閏年にも対応しています。該当の日付がない場合は、返り値は `NA` になります。

コード 4.7 (date-months-years.R) : 日付を操作する関数

```
day_base <- ymd("2024-04-03")
day_base + months(1:4) # months()は base の関数
## [1] "2024-05-03" "2024-06-03" "2024-07-03" "2024-08-03"
day_base + years(1:4) # 1-4 年後まで
## [1] "2025-04-03" "2026-04-03" "2027-04-03" "2028-04-03"
day_base + days(365 * 1:4) # + years()とは異なる
## [1] "2025-04-03" "2026-04-03" "2027-04-03" "2028-04-02"
ymd("2024-02-29") + years(0:4) # 該当日がないときは NA
## [1] "2024-02-29" NA NA NA "2028-02-29"
```

4.4 曜日の取得

`wday()`で指定した日付の曜日を求められます。ただし、既定値では日曜日を 1、月曜日を 2 のように日曜始まり序数を示します。`label = TRUE` とすると、`factor` としての曜日を返します。`locale`(地域設定)を指定しないと OS や R での既定値の設定が使われます。地域設定を変更するには、`locale` で指定します。

コード 4.8 (date-wday.R) : 曜日を求める

```
wday(day_base) # week of the day
## [1] 4
wday(day_base, label = TRUE)
## [1] 水
## Levels: 日 < 月 < 火 < 水 < 木 < 金 < 土
wday(day_base, label = TRUE, locale = "EN-us")
```

```
## [1] Wed
## Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

4.5 日付っぽい文字列

文書の中では"2024年5月1日"のようにわかりやすく日付が表示されているとは限りません。西暦のこともあります、和暦のこともあります。西暦での年は下2桁しか使っていないこともあります。和暦の元号は"令和6年"を"R6"のように省略することもあります。また、そもそも年を省略することさえあります。このような日付っぽい文字列に全て対応することはできませんが、正規表現を使えば多くの表現に対応できます。海外では「月・日・年」のような順序での表記もよくありますが、ここでは「年・月・日」の順序での表記のみとします。日付っぽい文字列を抽出できれば、それを日付クラスに変換したり、1年後の日付を取得したり、曜日との対応を見つかりするのは簡単です。

まずは、年について考えます。和暦では行頭に元号かその省略形があります。西暦では4桁あるいは2桁の数字が普通ですが、年がない時もあるので0回から4回までの数字とし、その後ろに"年"あるいは区切り文字とします。正規表現では数字は"\d"と表せます。なお、"\d"は半角数字にも全角数字にもマッチします。ここでは、区切り文字は"."(ドット), "-"(ハイフン), "_"(アンダーバー), "/"(スラッシュ)のいずれかとします。念のため全角文字にも対応させます。

コード 4.9 (date-date-ish-yr.R) : 元号と年の正規表現

```
era <- "([MTSHRM T S H R]|明治|大正|昭和|平成|令和)?"
yr <- "[\d 元]{0,4}[-.. __/年]?"
```

月や日も同様に考えますが、数字が1回または2回です。月の後ろの"月"あるいは区切り文字は必須とします。

コード 4.10 (date-date-ish-md.R) : 月と日の正規表現

```
mn <- "\d{1,2}[-.. __/月]"
dy <- "\d{1,2}日?"
```

曜日は半角か全角のカッコで囲われることが多いので、それに対応させます。曜日はない場合があるので、0回または1回の意味の "?"を追加しています。

コード 4.11 (date-date-ish-dw.R) : 曜日の正規表現

```
dw <- "([\(\ ([月火水木金土日祝]+[\]) ]))?"
```

《Tips》 正規表現でどこまで幅広く対応するのかは難しいところです。厳密な定義をするには、長いあるいは難しい正規表現が必要です。一方、定義がゆるいと日付以外にもマッチしてしまい

ます。上記ではわかりやすさを優先して"\d"を使い、日付以外にもマッチすることは受け入れます。"\d"では全角の数字にもマッチしますので、全角文字が日付に入っていそうなときには便利です。

区切り文字なしで、数字だけで日付を表すことがあります。ただし、その場合は日付以外にマッチしないように、やや厳密な正規表現を使います。たとえば、"0[1-9]|1[0-2]"は"01"から"12"まではマッチしますが、それよりも大きな数字にはマッチしません。これをもとに、数字のみでの日付の正規表現をつくります。

コード 4.12 (date-date-ish-pattern.R) : 数字のみでの日付の正規表現

```
mn_dy <- "(0[1-9]|[12][0-9]|3[01])" # 月日
yr_4 <- "(19|20)?[0-9]{2}"           # 2 衡か 4 衡の年
paste0(mn_dy)                      # 月日のみ
## [1] "(0[1-9]|[12][0-9]|3[01])"
paste0(yr_4, mn_dy)                # 2 衡か 4 衡の年と月日
## [1] "(19|20)?[0-9]{2}(0[1-9]|[12][0-9]|3[01])"
```

これまでの内容をまとめます。

コード 4.13 (date-date-ish-fun.R) : 日付っぽい文字列の正規表現を返す関数

```
date_ish <- function(){
  era <- "([MTSHRM T S H R]|明治|大正|昭和|平成|令和)?"
  yr <- "[\d 元]{0,4}[-..__//年]?"
  mn <- "\d{1,2}[-..__//月]"
  dy <- "\d{1,2}日?"
  dw <- "([\(\ )[月火水木金土日祝]+[\)])?"
  mn_dy <- "(0[1-9]|[12][0-9]|3[01])" # 月日
  yr_4 <- "(19|20)?[0-9]{2}"           # 2 衡か 4 衡の年
  p_1 <- paste0(era, yr, mn, dy, dw)
  p_2 <- paste0(mn_dy)                 # 数字のみの月日
  p_3 <- paste0(yr_4, mn_dy)          # 数字のみの年月日
  pattern <- paste(p_1, p_2, p_3, sep = "|")
  return(pattern)
}
```

日付っぽい文字列を使って、関数の動作を確認します。その動作確認用の文字列を作ります。和暦と西暦、2 衡と 4 衡の西暦年、年の有無、全角文字と半角文字などを組み合わせています。全体で 40 のうち、12 が和暦、28 が西暦です。

コード 4.14 (date-for-check.R) : 日付っぽい文字列

```

dates <- c("令和 6 年 5 月 26 日", "令和 6.5.26", "R6.05.26",
         "2024 年 7 月 23 日", "2024-7-23", "24.7.23",
         "20240723", "240723", "11.27", "1127")
dates_half <- c(dates, paste0(dates, "(日)"))
dates_full <- stringi::stri_trans_general(dates_half, "halfwidth-fullwidth")
dates <- c(dates_half, dates_full)
dates
## [1] "令和 6 年 5 月 26 日"           "令和 6.5.26"
## [3] "R6.05.26"                      "2024 年 7 月 23 日"
## [5] "2024-7-23"                     "24.7.23"
## [7] "20240723"                     "240723"
## [9] "11.27"                         "1127"
## [11] "令和 6 年 5 月 26 日(日)"       "令和 6.5.26(日)"
## [13] "R6.05.26(日)"                  "2024 年 7 月 23 日(日)"
## [15] "2024-7-23(日)"                 "24.7.23(日)"
## [17] "20240723(日)"                 "240723(日)"
## [19] "11.27(日)"                    "1127(日)"
## [21] "令和 6 年 5 月 26 日"          "令和 6.5.26"
## [23] "R6.05.26"                     "2024 年 7 月 23 日"
## [25] "2024-7-23"                    "24.7.23"
## [27] "20240723"                    "240723"
## [29] "11.27"                        "1127"
## [31] "令和 6 年 5 月 26 日 (日)"    "令和 6.5.26 (日)"
## [33] "R6.05.26 (日)"                "2024 年 7 月 23 日 (日)"
## [35] "2024-7-23 (日)"               "24.7.23 (日)"
## [37] "20240723 (日)"                "240723 (日)"
## [39] "11.27 (日)"                  "1127 (日)"

```

日付っぽい文字列がどのようにマッチするのか確認します。また、全角の数字だけのものとこれに曜日を追加したものは、6つありこれらはマッチしないはずです。

コード 4.15 (date-str-detect.R) : 日付っぽい文字列の動作確認

```

# stringr::str_view(dates, date_ish()) |> print(n = Inf)
length(dates) # 全体の数
## [1] 40
stringr::str_detect(dates, date_ish()) |> sum() # マッチした数
## [1] 34
str_view(dates, date_ish()) |>
  print(n = 10)
## [1] | <令和 6 年 5 月 26 日>
## [2] | <令和 6.5.26>
## [3] | <R6.05.26>
## [4] | <2024 年 7 月 23 日>
## [5] | <2024-7-23>
## [6] | <24.7.23>
## [7] | <20><24><07><23>
## [8] | <24><07><23>
## [9] | <11.27>

```

```

## [10] | <11><27>
## ... and 24 more
stringr::str_subset(dates, date_ish(), negate = TRUE) # マッチせず
## [1] "20240723"      "240723"
## [3] "1127"           "20240723 (日)"
## [5] "240723 (日)"    "1127 (日)"

```

意図したとおりに文字列が日付っぽい文字列とマッチします。マッチしていないものは、全角文字列の数字だけのものです。

stringr パッケージの `str_extract_all()` で、文字列から日付っぽい表現を抽出する関数を定義します。

コード 4.16 (date-extract-date-ish-fun.R) : 日付っぽい文字列を抽出する関数

```

extract_date_ish <- function(str, simplify = FALSE){
  pattern <- date_ish()
  res <-
    stringr::str_extract_all(str, pattern = pattern, simplify = simplify)
  if(length(res) == 1){
    res <- res[[1]]
  }
  return(res)
}

```

動作確認をします。正しくは、34 が抽出できるはずです。

コード 4.17 (date-extract-date-ish.R) : 日付っぽい文字列の抽出

```

paste(dates, collapse = "◆") |> #
  print() |>
  extract_date_ish() |>
  length()
## [1] "令和6年5月26日◆令和6.5.26◆R6.05.26◆2024年7月23日◆2024-7-23◆24.7.23
◆20240723◆240723◆11.27◆1127◆令和6年5月26日(日)◆令和6.5.26(日)◆R6.05.26(日)◆202
4年7月23日(日)◆2024-7-23(日)◆24.7.23(日)◆20240723(日)◆240723(日)◆11.27(日)◆1127
(日)◆令和6年5月26日◆令和6.5.26◆R6.05.26◆2024年7月23日◆202
4-7-23◆24.7.23◆20240723◆240723◆11.27◆1127◆令和6年
5月26日(日)◆令和6.5.26(日)◆R6.05.26(日)◆2024年7月23日
(日)◆2024-7-23(日)◆24.7.23(日)◆20240723(日)◆24072
3(日)◆11.27(日)◆1127(日)"
## [1] 46

```

途中に文字列が挟まっても正しく抽出できますので、`extract_date_ish()` 使うと、テキストファイルから日付っぽい文字列を抽出可能です。

4.6 和暦への対応

日付っぽい文字列が抽出できれば、それを日付クラスに変換します。ただし、`lubridate::ymd()`では、和暦は正しい日付クラスに変換できません。令和6年と2024年は同じ年ですが、令和6年は2006年として認識されてしまいます。

コード 4.18 (`date-ymd-not-work.R`) : `ymd()`は和暦に対応していない

```
c("令和 06 年 01 月 01 日", "2024-01-01") |>  
  ymd()  
## [1] "2006-01-01" "2024-01-01"
```

和暦の文字列は、`zipangu`パッケージの`convert_jdate()`で日付クラスに変換できます。

コード 4.19 (`date-convert-jdate.R`) : 和暦の変換

```
# 和暦は OK  
dates[c(1:3, 11:13, 21:23, 31:33)] |>  
  print() |>  
  zipangu::convert_jdate()  
## [1] "令和 6 年 5 月 26 日"          "令和 6.5.26"  
## [3] "R6.05.26"                      "令和 6 年 5 月 26 日(日)"  
## [5] "令和 6.5.26(日)"                "R6.05.26(日)"  
## [7] "令和 6 年 5 月 26 日"            "令和 6. 5 . 2 6 "  
## [9] "R 6 . 0 5 . 2 6 "                "令和 6 年 5 月 26 日 ( 日 )"  
## [11] "令和 6 . 5 . 2 6 ( 日 )"         "R 6 . 0 5 . 2 6 ( 日 )"  
## [1] "2024-05-26" "2024-05-26" "2024-05-26" "2024-05-26" "2024-05-26"  
## [6] "2024-05-26" "2024-05-26" "2024-05-26" "2024-05-26" "2024-05-26"  
## [11] "2024-05-26" "2024-05-26"  
# 西暦はダメ  
dates[c(4:6, 14:16, 24:26, 34:36)] |>  
  print() |>  
  zipangu::convert_jdate()  
## [1] "2024 年 7 月 23 日"              "2024-7-23"  
## [3] "24.7.23"                       "2024 年 7 月 23 日(日)"  
## [5] "2024-7-23(日)"                  "24.7.23(日)"  
## [7] "2 0 2 4 年 7 月 2 3 日"        "2 0 2 4 - 7 - 2 3 "  
## [9] "2 4 . 7 . 2 3 "                 "2 0 2 4 年 7 月 2 3 日 ( 日 )"  
## [11] "2 0 2 4 - 7 - 2 3 ( 日 )"       "2 4 . 7 . 2 3 ( 日 )"  
## [1] "4042-07-23" "4042-07-23" "2042-07-23" "4042-07-23" "4042-07-23"  
## [6] "2042-07-23" "4042-07-23" "4042-07-23" "2042-07-23" "4042-07-23"  
## [11] "4042-07-23" "2042-07-23"
```

`convert_jdate()`は全角文字には対応していますが、西暦はうまく変換できません。そこで、西暦は`lubridate`で、和暦は`zipangu`で対応します。条件整理のために和暦の判別をする関数を定義します。行頭に元号か省略形さらに数字か"元"があれば`TRUE`を返します。

コード 4.20 (date-is-jp-date-fun.R) : 和暦判別の関数

```
is_jp_date <- function(str){  
  era <- "^( [MTSHRM T S H R ] | 明治 | 大正 | 昭和 | 平成 | 令和 ) [ \\\d 元 ] "  
  stringr::str_detect(str, era)  
}
```

定義した関数を試します。なお、dates には和暦が 12 あります。

コード 4.21 (date-is-jp-date.R) : 和暦の確認

```
dates[is_jp_date(dates)]  
## [1] "令和 6 年 5 月 26 日"          "令和 6.5.26"  
## [3] "R6.05.26"                      "令和 6 年 5 月 26 日(日)"  
## [5] "令和 6.5.26(日)"                "R6.05.26(日)"  
## [7] "令和 6 年 5 月 26 日"          "令和 6. 5. 26"  
## [9] "R6.05.26"                      "令和 6 年 5 月 26 日(日)"  
## [11] "令和 6. 5. 26 (日)"            "R6.05.26 (日)"
```

和暦のものは全部で 12 あり、すべて和名として判定できたことがわかります。

これで和暦と西暦の区別ができます。ただし、`ymd()`は全角文字に対応しないため、`stringi::stri_trans_general(str, "fullwidth-halfwidth")`で半角文字に変換します。また、`ymd()`のエラー出力を抑制するために、`quiet = TRUE` を指定します。簡潔に表示するために名前付きリストにして、構造(structure)を示す `str()`を使っています。

コード 4.22 (date-is-jp-date-str.R) : 和暦と西暦の判別

```
dates_half <- stringi::stri_trans_general(dates, "fullwidth-halfwidth")  
converted <-  
  dplyr::if_else(is_jp_date(dates_half),           # 和暦と西暦の判別  
                 zipangu::convert_jdate(dates_half), # 和暦  
                 ymd(dates_half, quiet = TRUE)) |>    # 西暦  
  rlang::set_names(dates_half) |>           # 名前付ける  
  as.list()                                # リストに変換  
str(converted[1:20])  
## List of 20  
## $ 令和 6 年 5 月 26 日   : Date[1:1], format: "2024-05-26"  
## $ 令和 6.5.26           : Date[1:1], format: "2024-05-26"  
## $ R6.05.26              : Date[1:1], format: "2024-05-26"  
## $ 2024 年 7 月 23 日    : Date[1:1], format: "2024-07-23"  
## $ 2024-7-23             : Date[1:1], format: "2024-07-23"  
## $ 24.7.23               : Date[1:1], format: "2024-07-23"  
## $ 20240723              : Date[1:1], format: "2024-07-23"  
## $ 240723                : Date[1:1], format: "2024-07-23"
```

```

## $ 11.27           : Date[1:1], format: "2011-02-07"
## $ 1127            : Date[1:1], format: NA
## $ 令和 6 年 5 月 26 日(日): Date[1:1], format: "2024-05-26"
## $ 令和 6.5.26(日)   : Date[1:1], format: "2024-05-26"
## $ R6.05.26(日)     : Date[1:1], format: "2024-05-26"
## $ 2024 年 7 月 23 日(日) : Date[1:1], format: "2024-07-23"
## $ 2024-7-23(日)    : Date[1:1], format: "2024-07-23"
## $ 24.7.23(日)      : Date[1:1], format: "2024-07-23"
## $ 20240723(日)    : Date[1:1], format: "2024-07-23"
## $ 240723(日)      : Date[1:1], format: "2024-07-23"
## $ 11.27(日)        : Date[1:1], format: "2011-02-07"
## $ 1127(日)         : Date[1:1], format: NA

```

\$のすぐ右側が変換前の `dates_half` で、右端の文字列が変換後の `converted` の値です。年のないものが変換できていないことがわかります。

4.7 年の追加

年のないものに対応するため、年を追加します。ただし、年を追加する前に、年の有無を判別する関数を定義します。

まず、`stringi` の `stri_trans_general()` を使って、全角文字を半角文字に修正し、曜日と最後の年月日を示す"日"を削除します。その後、数字だけの場合は桁数で年の有無を判別します。数字以外を含む場合は、数字以外の区切り文字の数で年の有無を判別します。

コード 4.23 (date-has-yr-fun.R) : 年の有無の判別関数

```

has_yr <- function(str){
  dw <- "\\([月火水木金土日祝]+\\)$"
  str <-
    str |>
    stringi::stri_trans_general("fullwidth-halfwidth") |>
    stringr::str_remove(dw) |>                                # 曜日を削除
    stringr::str_remove("日$")                                    # 最後の"日"を削除
  res <-
    dplyr::if_else(stringr::str_count(str, "[^0-9]") == 0,  # [^0-9] : 数字以外
                  dplyr::if_else(stringr::str_count(str, "[0-9]") >= 6, # 数字のみ
                                         TRUE,                                # 6 桁以上
                                         FALSE),                                # 5 桁以下
                  dplyr::if_else(stringr::str_count(str, "[^0-9]") >= 2, # 数字以外あり
                                         TRUE,                                # 区切り文字が 2 つ以上 : 年あり
                                         FALSE))                                # 区切り文字が 1 つ : 年なし
    )
  return(res)
}

```

関数を実行すると、年の有無を判別できていることがわかります。

コード 4.24 (date-has-yr.R) : 年の有無の判別

```
dates_west <- dates[!is_jp_date(dates)] # 西暦のみ
dates_west[has_yr(dates_west)] # 年あり
## [1] "2024年7月23日"           "2024-7-23"
## [3] "24.7.23"                 "20240723"
## [5] "240723"                  "2024年7月23日(日)"
## [7] "2024-7-23(日)"           "24.7.23(日)"
## [9] "20240723(日)"            "240723(日)"
## [11] "2024年7月23日"           "2024-7-23"
## [13] "24.7.23"                 "20240723"
## [15] "240723"                  "2024年7月23日(日)"
## [17] "2024-7-23(日)"          "24.7.23(日)"
## [19] "20240723(日)"           "240723(日)"
dates_west[!has_yr(dates_west)] # 年なし
## [1] "11.27"                   "1127"                  "11.27(日)"
## [4] "1127(日)"                "11.27"                 "1127"
## [7] "11.27(日)"               "1127(日)"
```

コード実行時に作成する文書には、実行時の年を追加するのが一般的です。一方、既存の文書のときには、過去1年間のうちの日付が記載されていることが普通かもしれません。そこで、コードの実行時の前後1年間のどちらでも年を追加できるようにします。

関数全体を作る前に、今日の日付を返す `today()` のように今年を返す関数として `this_year()` と、未来の日付であるかを判定する `is_future()` を定義します。`this_year()` は、`today()` から年を取り出すだけです。`is_future()` は、`today()` と日付を比較するだけです。

コード 4.25 (date-paste-year-helper-fun.R) : 年追加の助関数

```
this_year <- function(){
  lubridate::today() |>
    lubridate::year()
}

is_future <- function(date){
  lubridate::today() < date
}
```

実行時の年を追加して日付クラスに変換します。その後、変換した日付と目的の日付を比較して、年の調整をします。

コード 4.26 (date-paste-year-fun.R) : 年の追加関数

```

paste_year <- function(str, past = FALSE){
  str <- stringi::stri_trans_general(str, "fullwidth-halfwidth")
  yr <- this_year()
  date <- lubridate::ymd(paste0(yr, "-", str), quiet = TRUE)
  date <-
    dplyr::if_else(past & is_future(date),      # 目的：過去, 変換：未来
                  date - lubridate::years(1),        # 1年前
                  date)                           # そのまま
  )
  date <-
    dplyr::if_else(!past & !is_future(date), # 目的：未来, 変換：過去
                  date + lubridate::years(1),        # 1年後
                  date)                           # そのまま
  )
  return(date)
}

```

年を追加する関数を実行します。ただし、実行日との関係で出力される年は異なります。

コード 4.27 (date-paste-year.R) : 年の追加

```

# today()が 2024 年 1 月 2 日-12 月 30 日のとき
str <- "12-31"
paste_year(str)
## [1] "2024-12-31"
paste_year(str, past = TRUE)
## [1] "2023-12-31"
str <- "1-1"
paste_year(str)
## [1] "2024-1-1"
paste_year(str, past = TRUE)
## [1] "2023-1-1"

```

作成した関数が正しく動いてくれました。

4.8 日付っぽい文字列の抽出

ここまでの中をまとめて、日付っぽい文字列を日付に変換する関数をつくります。といっても、条件を整理してそれぞれに合致する関数を使うだけです。和暦は zipangu の convert_jdate() で変換します。また、年のある場合は lubridate の ymd() で変換します。最後に年のないものは、paste_year() で年を追加してから変換します。

コード 4.28 (date-date-ish2date-fun.R) : 日付っぽい文字列を日付に変換する関数

```

date_ish2date <- function(str, past = FALSE){
  str <- stringi::stri_trans_general(str, "fullwidth-halfwidth")
  str <-
    dplyr::if_else(is_jp_date(str),           # 和暦 or 西暦
                  zipangu::convert_jdate(str),   # 和暦を日付に変換
                  dplyr::if_else(has_yr(str),    # 西暦, 年の有無
                                  lubridate::ymd(str, quiet = TRUE), # 日付に変換
                                  paste_year(str, past = past))  # 年を追加して日付に変換
    )
  )
  return(str)
}

```

実際に日付のデータを1つの文字列として結合してから、日付っぽいものを抽出して、日付に変換します。

コード 4.29 (date-date-ish2date.R) : 日付っぽい文字列の日付への変換

```

converted_dates <-
  dates_half |>
  paste(collapse = "間の文字など") |> # 結合
  extract_date_ish() |>               # 抽出
  date_ish2date()                   # 変換

tibble::tibble(dates_half, converted_dates)
## # A tibble: 40 × 2
##   dates_half      converted_dates
##   <chr>          <date>
## 1 令和6年5月26日 2024-05-26
## 2 令和6.5.26     2024-05-26
## 3 R6.05.26       2024-05-26
## 4 2024年7月23日 2024-07-23
## 5 2024-7-23      2024-07-23
## 6 24.7.23        2024-07-23
## 7 20240723       2024-07-23
## 8 240723         2024-07-23
## 9 11.27          2024-11-27
## 10 1127          2024-11-27
## # i 30 more rows
## # i Use `print(n = ...)` to see more rows

```

4.9 西暦と和暦の相互変換

和暦から西暦への変換は、zipangu の関数を使います。convert_jdate()は日付をconvert_jyear()は年を変換します。全角の文字列が含まれていても大丈夫です。

コード 4.30 (date-zipangu.R) : 和暦から西暦への変換

```
zipangu::convert_jdate(c("H29,1,1", "R6/10/30"))
## [1] "2017-01-01" "2024-10-30"
zipangu::convert_jyear(c("S63", "平成 4 年 5 月 1 日"))
## [1] 1988 1992
```

西暦から和暦への変換は、`stringi::stri_datetime_format()`で変換可能です。引数 `time` に日付オブジェクトを、`format` に出力形式を、`tz` には標準時間帯(タイムゾーン)を、`locale` には地域設定をそれぞれ指定します。

重要な引数は `locale` と `format` です。和暦を使うには、`locale` を"ja_JP@calendar=japanese"と指定します。`format` は出力の順序や桁数などを決めます。`format` の既定値は"uuuu-MM-dd HH:mm:ss"で、"2024-05-01 09:00:00"のように表示されます。年を元号と2桁の数字にするには、"Gyy"("G"は元号)を使います。"令和 6 年 5 月 15 日(水)"のようにするには、"Gy 年 M 月 d 日(E)"とします。

西暦から和暦への変換はややこしい問題を含みます。年の途中で元号が変わっているためですが、`stringi::stri_datetime_format` は正しく変換してくれます。

コード 4.31 (date-stri-datetime-format.R) : 西暦から和暦への変換

```
c("2019-04-30", "2019-05-01") |>
  ymd() |>
  stringi::stri_datetime_format(format = "Gy 年 M 月 d 日(E)",
                                 locale = "ja_JP@calendar=japanese")
## [1] "平成 31 年 4 月 30 日(火)" "令和元年 5 月 1 日(水)"
```

次に、西暦と和暦の年の対応を返す関数を定義します。元号が変わるときがややこしいので、ここでは全て 12 月 31 日で考えます。

コード 4.32 (date-convert-yr-fun.R) : 西暦年と和暦年を変換する関数

```
convert_yr <- function(str, out_format = "west"){
  no_nen <- stringr::str_which(str, "年", negate = TRUE) # "年"無の序数
  str <-
    paste0(str, "-12-31") |>                      # 12 月 31 日として処理
    stringr::str_remove("年") |>                   # "年"の除去
    date_ish2date() |>
    format_year(out_format = out_format) # 変換
  nen <- rep("年", times = length(str))
  nen[no_nen] <- ""                                # 年の有無に合わせる
  str <- paste0(str, nen)
  return(str)
}
```

```

# 和暦年と西暦年の書式
format_year <- function(x, out_format = "west"){
  if(out_format == "west"){
    format <- "uuuu"
    locale <- NULL
  }
  if(out_format == "jp"){
    format <- "Gy"
    locale <- "ja_JP@calendar=japanese"
  }
  x <- stringi::stri_datetime_format(x, format = format, locale = locale)
  return(x)
}

```

《Tips》 もとの文書に全角文字の数字があって全て半角で処理するには、`stri_trans_general(str, "fullwidth-halfwidth")`で入力文字列を半角にします(4.7節を参照)。逆に、全角にするには、`stri_trans_general(str, "halfwidth-fullwidth")`を使います。

定義した関数を試してみます。

コード 4.33 (date-convert-yr.R) : 西暦年と和暦年の変換

```

str <- c("昭和 50", "1992", "令和元年", "2024 年")
convert_yr(str, out_format = "jp")
## [1] "昭和 50" "平成 4"   "令和 1 年" "令和 6 年"
convert_yr(str, out_format = "west")
## [1] "1975"   "1992"   "2019 年" "2024 年"

```

入力の年の有無に合わせた形で変換されています。

実際の文章の中では特定の年だけを変換することもあるでしょうが、しらみつぶし的に一括で和暦を西暦に変換することがあるでしょう。その場合は、`stringr::str_replace_all()`を使います。`for` ループを使っても構いませんが、名前付きベクトルを使うとコードが簡潔になります。`purrr::reduce2()`を使う方法もあります。

コード 4.34 (date-convert-yr-replace.R) : 和暦・西暦の一括変換

```

yr_west <- paste0(as.character(1950:2024), "年")
yr_jp <- convert_yr(yr_west, out_format = "jp")
sentence <- "昭和 48 年生まれは、平成 30 年で 45 歳、令和 5 年で 50 歳です。
          昭和 50 年生まれは、平成 30 年で 43 歳、令和 5 年で 48 歳です。" |>
  stringr::str_remove_all(" ") # 空白を削除
  # 置換するとき
pattern <- yr_west

```

```

names(pattern) <- yr_jp # ベクトルに名前を付ける
stringr::str_replace_all(sentence, pattern) |>
  cat()
## 1973 年生まれは、2018 年で 45 歳、2023 年で 50 歳です。
## 1975 年生まれは、2018 年で 43 歳、2023 年で 48 歳です。
# 併記するとき
pattern <- paste0(stringr::str_remove(yr_west, "年"),
                    "( ", yr_jp, " )")
names(pattern) <- yr_jp # ベクトルに名前を付ける
stringr::str_replace_all(sentence, pattern) |>
  cat()
## 1973(昭和 48 年)生まれは、2018(平成 30 年)で 45 歳、2023(令和 5 年)で 50 歳です。
## 1975(昭和 50 年)生まれは、2018(平成 30 年)で 43 歳、2023(令和 5 年)で 48 歳です。

```

ワードで本文中の文字列を変換するときには、`officer::body_replace_all_text()`を使います（8.5 節を参照）。`body_replace_all_text()`は名前付きベクトルには対応していないので、`reduce2()`か `for` ループを使ってください。

コード 4.35 (date-convert-yr-replace-word.R) : ワードでの一括変換(擬似コード)

```

path <- "DIRECTORY/word.docx"
doc <- officer::read_docx(path)
doc <- purrr::reduce2(.x = yr_jp, .y = yr_west,
                      .f = officer::body_replace_all_text, .init = doc)

```

4.10 曜日の取り出し

文字列を日付クラスに変換すると、もともと含んでいた文字列としての曜日は削除されます。記載の曜日が正しいか確認するには、もとの曜日を取得する必要がありますので、その関数を定義します。年月日の月と日が、曜日の日と月の文字と同じである点がややこしいです。そのため、`str_remove()`で曜日ではない部分と括弧を取り除いてから、曜日にあたる"[月火水木金土日]"を取り出します。

コード 4.36 (date-extract-wday-fun.R) : 曜日を取り出す関数

```

extract_wday <- function(str){
  str <- stringi::stri_trans_general(str, "fullwidth-halfwidth")
  mn <- "\\\d{1,2}[-.,/_月]" # 月
  dy <- "\\\d{1,2}日?" # 日
  dw <- "[月火水木金土日]" # 曜日
  md <- paste0(mn, dy)
  wd <-
    str |>

```

```

stringr::str_remove(md) |> # 月日の除去
stringr::str_remove_all("[\\(\\)]") |> # ()の除去
stringr::str_extract(dw) # 曜日の抽出
return(wd)
}

```

作った関数で曜日を取り出します。dates のうち曜日のあるものは、全体の半分の 20 です。

コード 4.37 (date-extract-wday.R) : 曜日の取り出し

```

wd <- extract_wday(dates)
names(wd) <- dates
wd[!is.na(wd)] |> # NA 以外
as.list() |> # リストに変換
str() # 構造を表示
## List of 20
## $ 令和6年5月26日(日)      : chr "日"
## $ 令和6.5.26(日)          : chr "日"
## $ R6.05.26(日)            : chr "日"
## $ 2024年7月23日(日)       : chr "日"
## $ 2024-7-23(日)           : chr "日"
## $ 24.7.23(日)             : chr "日"
## $ 20240723(日)            : chr "日"
## $ 240723(日)              : chr "日"
## $ 11.27(日)               : chr "日"
## $ 1127(日)                : chr "日"
## $ 令和6年5月26日(日)      : chr "日"
## $ 令和6.5.26(日)          : chr "日"
## $ R6.05.26(日)            : chr "日"
## $ 2024年7月23日(日)       : chr "日"
## $ 2024-7-23(日)           : chr "日"
## $ 24.7.23(日)             : chr "日"
## $ 20240723(日)            : chr "日"
## $ 240723(日)              : chr "日"
## $ 11.27(日)               : chr "日"
## $ 1127(日)                : chr "日"

```

4.11 日付と曜日との整合性の確認

日付っぽい文字列の日付クラスへの変換と、曜日の取り出しができるようになりましたので、日付と曜日の整合性を確認する関数を作ります。日付っぽい文字列から元の曜日を取り出すとともに、日付クラスに変換してから正しい曜日を取り出します。元の曜日と正しい曜日の文字列を比較して、正しい曜日か判定します。また、入力や正しい曜日を一覧にして返します。

コード 4.38 (date-is-correct-wday-fun.R) : 日付と曜日との整合性を確認する関数

```
is_correct_wday <- function(str){  
  wday_orig <- extract_wday(str) # 元の曜日  
  date <- date_ish2date(str)      # 日付  
  wday <-                                         # 日付にあう曜日  
    date |>  
    lubridate::wday(label = TRUE, locale = "Japanese_Japan.utf8") |>  
    as.character()  
  is_correct <- (wday == wday_orig) # 正しいか  
  res <- list(is_correct = is_correct, date_orig = str,  
             wday_orig = wday_orig, date = date, wday = wday)  
  return(res)  
}
```

ここでは dates の曜日の確認をします。曜日と日付の対応が間違っていることがわかります。

コード 4.39 (date-is-correct-wday.R) : 日付と曜日との整合性の確認

```
is_correct_wday(dates) |>  
  tibble::as_tibble() |>  
  na.omit() |>  
  print(n = 20)  
## # A tibble: 20 × 5  
##   is_correct date_orig          wday_orig date       wday  
##   <lgl>     <chr>            <chr>      <date>     <chr>  
## 1 TRUE      令和6年5月26日(日)    日        2024-05-26 日  
## 2 TRUE      令和6.5.26(日)       日        2024-05-26 日  
## 3 TRUE      R6.05.26(日)        日        2024-05-26 日  
## 4 FALSE     2024年7月23日(日)    日        2024-07-23 火  
## 5 FALSE     2024-7-23(日)       日        2024-07-23 火  
## 6 FALSE     24.7.23(日)        日        2024-07-23 火  
## 7 FALSE     20240723(日)       日        2024-07-23 火  
## 8 FALSE     240723(日)        日        2024-07-23 火  
## 9 FALSE     11.27(日)         日        2024-11-27 水  
## 10 FALSE    1127(日)          日        2024-11-27 水  
## 11 TRUE     令和6年5月26日(日)  日        2024-05-26 日  
## 12 TRUE     令和6.5.26(日)      日        2024-05-26 日  
## 13 TRUE     R6.05.26(日)       日        2024-05-26 日  
## 14 FALSE    2024年7月23日(日)  日        2024-07-23 火  
## 15 FALSE    2024-7-23(日)      日        2024-07-23 火  
## 16 FALSE    24.7.23(日)        日        2024-07-23 火  
## 17 FALSE    20240723(日)      日        2024-07-23 火  
## 18 FALSE    240723(日)        日        2024-07-23 火  
## 19 FALSE    11.27(日)         日        2024-11-27 水  
## 20 FALSE    1127(日)          日        2024-11-27 水
```

これまでの内容をもとにして、曜日を修正する関数を定義します。曜日が正しいか判定とともに、指定した書式に従って日付を返します。書式(out_format)は、"west"が既定値で西暦に統一、"jp"が和暦に統一、"original"がもとの書式のまま曜日のみ置換します。

コード 4.40 (date-update-wday-fun.R) : 曜日を修正する関数

```
update_wday <- function(str, out_format = "west"){
  res <- is_correct_wday(str) # 曜日が正しいか判定
  if(out_format == "original"){ # 元の書式
    date <- replace_wday(str, res$wday_orig, res$wday)
  }else{ # 和暦か西暦
    date <- format_date(res$date, out_format = out_format)
  }
  return(date)
}
```

書式(out_format)が"original"のとき、もの書式のまま曜日のみ置換する関数を定義します。

コード 4.41 (date-replace-wday-fun.R) : 元の書式のまま曜日のみ置換する関数

```
replace_wday <- function(str, wday_orig, wday){
  pattern <- paste0("([\\((\\)])", wday_orig, "([\\))])") # 置換前
  replacement <- paste0("\\1", wday, "\\2") # 置換後
  date <- stringr::str_replace(str, pattern, replacement)
  return(date)
}
```

書式(out_format)が"west"か"jp"のときに、西暦と和暦で日付を返す関数を定義します。

コード 4.42 (date-format-date-fun.R) : 日付を指定の書式にする関数

```
format_date <- function(x, out_format = "west"){
  if(out_format == "west"){ # 西暦
    format <- "uuuu 年 M 月 d 日(E)"
    locale <- NULL
  }
  if(out_format == "jp"){ # 和暦
    format <- "Gy 年 M 月 d 日(E)"
    locale <- "ja_JP@calendar=japanese"
  }
  x <- stringi::stri_datetime_format(x, format = format, locale = locale)
  return(x)
}
```

これで準備ができましたので、日付データで試します。

コード 4.43 (date-update-wday.R) : 曜日の修正

```
tibble::tibble(west = update_wday(dates, out_format = "west"),
               jp = update_wday(dates, out_format = "jp"),
               update = update_wday(dates, out_format = "original"),
               orig = dates) |>
print(n = 20)
## # A tibble: 40 × 4
##   west          jp        update      orig
##   <chr>        <chr>     <chr>       <chr>
## 1 2024 年 5 月 26 日(日) 令和 6 年 5 月 26 日(日) 令和 6 年 5 月 26 日    令和 6 年 5...
## 2 2024 年 5 月 26 日(日) 令和 6 年 5 月 26 日(日) 令和 6.5.26           令和 6.5....
## 3 2024 年 5 月 26 日(日) 令和 6 年 5 月 26 日(日) R6.05.26            R6.05.26
## 4 2024 年 7 月 23 日(火) 令和 6 年 7 月 23 日(火) 2024 年 7 月 23 日    2024 年 7 ...
## 5 2024 年 7 月 23 日(火) 令和 6 年 7 月 23 日(火) 2024-7-23           2024-7-23
## 6 2024 年 7 月 23 日(火) 令和 6 年 7 月 23 日(火) 24.7.23            24.7.23
## 7 2024 年 7 月 23 日(火) 令和 6 年 7 月 23 日(火) 20240723           20240723
## 8 2024 年 7 月 23 日(火) 令和 6 年 7 月 23 日(火) 240723            240723
## 9 2024 年 11 月 27 日(水) 令和 6 年 11 月 27 日(水) 11.27             11.27
## 10 2024 年 11 月 27 日(水) 令和 6 年 11 月 27 日(水) 1127            1127
## 11 2024 年 5 月 26 日(日) 令和 6 年 5 月 26 日(日) 令和 6 年 5 月 26 日(日) 令和 6 年 5...
## 12 2024 年 5 月 26 日(日) 令和 6 年 5 月 26 日(日) 令和 6.5.26(日)       令和 6.5....
## 13 2024 年 5 月 26 日(日) 令和 6 年 5 月 26 日(日) R6.05.26(日)         R6.05.26...
## 14 2024 年 7 月 23 日(火) 令和 6 年 7 月 23 日(火) 2024 年 7 月 23 日(火) 2024 年 7 ...
## 15 2024 年 7 月 23 日(火) 令和 6 年 7 月 23 日(火) 2024-7-23(火)       2024-7-2...
## 16 2024 年 7 月 23 日(火) 令和 6 年 7 月 23 日(火) 24.7.23(火)         24.7.23(...
## 17 2024 年 7 月 23 日(火) 令和 6 年 7 月 23 日(火) 20240723(火)       20240723...
## 18 2024 年 7 月 23 日(火) 令和 6 年 7 月 23 日(火) 240723(火)         240723(日)
## 19 2024 年 11 月 27 日(水) 令和 6 年 11 月 27 日(水) 11.27(水)         11.27(日)
## 20 2024 年 11 月 27 日(水) 令和 6 年 11 月 27 日(水) 1127(水)         1127(日)
## # i 20 more rows
## # i Use `print(n = ...)` to see more rows
```

実際の日付データの処理では、`stringr::str_remove_all()`を使って、`orig`列から`update`等の列に置換するコードによって、文書中の日付データの修正や統一ができます。

4.12 1年後の同一位置への更新

10月の第4土曜日に文化祭を毎年行うなど、日付は異なるが、毎年同じ時期に行うイベントがあると思います。このようなときのために、位置固定(m月のn番目のw曜日)のときでの1年後の年月日を求めることがあります。これとワードの操作(第8章)と合わせると、資料内の日付を自動で更新・確認できます。

コード 4.44 (date-years.R) : 1 年後の同一日付

```
x <- ymd("2024-05-01")
x + years(1)
## [1] "2025-05-01"
```

位置固定の場合は、年月日から第何の何曜日かを知る必要があります。曜日は `wday()` で求められるため、各曜日の何番目かつまり序数を求める関数が必要です。各月の1日から7日までは各曜日の第1の日付、8日から14日までは第2の日付です。日付に6を足して、7で割ったときの商の整数部分が各曜日での序数です。

コード 4.45 (date-mweek-fun.R) : 各曜日での序数を得る関数

```
mweek <- function(x){
  (lubridate::mday(x) + 6) %% 7
}
```

次に、年月日から1年後の年と月を分離してそこから求めたい月の1日を `base` の日付とします。`base` に `mweek(x) - 1) * 7` を足して、n番目の曜日の日付とします。さらに、これに曜日の補正をするため、`base` と元の日付(`x`)との曜日の差を追加します。ただし、差が負の場合は7から引いて正にします。なお、`for(i in seq_along(diff))` でループしている部分は、ベクトルへの対応です。

これで、一応出来上がりしました。ただし、第5の曜日の場合は、次の月にずれてしまっている可能性があります。そこで、月がズレていないか確認して、ズれている場合は `NA` を返します。

コード 4.46 (date-same-pos-next-yr-fun.R) : 1年後の同一位置の年月日を取得する関数

```
same_pos_next_yr <- function(x, out_format = "west"){
  yr <- lubridate::year(x) # 年
  mn <- lubridate::month(x) # 月
  base <- lubridate::ymd(paste0(yr + 1, "-", mn, "-", 1)) # 1日
  diff <- lubridate::wday(x) - lubridate::wday(base) # 曜日位置の差
  diff <- dplyr::if_else(diff >= 0, diff, diff + 7) # 負をは正に変換
  same_pos_day <- base + (mweek(x) - 1) * 7 + diff # 同じ位置
  diff <- dplyr::if_else(diff >= 0, diff, diff + 7) # 負のときは正に変換
  for(i in seq_along(same_pos_day)){
    if(month(same_pos_day[i]) != mn[i]){ # 月が異なるとき
      same_pos_day[i] <- NA # 該当日なし
      warning("No same date as ", x[i], "!")
    }
  }
  same_pos_day <- # 指定の書式に変換
  format_date(same_pos_day, out_format = out_format)
```

```

    return(same_pos_day)
}

```

実際の日付で確認してみます。カレンダーで見ると、曜日固定の1年後の日付が正しく計算されたことがわかります。

コード 4.47 (date-same-pos-next-yr.R) : 1年後の同一位置の年月日の取得

```

days <- x + 0:30
days_next_yr <- same_pos_next_yr(days)
## Warning in same_pos_next_yr(days): No same date as 2024-05-29!
days
## [1] "2024-05-01" "2024-05-02" "2024-05-03" "2024-05-04" "2024-05-05"
## [6] "2024-05-06" "2024-05-07" "2024-05-08" "2024-05-09" "2024-05-10"
## [11] "2024-05-11" "2024-05-12" "2024-05-13" "2024-05-14" "2024-05-15"
## [16] "2024-05-16" "2024-05-17" "2024-05-18" "2024-05-19" "2024-05-20"
## [21] "2024-05-21" "2024-05-22" "2024-05-23" "2024-05-24" "2024-05-25"
## [26] "2024-05-26" "2024-05-27" "2024-05-28" "2024-05-29" "2024-05-30"
## [31] "2024-05-31"
days_next_yr
## [1] "2025 年 5 月 7 日(水)" "2025 年 5 月 1 日(木)" "2025 年 5 月 2 日(金)"
## [4] "2025 年 5 月 3 日(土)" "2025 年 5 月 4 日(日)" "2025 年 5 月 5 日(月)"
## [7] "2025 年 5 月 6 日(火)" "2025 年 5 月 14 日(水)" "2025 年 5 月 8 日(木)"
## [10] "2025 年 5 月 9 日(金)" "2025 年 5 月 10 日(土)" "2025 年 5 月 11 日(日)"
## [13] "2025 年 5 月 12 日(月)" "2025 年 5 月 13 日(火)" "2025 年 5 月 21 日(水)"
## [16] "2025 年 5 月 15 日(木)" "2025 年 5 月 16 日(金)" "2025 年 5 月 17 日(土)"
## [19] "2025 年 5 月 18 日(日)" "2025 年 5 月 19 日(月)" "2025 年 5 月 20 日(火)"
## [22] "2025 年 5 月 28 日(水)" "2025 年 5 月 22 日(木)" "2025 年 5 月 23 日(金)"
## [25] "2025 年 5 月 24 日(土)" "2025 年 5 月 25 日(日)" "2025 年 5 月 26 日(月)"
## [28] "2025 年 5 月 27 日(火)" NA "2025 年 5 月 29 日(木)"
## [31] "2025 年 5 月 30 日(金)"

```

2024年4月1日の同一位置の日付を same_pos_next_yr() で計算した結果は、2025年4月7日です。カレンダーで表示してみると、どちらも4月の第1月曜日で正しいことが分かります。2024年4月29日は第5月曜日ですが、その同一位置の日付は2025年ではNAとなっています。カレンダーで見ると2025年は第5月曜日が無いためです。

コード 4.48 (date-gen-cal.R) : カレンダーでの確認

```

weeknames <- c("M", "T", "W", "T", "F", "S", "S")
title_1 <- paste0(year(x), "-", month(x))
title_2 <- paste0(year(x) + 1, "-", month(x))
# カレンダーでの表示
calendR::calendR(year(x), month(x),
  title = title_1, start = "M", weeknames = weeknames)

```

2024-5

M	T	W	T	F	S	S
		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		

```
calendR::calendR(year(x) + 1, month(x),
  title = title_2, start = "M", weeknames = weeknames )
```

2025-5

M	T	W	T	F	S	S
			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	

コード 4.49 (date-same-pos-next-yr-example.R) : 1 年後の日付への更新

```
sentence <- "大学祭は、2024年10月26日と10月27日に開催します。"
days_this_yr <- extract_date_ish(sentence)
days_next_yr <-
  days_this_yr |>
  date_ish2date() |>
  same_pos_next_yr(out_format = "west") |>
  rlang::set_names(days_this_yr) # 名前が置換前、値が置換後
sentence |>
  stringr::str_replace_all(days_next_yr)
## [1] "大学祭は、2025年10月25日(土)と2025年10月26日(日)に開催します。"
```

5 コマンドの実行

アプリを起動するとき、デスクトップやメニューにあるアイコンをマウスでクリックする人が多いと思います。そのとき、アイコンを見つけるのに時間がかかることはありませんか。深い場所のファイルを探すのに時間と労力を奪われることもあるでしょう。それぞれの時間は数十秒や数分でも、何度も繰り返しているとかなりの時間になります。しかし、少し準備すれば、2段階のキーボードの操作だけでアプリやファイルを瞬間起動できるようになります。本章では、このようなアプリの瞬間起動の方法について説明します。

次に、ファイルを関連付けしたアプリでRから開く方法、Rをバッチモードで実行する方法、RからOSのコマンドを実行する方法を説明します。それぞれの部分だけでは自動化作業に直接役立つわけではありませんが、自動化の脇役としていぶし銀のはたらきをします。たとえば、関連付けしたアプリでRからファイルを開ければ、生成したファイルを自動化作業のすぐ後に確認できます。Rのバッチモードでの実行とアプリの瞬間起動を組み合わせれば、自動化のスクリプトを一瞬で実行できます。また、RのコードとOSのコマンドの実行を組み合わせれば、自動化の幅は広がります。

本章の最後には、自動化の例としてzipファイルの解凍をRスクリプトで簡単に行う方法を紹介します。紹介するコードを参考にして、繰り返し作業の自動化のヒントにしてください。

なお、本章ではOSの機能を使った自動化作業が多くあります。そのため、OSによって作業の方法や自動化できることが異なります。基本的な操作は著者の主な使用環境のWindows11での方法です。ただし、可能な範囲でMac(Big Sir)とUbuntu/Desktop 22.04.3 LTSでの方法もご紹介します。

5.1 アプリケーションの瞬間起動

ここで紹介するように、キーボードの操作でアプリを瞬間的に起動できれば日々のパソコンの操作が楽になります。もちろんアイコンを探す必要がありません。Windowsなら[Win] + [R]、Macなら[Command] + [Space]、Ubuntuなら[Ctrl] + [Alt] + [T]のあとに、2-3文字を入力するだけでアプリを起動できます。ちょっと準備が必要ですが、毎日起動するアプリなら瞬間起動での時短効果は絶大です。

アプリの瞬間起動の準備としては、パスの通ったディレクトリにアプリなどのショートカットを保存する必要があります。よく使うアプリ(ワードやエクセル)や定期的に使うファイルあるいはディレクトリのショートカットをパスの通った場所に保存してください。ショートカットは作ったままの名前ではなく、短い名前に変更してください。2文字のショートカットを使うことをおすすめします。短いほうが少ない入力で起動できますが、アルファベット1文字では26個しか使えません。2文字だと入力に手間はかかりませんし、かなりの数のアプリを使えるからです。また、1文字だけよりもアプリの名称に近いものになるので覚えやすいです。

表 5.1: アプリと省略形の例

アプリ	省略形
GoogleChrome	gc

アプリ	省略形
R	r
RStudio	rs
Word	wd
Excel	ex
マイコンピュータ (MY computer)	my
コントロールパネル (Control Panel)	cp

以下ではパスについてとパスの通し方、またそのRのプログラムを説明します。パスの通ったディレクトリにアプリなどのショートカットの保存が自力ができる読者は、「瞬間起動の準備」はとばしてその次の「瞬間起動」に移動してください。

5.1.1 瞬間起動の準備

瞬間起動の準備として、パスを通った場所にショートカットを保存します。パスの通ったディレクトリとは、WindowsなどのOSが認識可能なディレクトリです。その場所にあるアプリやショートカットはファイル名だけで起動できます。一方、パスが通っていないフルパス ("c:/Users/USERNAME/FILENAME.txt"など)か、そのディレクトリを開いてファイル名を指定しないと起動できません。

Windowsでは、c:/winwows/system32に普通はパスが通っています。ですので、ここにショートカットを保存すると瞬間起動できますが、登録数が多くなると管理が面倒です。そこで著者はshortcutという専用ディレクトリを作成して、その中に保存しています。パスが通っていれば、shortcutの場どこでも構いません。たとえば、"c:/Users/USERNAME/shortcut"などです。

パスの通った場所はRからも確認できます。

コード 5.1 (command-get-path.R) : パスの通った場所の確認

```
Sys.getenv("PATH") |>
  stringr::str_split_1(";") |>
  head(3)
## [1] 'C:\\rtools43/usr/bin' 'C:\\WINDOWS' 'C:\\WINDOWS\\system32'
```

【注意】ショートカットをパスの通ったディレクトリに保存して、アプリの瞬間起動をすると非常に便利です。ただし、ショートカットと同じ名前をOSなどが使っている可能性があります。そのため、ショートカットの名前を決める前に確認が必要です。Windowsなら[Win] + [R]の「ファイル名を指定して実行」で、使いたいショートカットの名前(たとえば、hogehogeなど)を入力します。「hogehogeが見つかりません。名前を正しく入力したかどうか確認してから、やり直してください。」と表示されたら、その名前を新たなショートカットとしても問題ありません。MacやUbuntuならターミナルで、使いたいショートカットの名前を入力して「Command 'hogehoge' not found」のようなエラーメッセージがでることを確認してください。何かのアプリが起動す

るときやエラーの表示がないときは、その名前は避けましょう。エラーの表示がないときは、入力した名前でファイルを呼び出しているからです。

パスの通った場所が分かれば、その場所にショートカットを保存すると準備は完了します。新しいディレクトリを作ってパスを通すことも可能です。

パスを通すのが1回だけであれば手作業で設定すれば良いでしょう。手作業での方法は、ネットで「ユーザの環境変数 パスを通す」などと検索してください。ただし、自動化作業を進めていくときには、同じようにパスの設定をすることができます。その時のためにパスを通す関数(Windows用)を作ります。

コード 5.2 (command-add-path-fun.R) : パスを通す関数(Windows用)

```
add_path <- function(new_path){  
  path <- get_user_path()  
  path <- paste0(normalizePath(new_path), ";", path)  
  cmd <- paste0("setx path ", path) # パス設定のdosコマンド  
  res <- system(cmd, intern = TRUE) # コマンド実行  
  message(iconv(res, "sjis", "utf8")) # 文字化け対策  
  return(path)  
}  
get_user_path <- function(){  
  # レジストリエディタでパスを取得するコマンド  
  cmd <- 'reg query "HKEY_CURRENT_USER\\Environment" /v "path"'  
  path <-  
    system(cmd, intern = TRUE)[3] |> # コマンド実行  
    stringr::str_remove(" *path *REG_[A-z]* *") |> # 必要部分の取り出し  
    double_quote()  
  return(path)  
}  
double_quote <- function(x){  
  paste0('\"', x, '\"') # 文字列をダブルクオートで囲む  
}
```

add_path()でパスを通すときは、絶対パスを指定します。

コード 5.3 (command-add-path.R) : パスを通す疑似コード

```
add_path("c:/Users/USERNAME/shortcut") # 絶対パスで指定
```

《注意》 ディレクトリにパスを通すこと自体は難しくはありませんが、既存のパスを削除しないでください。誤って必要なパスを削除すると、システムが動かなくなる可能性があります。必要に応じて復元ポイントを作成するなどの対策をしてください。

いくつかのショートカット作成ぐらいなら手作業でもそれほど手間はかかりません。せっかくなので、以下で R からショートカットを作成するコード(Windows 用)を作ってみます。
C:/Windows/System32 にショートカットを作るのは管理者権限が必要なため、R からの操作は難しいようです。そこで、C:/Users/USERNAME に shortcut というディレクトリを作成して、そこにショートカットを保存するとともにパスを通します。

なお、ショートカットを作成する関数である make_shortcut() を次のコードで読み込みます。
make_shortcut() の内部では、shell() (5.4 節を参照) を活用しています。

コード 5.4 (command-make-shortcut.R) : ショートカットを作成する関数の読み込み

```
source("https://matutosi.github.io/r-auto/R/09__excel_funs.R")
```

【報告】 前回では、automater というパッケージを使ってショートカットを作るとしていましたが、automater パッケージの管理を怠っていたため、現在インストールできない状態です。そこで、別途読み込む方式に変更しました。パスを通す関数も本題とは深くは関係ないので、同じようにしても構いません。

コード 5.5 (make-shortcut.R) : R と RStudio のショートカットを作成してパスを通す

```
# RStudio のショートカット作成 パスが異なるときは適宜変更 exe <-
# fs::path_home('Appdata/Local/Programs/RStudio/rstudio.exe')
exe <- "C:/Program~1/rstudio/rstudio.exe" # こちらの可能性あり
shortcut <- "rs"
wd <- fs::path_home("shortcut")
size <- 3
make_shortcut(exe, shortcut = shortcut, size = size, wd = wd)

# R のショートカット作成 パスが異なるときは適宜変更
exe <- fs::path(Sys.getenv("R_HOME"), "bin/x64/Rgui.exe")
shortcut <- "r"
# --no-restore : 環境を復元しない、--no-save : 終了時に保存しない
# --sdi : SDI で起動、--silent : 起動時メッセージを出さない
arg <- "--no-restore --no-save --sdi --silent"
make_shortcut(exe, shortcut, arg = arg, size = size, wd = wd)

# C:/Users/USERNAME/shortcut にパスを通す
new_path <- add_path(wd)
```

《Tips》 Windows では文字コードとして、いわゆる ShiftJIS を使っています。R の文字コードは UTF-8 のため、system() の返り値に日本語を含むときは文字化けします。この文字化け対策として、iconv() で文字コードを変換しています。

5.1.2 瞬間起動

パスの通ったディレクトリにショートカットを保存できれば準備は完了です。Windowsでは、[Win] + [R]を押したあと、「ファイル名を指定して実行」でショートカットの名前を入力するとアプリが起動します。上記で登録したものであれば、"rs"を入力すればRStudio、"r"を入力すればRが起動します。



図 5.1: Ctrl + R で「ファイル名を指定して実行」からアプリを起動

Mac や Linux でもパスの通ったところにあれば、簡単にアプリを起動できます。Mac は[Command] + [Space]で Spotlight 検索が、Ubuntu なら[Ctrl] + [Alt] + [T]でターミナルが開きます。Spotlight 検索やターミナルにパスの通ったディレクトリにあるアプリなどの名前を入力すると起動できます。

5.2 関連付けアプリでの起動

R の作業中に、保存したファイルの確認のために関連付けアプリでファイルを開くことがあります。エクセルのワークブックに R でオートフィルタを設定(9.7 節を参照)しても、実際のエクセルの状態がわからないからです。このときに、エクスプローラでファイルをクリックして起動するのではなく、R から起動できれば便利です。

Windows では `shell.exec("DIR/Filename")` で関連付けのアプリで開くことができます。Mac や Ubuntu は、`system()`を使って関数を作ることができます。完全に同じではないかもしれません、関連付けしたアプリでファイルを開くことができます。

コード 5.6 (command-exec-mac-fun.R) : 関連付けアプリで開く関数

```
shell.exec <- function(file) {  
  cmd <- paste0("open ", file)  
  system(cmd)  
}
```

《Tips》 関数に適切な名前をつけると、その関数の動作を理解しやすくなります。関数は動詞のみあるいは動詞_名詞で名前をつけると良いことが多いです。たとえば、dplyr の `bind_rows()` は、bind が結びつける、rows が行という意味ですので、複数のデータフレームの行を結びつける動作と関数の名前とが合致しています。

《Tips》 プログラミングでの関数や変数の命名規則にはいくつかの方法があります。`function_name` のようなスネークケースや `functionName` のようなキャメルケースが使われ、Rではスネークケースが多いです。本書ではスネークケースでの命名を基本としていますが、ここではWindowsの関数名と合わせるために `shell.exec()` としました。

コード 5.7 (command-exec-xlsx.R) : ワークブックをエクセルで開く

```
path <- fs::path_temp("iris.xlsx")
wb <- openxlsx::write.xlsx(iris, path)
openxlsx::addFilter(wb, sheet = 1, rows = 1, cols = 1:5)
openxlsx::saveWorkbook(wb, path, overwrite = TRUE)
shell.exec(path)
```

関連付けていれば、エクセル以外のファイルも開けます。URLを指定すれば既定のブラウザで立ち上がります。本書のサポートページを開いてみましょう。

コード 5.8 (command-exec-url.R) : URLからブラウザを起動

```
url <- "https://github.com/matutosi/r-auto"
shell.exec(url)
```

ディレクトリを指定すれば、エクスプローラやファインダーなどのファイルが起動します。

コード 5.9 (command-exec-directory.R) : ホームディレクトリを開く

```
home <- fs::path_home()
shell.exec(home)
```

5.3 Rのバッチモードでの実行

【大きな変更】`Rscript`との関連付けをバッチモードでの説明に変更しました。バッチモードの方が、読者の作業と紙面も少なくてすみ、説明も楽なためです。

プログラムを書いている途中では、RやRStudioのコンソール画面で対話的にプログラムを実行することが多いでしょう。一方、プログラムの内容を変更することができないときに、プログラムのコードをエディタやRStudioで開いてから実行するのは手間がかかります。Rのバッチモードを使えば、プログラムのファイルを開かずに実行できて便利です。たとえば次は乱数の散布図をPDFとして保存する見本のプログラムです。

コード 5.10 (command-r-script-plot.R) : 亂数の散布図を PDF で保存

```

path_pdf <- fs::path_home("desktop/plot.pdf")
pdf(path_pdf) # pdf デバイスを開く
  plot(rnorm(100), rnorm(100)) # 散布図の描画
dev.off() # デバイスを閉じる
shell.exec(path_pdf) # PDF ファイルを開く

```

Windows でこのプログラムを `c:/Users/USERNAME/RR(%homepath%/RR)` に `plot.R` として保存しているとします。R をバッチモードで実行するには、コマンドプロンプトで以下のようにします。

コード 5.11 : R をバッチモードで保存するコマンド(Windows)

```

cd c:/program files/R/R-4.3.3/bin/x64
R CMD BATCH --silent --vanilla %homepath%/RR/plot.R %homepath%/RR/plot.log

```

実行すると、デスクトップに散布図の PDF ファイルが保存されます(図 5.2)。

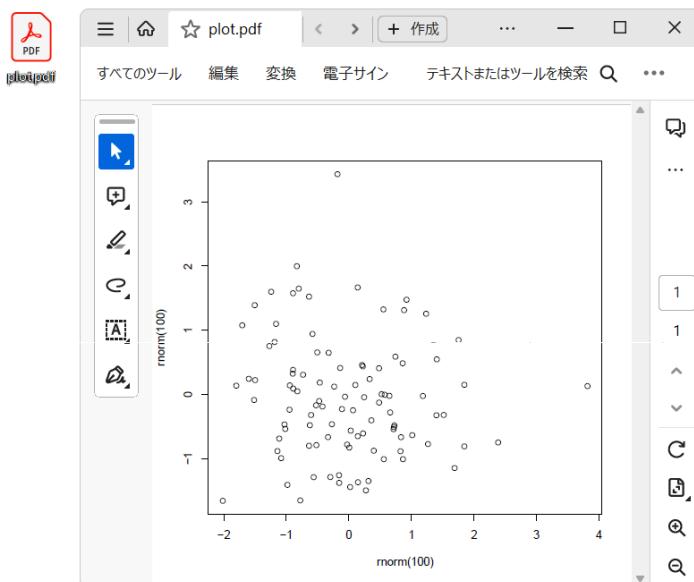


図 5.2: バッチモードで作成した散布図の PDF

1 行目の `cd` でディレクトリを変更します。R のインストール先やバージョンに合わせて変更してください。なお、`R.exe` のディレクトリにパスが通っていれば、ディレクトリの変更は不要です。

2 行目の `R CMD BATCH` で R をバッチモードで起動します。`--silent` と `--vanilla` の 2 つの起動オプションを指定します。`--silent` は起動時のメッセージ(R version 4.3.3 (2024-02-29 ucrt)などの数行)を表示させないもので、`--vanilla` はまっさらな状態で R を起動するものです。`%homepath%` はユーザのホームディレクトリで、実行時に `c:/Users/USERNAME` に変換されます。もちろん、`c:/Users/USERNAME` のように直接書いても構いません。

指定したプログラムを実行して、ログをファイルに保存します。うまく実行できていれば、`plot.log` には次のような実行結果が記録されているはずです。なお、`proc.time()` は自動で実行されているもので、実行時間を表示しています。

うまくいかない場合は、ログファイル(plot.log)を確認してください。ログファイルでエラーが表示されているときは、plot.Rの内容を確認し、内容に応じた対処が必要です。よくあるのは、必要なパッケージのインストールなどです。

コード 5.12：ログファイルの内容

```
> path_pdf <- fs::path_home("desktop/plot.pdf")
> pdf(path_pdf)                      # pdf デバイスを開く
> plot(rnorm(100), rnorm(100))        # 散布図の描画
> dev.off()                          # デバイスを閉じる
null device
      1
shell.exec(path_pdf)                 # PDF ファイルを開く
>
> proc.time()
  user  system elapsed
 0.32   0.04   0.43
```

Mac や Linux では、プログラムを/usr/USERNAME/RR(~/RR)に plot.R として保存しているとします。R をバッチモードで実行するには、ターミナルで以下のようにします。なお、R のインストール先のディレクトリが異なる時は、変更してください。

コード 5.13：R をバッチモードで保存するコマンド(Mac と Linux)

```
#!/bin/bash          # shebang
cd /usr/local/bin/  # ディレクトリの変更
R CMD BATCH \         # R をバッチモードで起動
--silent --vanilla \ # 起動オプション
~/RR/plot.R \         # プログラムのファイル
~/RR/plot.log        # ログを保存するファイル
```

shebang はシバンやシェバンとよばれるもので、/bin/bash というシェルで実行するという意味です。ターミナルでの実行時にはシバンは不要ですが、後で使うシェルスクリプトでは必要なのでここにも記載しています。その他の部分は、Windows での説明を参考にしてください。

コード 5.11 や 5.13 をコマンドプロンプト等で実行するのは面倒です。そこで、バッチファイルやシェルスクリプトから実行できるようにします。

Windows では、コード 5.11 を plot.bat というファイルとして保存します。Mac や Linux では、コード 5.13 を plot.sh というファイルとして保存します。

plot.bat や plot.sh の保存先がパスのとおったディレクトリであれば、瞬間起動が使えます(5.1 節を参照)。また、ショートカットやエイリアスをパスのとおったディレクトリや Spotlight 検索の対象ディレクトリに保存しても構いません。なお、Mac や Linux で実行できないときは、

`plot.sh` に実行権限がないことが考えられます。 ターミナルから `plot.sh` のあるディレクトリで以下を実行して、実行権限を付与してください。

```
chmod +x plot.sh
```

なお、Mac で Spotlight 検索から起動する場合は、`/usr/local/bin` などの検索対象のディレクトリに `plot.sh` かエイリアスを保存します。 また、シェルスクリプトの実行後にウィンドウを閉じるには、ターミナルの環境設定の[プロファイル]-[シェル]で[シェルの終了後]を「シェルが正常に終了した場合は閉じる」にします。

内容を変更しないプログラムで、何度も実行するものであれば、R のバッチモードでの実行と瞬間起動を組み合わせると自動化の利点を最大限に活かせます。

5.4 OS コマンドの実行

本書では、R から様々な形式のファイルの操作方法を紹介しています。 それでも説明できていないものも多くあります。 その操作を R から実行するには 2 つの方法があります。 1 つ目は対象の操作をするパッケージを探す方法で、2 つ目は `system()` などの R の関数を使う方法です。

パッケージを探すときは、CRAN や GitHub で検索すれば良いでしょう。`pdf` に変換する方法を探す場合は、「CRAN 変換 pdf」あるいは「CRAN convert pdf」のように検索します。 可能であれば、英語で検索するとパッケージの見つかる可能性が高まります。 なお、CRAN のパッケージ一覧を取得する方法は、15.6 節を参照してください。

2 つ目の `system()` などの関数はいわゆるシェルで命令を実行します。 シェルとは、Windows ならコマンドプロンプト(昔でいう dos 窓)や PowerShell、Mac や Ubuntu ならターミナルのことです。 アプリによりますが、シェルから命令を実行できるものは結構多くあります。

`system()` では、OS のコマンドを使うことができます。そのため、Windows のバッチファイルや `dos` コマンド、あるいは Mac や Linux のシェルスクリプトをもとに自動化できます。 その際には、OS のコマンドと R のいいとこ取りをすると良いでしょう。 つまり、バッチファイルやシェルスクリプトでの扱い方が難しいところは R のコマンドで実行し、簡単なコマンドで処理できる部分はバッチファイルやシェルスクリプトで処理します。

《Tips》 Windows では `shell()` という関数も使うことができます。`system()` と使い方はほぼ同じです。 同じコマンドでも実行した結果が異なることがあります。 つまり、`system()` ではうまく実行できなくても、`shell()` だとうまくいくことがあります。

拡張子のあるファイルは、アプリと関連付けして `shell.exec()` で起動可能です(5.2 節を参照)。一方、拡張子のないファイルは、Windows では関連付けができないので `shell.exec()` は使えません。 ただし、一般的なアプリでは実行ファイル名の次にファイルを指定して開くことができます。

たとえば、RStudio の最近のプロジェクト一覧(メニューの[File] - [Recent Projects])を編集したいとします。 [Clear Project list] ですべて削除することはできますが、一部だけの編集は RStudio からはできません。 一部を編集するには、一覧の入っているファイルである

`project_mru` を編集します。このファイルは拡張子がありませんが、テキストファイルなので、テキストエディタで編集可能です。そこで、次のコードのようにテキストエディタの実行ファイルと開く対象のファイルの名前を組み合わせたコマンドを作成して `system()` で実行します。著者は秀丸エディタを愛用していますが、`bin` は読者の方が使っているエディタに変更してください。

コード 5.14 (`command-system-rstudio-menu.R`) : 拡張子のないファイルをアプリを指定して起動

```
# 半角スペースがあるので文字列として「」を含める
# bin <- '"c:/Program..."' のシングルクオーテーション(')は
# ダブルクオーテーション(")と区別するため
bin <- '"c:/Program Files/hidemaru/hidemaru.exe"'
file <- fs::path(Sys.getenv("LOCALAPPDATA"),
                 "RStudio/monitored/lists/project_mru")
cmd <- paste0(c(bin, file), collapse = " ")
system(cmd, wait = FALSE)
## [1] 0
```

`system()` でコマンドの実行が成功すれば、`0` が返ってきます。

また、次のようにファイルを秀丸エディタで開く関数を定義することも可能です。

コード 5.15 (`command-hidemaru-fun.R`) : 秀丸エディタでファイルを開く関数

```
open_with_hidemaru <- function(file){
  bin <- '"c:/Program Files/hidemaru/hidemaru.exe"'
  cmd <- paste0(c(bin, file), collapse = " ")
  res <- system(cmd, wait = FALSE)
  return(res)
}
```

《Tips》 R では文字列を指定するときには、ダブルクオーテーション(")を使うことが多いですが、シングルクオーテーション(')を使うこともできます。文字列の中でダブルクオーテーションを使うときには、\"のようにエスケープすることができますが、文字列の範囲が分かりづらくなります。上のコードの 1 行目(`bin <- '"c:/Program...."`)のように、文字列の指定にシングルクオーテーションを使えば、その中ではダブルクオーテーションをそのまま使うことができます。

コード 5.16 (`command-quotation-identical.R`) : 文字列の同一性確認

```
# どちらも同じ文字列になる
double_quo <- "\c:/Program Files/hidemaru/hidemaru.exe\""
single_quo <- '"c:/Program Files/hidemaru/hidemaru.exe"'
identical(double_quo, single_quo) # 同一性の確認
## [1] TRUE
```

コード 5.14 を実行するとテキストエディタが起動しますので、内容を編集してから閉じます。既定値である `wait = TRUE` のときは、エディタが閉じるまでは R(あるいは RStudio)は待機しています。 `wait = FALSE` とすると、エディタを閉じなくとも R は次のコマンドを受け付けることができます。ただし、`intern = TRUE` を指定したときは、`wait = FALSE` は無視されるため `system()` で起動したアプリが終了するまで待機します。

《Tips》 ディレクトリやファイルの名前には、半角スペースを使わない方が良いです。プログラムや OS のコマンドでは、スペースが区切りとして認識されて、誤動作の原因になるためです。ただし、`Program Files` のようにもともと半角スペースを含むときは、"Program Files" のようにダブルクオーテーションで囲まなければなりません。

5.5 アプリの終了

R から他のプログラムを起動したあとで、アプリを閉じるときにマウスによる操作でボタンをクリックするのはちょっと悔しい気持ちになります。せっかくならこれも R から操作したいものです。Windows では `taskkill` で終了できます。終了したいものを /pid(プロセス ID) か /im(イメージ名 = ファイル名) で指定します。プロセス ID は実行するたびに変わるので、アプリのファイル名を指定して終了するのが良いでしょう。

次のコードでは、描画した散布図を関連付けアプリで起動したあとに、`Sys.sleep()` で 3 秒待ってから、`system()` を使って終了させます。`intern = TRUE` で実行した結果を R で受け取って表示します。

コード 5.17 (command-taskkill-win.R) : アプリの終了コマンド

```
fs::path(fs::path_home(), "plot.pdf")
pdf(path_pdf)
plot(rnorm(100), rnorm(100))
dev.off()
shell.exec(path_pdf) # pdf を開く
Sys.sleep(3) # 3 秒待つ
cmd <- "taskkill /im Acrobat.exe" # コマンド
(res <- system(cmd, intern = TRUE)) # 実行
iconv(res, "sjis", "utf8") # 文字コード変換
```

5.6 zip ファイルの解凍

`zip` ファイルを右クリックして「すべてを展開」しても、十秒以内で作業は終わるかもしれません。しかし、何度も `zip` ファイルを解凍すれば、長期的には R での自動化の効果があります。そこで本節では、`zip` ファイルの解凍を自動化する方法を紹介します。スクリプトの作成・バッチファイルからの実行・瞬間起動の 3 つの段階が必要ですが、バッチモードでの実行は 5.3 節を、瞬間起動は 5.1 節をそれぞれ参照してください。最初にパスワードなしの `zip` ファイルの解凍方

法を、次にパスワード付きの解凍方法を説明します。なお、スクリプトの自動化によって大量の zip ファイルを簡単に解凍できますが、解凍後のファイルの安全性には十分気をつけてください。

5.6.1 パスワードなしの解凍

ここでは、複数の zip ファイルの解凍を想定していますので、解凍したファイル名が重複する可能性があります。そこで、zip ファイルのファイル名をもとにディレクトリを生成して、その中に解凍したファイルを保存します。その関数を `unzip_with_dir()` として定義しています。fs パッケージの関数の詳細は第 1 章やヘルプを参照してください。`unzip()` では `exdir = unzip_dir` として解凍先を指定しています。最後に、解凍先のディレクトリを `return(unzip_dir)` として返します。

関数の外では、デスクトップのディレクトリを変数 `dsk` に入れて、zip ファイルの一覧を取得しています。その後、purrr パッケージの関数 `map()` を使って、複数の zip ファイル解凍と複数のディレクトリ起動をします。

コード 5.18 (command-unzip-fun.R) : zip ファイルを解凍する関数

```
unzip_with_dir <- function(zip){  
  dir <- fs::path_dir(zip)                      # ディレクトリ  
  unzip_dir <- fs::path_file(zip)                # ファイル名  
  unzip_dir <- fs::path_ext_remove(unzip_dir)    # 拡張子除去  
  unzip_dir <- fs::path(dir, unzip_dir)          # 解凍先ディレクトリ  
  fs::dir_create(unzip_dir)                      # ディレクトリ生成  
  utils::unzip(zip, exdir = unzip_dir)           # 解凍  
  return(unzip_dir)  
}
```

デスクトップにある zip ファイルをすべて解凍するには、次のようにします。

コード 5.19 (command-unzip.R) : zip ファイルの解凍

```
dir_usr <- Sys.getenv("USERPROFILE")      # "c:/Users/USERNAME"  
dsk <- fs::path(dir_usr, "Desktop")       # デスクトップのディレクトリ  
zips <- fs::dir_ls(dsk, regexp = "\\.zip") # zip ファイル一覧  
dirs <- purrr::map(zips, unzip_with_dir)   # 解凍  
purrr::map(dirs, shell.exec)               # ディレクトリを開く
```

zip ファイル解凍のスクリプト(コード 5.19)を `UnZip.R` の名前で適切な場所に保存します。また、5.3 節を参考にして、`UnZip.bat` を保存してください。`UnZip.bat` へのショートカット(uz など)をパスの通ったディレクトリに保存します(5.1.1 節を参照)。その後は [Win] + [R] のあとに [uz] を入力して起動します(5.1 節を参照)。これでデスクトップに保存した zip ファイルをいつでもディレクトリ付きで簡単に解凍できます。ブラウザでのダウンロードファイルの保存先をデスクトップにするなど、ダウンロード先と上記のコードの zip ファイルの検索場所をあわせておくと便利です。

《応用》 zip ファイルのディレクトリをマウスで指定するには、 `tcltk::tk_choose.dir()` を使います(1.8 節を参照)。

《応用》 解凍ファイルを 1 つのディレクトリにまとめるには、解凍後にファイル名の変更と移動をします。ファイル名の変更は、同名ファイルの上書きを防ぐためです。`abc.zip` に `hoge.txt` が含まれるときは、`stringr::str_c()` を使って `abc_hoge.txt` などとすれば良いでしょう(3.2 節を参照)。

5.6.2 パスワード付きの zip ファイルの解凍

電子メールで圧縮ファイルをパスワード付きで送られることがあります。そのとき、「すべてを展開」してからパスワードを貼り付けるのは、単に解凍するよりもさらに手間がかかります。そんなときのために、クリップボードにコピーしたパスワードを使ってデスクトップのパスワード付き zip ファイルを解凍する関数を定義します。

基本的な内容はパスワードなしの解凍(コード 5.19)と同じですが、パスワードに対応するために解凍ソフトとして 7zip を使います。7zip のインストール方法は "7zip インストール" などで検索してください。7zip をインストールしたディレクトリを `bin_path` で設定する必要があります。複数の zip ファイルでも、すべて同じパスワードであれば、1 回の動作で解凍可能です。

【補足】以下の説明を入れようとしたのですが、パスワード付きの解凍では 7zip 以外では良さそうなものがなかったので、7zip のみの記述にしました。ただし、念のため「もし、…」の説明を入れています。

もし、他の解凍ソフトを使う場合は、`system(cmd)` で実行するコマンドの文字列(`cmd`)を変更する必要があります。`cmd <- paste0()` の部分を解凍ソフトのヘルプなどを参考に変更してください。

コード 5.20 (command-unzip-pass-fun.R) : パスワード付きの zip ファイルを解凍する関数

```
unzip_with_password <- function(zip, passwd = "", bin_path = ""){
  dir <- fs::path_dir(zip)
  unzip_dir <-
    fs::path_file(zip) |>
    fs::path_ext_remove()                                # 拡張子除去
  unzip_dir <-
    fs::path(dir, unzip_dir) |> #
    stringr::str_replace_all(" ", "_") |> # スペースを置換
    fs::dir_create()                                    # ディレクトリ作成
  zip <- paste0("7z x", zip, " -p")
  if(passwd == ""){
    passwd <- read.table("clipboard")[1,] # クリップボードから
  }
  cmd <- paste0(bin_path, "7z x", zip, " -p", passwd, " -o", unzip_dir)
  system(cmd)
```

```
    return(unzip_dir)
}
```

デスクトップにあるパスワード付きの zip ファイルを解凍するには、以下のようにします。ただし、関数の実行前にパスワードをコピーしてクリップボードに保存しておきます。

《Tips》 if と同じ行に TRUE の場合のコードを書く場合、{}を使わなくても大丈夫です。つまり、以下の 2 行とも同じです。ただし、範囲を明確にするために、上のように{}で囲うことがあります。

```
if(passwd == ""){ passwd <- read.table("clipboard")[1,] }
if(passwd == "")  passwd <- read.table("clipboard")[1,]
```

コード 5.21 (command-unzip-pass.R) : パスワード付きの zip ファイルの解凍

```
zips <-
  fs::path_home("Desktop") |>
  fs::dir_ls(regexp = "\\.zip")
bin_path <- "c:/DIRECTORY/7zip/" # 要修正
dirs <- zips |>
  purrr::map(unzip_with_password, pass = "", bin_path = bin_path) # 解凍
purrr::map(dirs, shell.exec) # ディレクトリを開く
```

《応用》 パスワードが zip ファイルごとに異なるときは、関数の外で read.table("clipboard") とすることでパスワードを複数取得することで対応可能です。たとえば、クリップボードに以下のパスワードが入っているとします。

```
aaa
bbb
ccc
```

このとき、以下のコードで passwd にクリップボードの内容を passwd に代入できます。

コード 5.22 (command-clipboard.R) : クリップボードの取り出し

```
passwd <- read.table("clipboard")[[1]] |>
  as.list()
```

passwd に代入したパスワードは [[i]](i は序数)で取り出せます。そのため purrr パッケージの map2() を使って、 purrr::map2(zips, unzip_with_password, passwd) として、 zip ファイルとパスワードの両方を引数にできます(2.8 節を参照)。ただし、ファイル名の順番とパスワードの順番を一致させる必要があります。

6 キーボードとマウスの操作

毎朝、職場でパソコンの電源ボタンをしてから仕事に取りかかるまでの間に、各種アプリの通知ボタンを1つずつクリックするモグラたたき的な操作をしていないでしょうか。また、メニューバーのタスクトレイのアプリのアイコンの右クリック・出たメニューのクリック・パスワードの入力・「OK」ボタンのクリックなどの作業があるかもしれません。パソコン起動時に毎回同じ操作をするのは本当に手間だし、気分が下がります。そんなことに時間を使うよりも、豆から挽いたコーヒーをじっくり淹れて仕事に集中する雰囲気をつくりたいものです。

パソコン起動時にリモートデスクトップ接続の操作とグループウェアからの通知を開く作業を著者は手作業でしていました。このとき、Windowsの内部では何らかのプログラムが動いているので、C言語などを駆使すれば操作できるはずです。ただし、使用者の少ないアプリではパッケージやAPIが整備されていません。自分でパッケージを作れば良いのでしょうか、それには膨大な知識と手間が必要です。

そんなときは、手作業で実行しているキーボードやマウスの動きをそのまま自動化して実行しましよう。1回きりのことであれば、手作業しかありませんが、定型的な作業は自動化できます。定型的なキーボードやマウスによる操作の自動化はKeyboardSimulatorパッケージで実現できます。自動化と顔認証でのログインを使えばキーボードとマウスに触れなくても、仕事をするための環境が整います。誤操作によるやり直しもありません。

【注意】 KeyboardSimulatorはWindowsのみで利用可能です。

6.1 キーボードとマウスを操作するパッケージ

KeyboardSimulatorパッケージのインストールと呼び出し方法は以下のとおりです。

コード 6.1 (kybd-install.R) : KeyboardSimulator のインストール

```
install.packages("KeyboardSimulator")
```

コード 6.2 (kybd-library.R) : KeyboardSimulator の呼び出し

```
library(KeyboardSimulator)
```

6.2 キーボードの操作・文字入力

キーボードの操作や文字の入力には、`keybd.press()`と`keybd.type_string()`を使います。たとえば、キーボードの操作でアプリの画面を左半分に配置するため、[Win] + (カーソルの)[左矢印]

を押すには、次のようにします。複数のキーボードを押すには+を使います。次のコードを RStudio で実行すると、 RStudio の画面が左側に移動するはずです。

コード 6.3 (kybd-keybd-press.R) : キーボードの操作

```
keybd.press("win+left") # アプリを左側に
```

keybd.press()でキーボードを押し続ける操作をしたければ、 hold = TRUE と指定し、 キーボードを離すには keybd.release()を使います。

文字の入力を自動化するには、 keybdtype_string()を使います。 "abc"と入力するには keybdtype_string("abc")とします。 keybd.press("a+b+c")としても結果は同じですが、 keybdtype_string()のほうが分かりやすいです。 次のコードを実行すると、 RStudio のコンソールに「abc」が表示されます。

コード 6.4 (kybd-keybd-type.R) : キーボードからの文字入力

```
keybd.type_string("abc") # abc を入力  
# keybd.press("a+b+c") # 上と同じ
```

KeyboardSimulator では、英語キーボードの配列で設定されているので、日本語配列のキーボードの場合は、+などの記号は別の文字列が入力されてしまいます。一般的な日本語キーボードでの入力と出力が異なるものは、表 6.1 を参考にしてください。

表 6.1: 「日本語キーボード 106/109」での入力と出力

入力	出力
"	~
&	'
'	^
=	;
~	、
,	@
*	(
-	=
()
)	出力無し
^	&
\`	\
@	"

入力 出力

```
; :  
: *
```

また, `keybd.type_string("あ")`のように日本語を直接入力することはできません。ローマ字変換の日本語入力モードになっていれば, `keybd.type_string("a")`とすると, "あ"を入力できます。

なお, `KeyboardSimulator`で使えるキーボードの値の一覧は, `keyboard_value`で表示できます。行数が多いので, 以下では 85-90 行目を表示しています。

コード 6.5 (kybd-keyboard-value.R) : キーボードの値一覧

```
dplyr::slice(keyboard_value, 85:90)  
##      button virt_code scan_code prefix_byte shift  
## 1 backspace     8       14    FALSE FALSE  
## 2 tab           9       15    FALSE FALSE  
## 3 enter         13      28    FALSE FALSE  
## 4 shift          16      42    FALSE FALSE  
## 5 ctrl           17      29    FALSE FALSE  
## 6 alt            18      56    FALSE FALSE
```

`KeyboardSimulator`を使って, アプリの瞬間起動(5.1 参照)と同じ操作をしてみます。キーボードから操作するには, [Win] + [R]を押して「ファイル名を指定して実行」に"cmd"と入力して Enter キーを押します。これでコマンドプロンプトが起動できます。以下のコードは, これと同じ操作をするものです。

著者の環境では, RStudio にコードを貼り付けたときとバッチモード(5.3 参照)では動作しました。ただし, R の対話的環境ではうまく動作しませんでした。“cmd”がRのコンソール画面に入力されてしまうためです。キーボード操作をRの対話的環境で使うことはほとんどなく, バッチモードで使うことがほとんどでしょうから, あまり問題はないと思います。

コード 6.6 (kybd-shell-instant.R) : コマンドプロンプトの起動

```
KeyboardSimulator::keybd.press("win+r")  
Sys.sleep(0.5) # エラーの場合は長くする  
KeyboardSimulator::keybd.type_string("cmd")  
KeyboardSimulator::keybd.press("enter")
```

《Tips》 操作内容によりますが, `Sys.sleep()`で待機しないと Windows が R の命令に追いつかず, エラーになります。著者の使用環境では, 0.3 秒以上の待機が必要でした。

6.3 マウスの位置取得

当然ですが、マウスによる操作で重要なのはクリックする位置の正確な指定です。画面上での定位置をクリックするのであれば、KeyboardSimulator の関数を使って位置を取得しておきます。定位置でなく画面上の画像を認識して位置を取得する方法は、6.6 節で説明します。

KeyboardSimulator には、画面上のマウスの位置を取得する関数に `mouse.get_cursor()` があります。

コード 6.7 (kybd-mouse-get-cursor.R) : マウス位置の取得

```
mouse.get_cursor()  
## [1] 297 306
```

1箇所だけならこの関数で良いのですが、何度かマウスをクリックする作業の場合はちょっと面倒です。たとえば、USB メモリの安全な取り外しのときには、何度か位置を取得する必要があり、タスクバーと R を往復することになります。ウィンドウを往復していると、タスクバーの操作が元の状態に戻り、正しいマウス位置の取得が難しくなります。そこで、マウス位置を複数回取得する関数を定義します。既定値では、キーボードで何か入力すると、その時のマウスの位置を取得します。引数 `interval` に正の数を指定すればその秒数ごとに記録します。

コード 6.8 (kybd-mouse-record-fun.R) : マウス位置の取得する関数

```
mouse_record <- function(n = 3, interval = -1){  
  pos <- list()  
  for(i in seq(n)){  
    if(interval < 0){  
      readline("Press Enter on R console") # クリックごと  
    }else{  
      Sys.sleep(interval) # 一定時間ごと  
    }  
    pos[[i]] <- KeyboardSimulator::mouse.get_cursor()  
    position <- paste0(i, ": x = ", pos[[i]][1], ", y = ", pos[[i]][2], "\n")  
    cat(position)  
  }  
  return(invisible(pos))  
}
```

関数を実行すると、マウスでクリックした位置の座標を取得できます。

コード 6.9 (kybd-mouse-record.R) : マウス位置の取得

```
mouse_record()  
## Press Enter on R console
```

```
## 1: x = 568, y = 143
## Press Enter on R console
## 2: x = 334, y = 564
## Press Enter on R console
## 3: x = 602, y = 484
```

6.4 マウスの移動

マウスの位置移動には、`mouse.move()`を使います。引数 `x` と `y` はそれぞれ移動先の `x` 軸と `y` 軸の座標の値です。なお、ディスプレイは左上が `xy` 座標の原点、横が `x` 軸で縦が `y` 軸です。ただし、`y` 軸は下向きに正の値ですので、ディスプレイの右下が最大値です。

コードを実行すると移動の様子を実際にみることができます。次のコードでは、マウスが右から左に徐々に動きます。`duration` では移動にかかる時間を秒単位で指定するので、この数値が大きいほどゆっくりした動きになります。`duration` を指定しないと、瞬間移動します。`step_ratio` では移動段階をどのように分けるかを指定し、この数値が小さいほど滑らかな動きになります。

コード 6.10 (kybd-mouse-move.R) : マウスの位置移動

```
mouse.move(x = 400, y = 200)
mouse.move(200, 200, duration = 1, step_ratio = 0.1)
```

6.5 マウスのクリック

マウスのクリックには`mouse.click()`を使います。`button = "right"`で右クリック、`hold = TRUE`でクリックしたままになります。クリックを解除するには、`mouse.release()`を使います。

コード 6.11 (kybd-mouse-click.R) : マウスのクリック

```
mouse.click(button = "left", hold = FALSE) # 既定値のクリック
```

マウスの移動とクリックは組み合わせて使うことが多いので、移動とクリックを1つにした関数を定義します。さらに、クリック後に待機するように、`sleep_sec` の引数を追加します(既定値は、0.1秒)。

コード 6.12 (kybd-mouse-move-click-fun.R) : マウスを移動してクリックする関数

```
mouse_move_click <- function(x, y, button = "left", hold = FALSE,
                           sleep_sec = 0.1){
  KeyboardSimulator::mouse.move(x, y)
  KeyboardSimulator::mouse.click(button = button, hold = hold)}
```

```
Sys.sleep(sleep_sec)
}
```

ディスプレイの原点に近い左上付近(たとえば、x 座標, y 座標とも 50 の位置)にあるファイルのアイコンをクリックしたい場合は、次のようにします。

コード 6.13 (kybd-mouse-move-click.R) : 左上のファイルをダブルクリック

```
mouse_move_click(50,50)
mouse_move_click(50,50)
```

先程の位置のアイコンやファイルをドラッグして少し右(xy とも 150 の位置)に移動するには、次のようにします。なお, sleep_sec = 0 とすると、エラーになる可能性があります。

コード 6.14 (kybd-mouse-move-click-hold.R) : 左上のファイルをドラッグして移動

```
mouse_move_click(50,50, hold = TRUE, sleep_sec = 0.1)
mouse_move_click(150,50)
mouse.release()
```

6.6 画像の位置特定によるマウスの移動・クリック

マウスでクリックするアイコンやウィンドウの位置が決まっていないときには、画像をもとにクリックする位置を特定します。ただ、KeyboardSimulator には、画像の位置特定の機能はありません。そこで、画面のスクリーンショットを取得して、画像の位置を特定する関数を含む screenshot パッケージを作成しました。screenshot は CRAN に登録していますので、install.packages() でインストール可能です。

コード 6.15 (kybd-screenshot-install.R) : screenshot のインストール

```
install.packages("screenshot")
```

コード 6.16 (kybd-screenshot-library.R) : screenshot の呼び出し

```
library(screenshot)
```

画像位置の特定は、locate_image() で実行します。Windows では事前にスクリーンショット撮影のバッチファイルを install_screenshot() でインストールする必要があります。

install_screenshot() で引数を指定しないと、screenshot パッケージのディレクトリにインストールされます。他のディレクトリにインストールしたいときは、bin_dir = "インストール先の

ディレクトリ"と指定します。ただし、省略するのをおすすめします。省略しておくと、スクリーンショットを撮影する関数の `screenshot()` でもインストール先の引数を省略できるからです。

コード 6.17 (kybd-screenshot-install-screenshot.R) : スクリーンショット撮影のバッチファイルのインストール

```
# fs::path_package("screenshot")にインストールされる
screenshot::install_screenshot()
```

コード 6.18 (kybd-screenshot.R) : スクリーンショット撮影

```
sc <- screenshot::screenshot()
## C:/Users/USERNAME/AppData/Local/Temp/RtmpkP1iLf/sc_287c4b4873da.png
imager::load.image(sc) # imager で読み込み
|> plot()             # 表示
# shell.exec(sc) # 関連付けアプリで開く場合
```

`locate_image()` を使う前に、位置を特定する画像を準備しなければなりません。画像の左上が一番分かりやすいので、`magick` パッケージの関数で、スクリーンショットの左上から縦横とも 60 ピクセルの画像を切り出し、`png` 形式で書き込みます(第 11 章を参照)。

コード 6.19 (kybd-screenshot-needle-image.R) : 位置特定用の画像の準備

```
needle_image <-
  magick::image_read(sc) |>
  magick::image_crop(geometry = "60x60+0+0")
plot(needle_image)
path_needle <- fs::file_temp(ext = "png")
magick::image_write(needle_image, path_needle)
# shell.exec(path_needle)
```

特定する関数と位置特定用の画像が揃いました。`locate_image()` の引数 `needle_image` では、位置を特定する画像ファイルを指定します。`center = TRUE`(既定値)では、特定した画像の中心の位置を返しますが、`center = FALSE` とすると左上の位置を返します。

コード 6.20 (kybd-screenshot-locate-image.R) : 画像の位置特定によるマウスの移動・クリック

```
screenshot::locate_image(needle_image = path_needle)
screenshot::locate_image(path_needle, center = FALSE)
```

【注意】 万が一、`screenshot::locate_image()` の動作にバグがあれば、著者までご連絡ください。

6.7 USB メモリの取り出しの自動化

ここでは、USB メモリの安全な取り出しを自動化します。この作業をしなくても、USB メモリが壊れることはほぼ無いですが、するに越したことはありません。単純で面倒なことは自動化しましょう。

タスクバーでの位置が固定されているのであれば、`mouse_record()`でクリックするべき位置を取得します。今回の場合は、図 6.1-6.4 の 4 つの画像の位置です。

これらの画像の位置を `mouse_record()` で特定します。なお、クリックの位置特定で使う画像は、クリックする位置が中央になるように切り取ってください。切り取りを簡単にする方法は、11.16 節を参照してください。

コード 6.21 (kybd-remove-usb-pos.R) : USB 取り出し用マウス位置の取得

```
pos <- mouse_record(n = 4)
## Press Enter on R console pos # xy 座標の位置は環境によって全く異なる
## 1: x = 1050, y = 671
## Press Enter on R console
## 2: x = 1055, y = 652
## Press Enter on R console
## 3: x = 1179, y = 695
## Press Enter on R console
## 4: x = 1021, y = 677
```

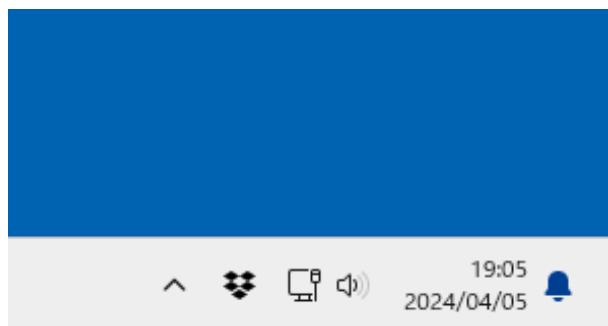


図 6.1: 上向き矢印をクリック

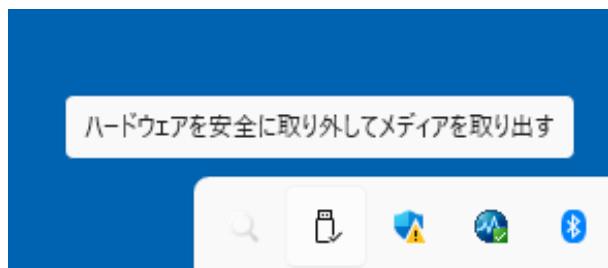


図 6.2: USB のアイコンをクリック

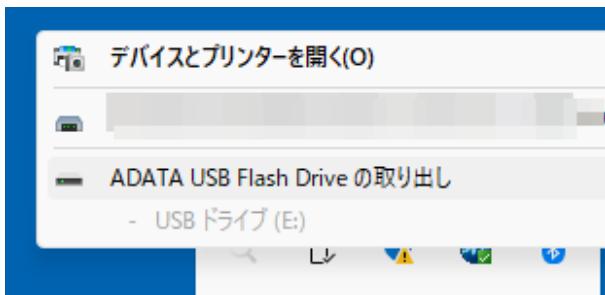


図 6.3: USB の名前をクリック

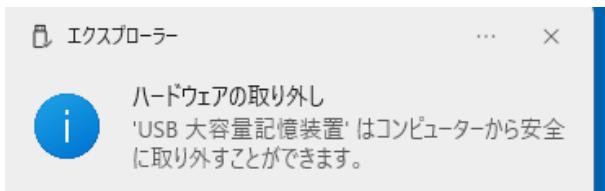


図 6.4: 取り出しの完了

クリックする位置が決まれば、あとは単純な作業です。手作業でコードを書いても構いませんが、クリック位置は `pos` に保存されているので、コード生成を自動化します。

コード 6.22 (kybd-gen-code.R) : コード生成の自動化

```
for(p in pos){
  pre <- "mouse_move_click("
  mid <- ", "
  post <- ")\n"
  paste0(pre, p[1], mid, p[2], post) |>
  cat()
}
## mouse_move_click(225, 843)
## mouse_move_click(1055, 652)
## mouse_move_click(1179, 695)
## mouse_move_click(1021, 677)
```

USB メモリの安全な取り出しをする前の位置にマウスを戻したい場合は、`mouse.get_cursor()` でマウスの位置を予め取得して、最後にその位置に戻します。また、実際に USB メモリをパソコンに取り付けて、動作確認の必要があります。プログラムの実行速度が早すぎて、パソコンの描画や動作よりも先にクリックしてしまう場合があるかもしれません。その場合は、

`mouse_move_click()` で引数を `sleep_sec = 1` のように設定して、マウスの移動後に待機時間を挟んでください。

コード 6.23 (kybd-remove-usb.R) : USB の取り出しコードの例

```
pos_original <- KeyboardSimulator::mouse.get_cursor()
# スクリプトで使用時は、mouse_move_click()の定義が必要
mouse_move_click(225, 843) # 位置は適宜変更の必要あり
mouse_move_click(1055, 652)
```

```
mouse_move_click(1179, 695)
mouse_move_click(1021, 677)
mouse_move_click(pos_original[1], pos_original[2])
```

以下のように、上記の内容をスクリプトとして保存して、バッチモードで起動(5.3節)すれば、瞬間起動(5.1節)できます。ただし、コード `mouse_move_click()` の定義(コード 6.12)をスクリプトの中に入れる必要があります。

- ・ R のコードをテキストファイル `RemoveUsb.R` として保存
- ・ 起動用のバッチファイル `RemoveUsb.bat` を作成して保存
- ・ `RemoveUsb.bat` をクリックして、動作確認
- ・ パスの通ったディレクトリに、`RemoveUsb.bat` のショートカットを `ru` として保存
- ・ [Win] + [R] の「ファイル名を指定して実行」に `ru` と入力

USB メモリの取り出しの他に、たまに接続する wifi への接続、リモートデスクトップへの接続、タスクバーに固定されているアプリの操作などが自動化できるでしょう。

《Tips》 バッチファイルの内容で、`.R` と `.log` を書き換えるのが面倒なときは、以下のようにしても構いません。

```
R CMD BATCH --silent --vanilla %~n0.R %~n0.log
```

`hogehoge.bat` というバッチファイル名のとき、バッチファイルを実行すると、`%~n0` はバッチファイルの拡張子を除去した部分の意味になり、`%~n0.R` は `hogehoge.R`、`%~n0.log` は `hogehoge.log` として実行されます。

7 PDF の操作

PDF(Portable Document Format)ファイルは資料を共有するときに便利です。そのため、多くの場面で使用するでしょう。たとえば、会議の配布資料を作成するたびに、PDFの結合・ページ番号付け・資料番号付けをするかもしれません。慣れた作業とはいえ、同じことの繰り返しは面倒です。また、作業が完了して資料を配信する直前に、資料の追加を指示されて困ったことはないでしょうか。数分の作業でも、一旦完了した作業のやり直しはつらいものがあります。

`pdftools` パッケージを使えば、PDFの分割・抽出・結合・分割・文字列の抽出・文字認識(OCR)を自動化できます。また、PDFファイルの画像への変換や、PDFに含まれる画像を抽出することもできます。さらに、`qpdf` パッケージを使えばページの重ね合わせができます。

本章では、PDFの基本操作である分割・抽出・結合を説明するとともに、ユーザが入力したファイルとページを抽出するプログラムを作成します。また、これらの基本操作とページの重ね合わせを組み合わせることで、ページ番号を自動的に付ける方法を紹介します。このような作業をバッチモードから起動できるようにしておけば(5.3節を参照)，自動化作業がワンクリックで完了します。

7.1 PDF を操作するパッケージ

PDFを操作するパッケージとして`pdftools`があります。ページ単位での操作のみで、PDFに含まれる文字列自体の編集はできません。ページ番号の付加のための重ね合わせの関数は`pdftools`にはありませんので、`qpdf`を使います。`pdftools`と`qpdf`の両方ともCRANからインストールできます。`pdftools`と`qpdf`に同名の関数がありますが、どちらを使っても機能は同じです。

コード 7.1 (pdf-install.R) : `pdftools` と `qpdf` のインストール

```
install.packages("pdftools")
install.packages("qpdf")
```

`pdftools`は内部でPopplerというPDFを操作するライブラリが使われています。WindowsとMacでは`pdftools`と合わせて自動的にインストールされます。Linuxでは、Popplerを別途インストールしてください。Popplerのインストール方法は、以下のURLのInstallationの項目に記載されています。

<https://docs.ropensci.org/pdftools/>

コード 7.2 (pdf-library.R) : `pdftools` と `qpdf` の呼び出し

```
library(pdftools) # Popplerのバージョンが表示される
## Using poppler version 23.08.0
library(qpdf)
```

なお、`pdftools` と `qpdf` の関数の多くは、引数に `input`, `output`, `password` をとります。`input` と `output` は入力と出力するファイルのパスで、`output` は省略可能です。ファイルにパスワードが設定されているときは、`password` を指定します。

7.2 作業用の PDF ファイルの用意

本章で操作する作業用の PDF ファイルを用意します。次のコードではサポートページから PDF をダウンロードしますが、読者自身の PDF でも構いません。ただし、作業内容を理解するには文書だけでなく画像を含む複数ページの PDF が良いです。また、あまり大きすぎるファイルは避けたほうがよいでしょう。

なお、次のコードではファイルのダウンロードに curl パッケージを使います。必要に応じてインストールしてください。

コード 7.3 (pdf-download.R) : 作業用 PDF のダウンロード

```
# install.packages('curl')
url <- "https://matutosi.github.io/r-auto/data/base.pdf"
pdf_base <- fs::path_temp("base.pdf")
curl::curl_download(url, pdf_base) # url から PDF をダウンロード
```



図 7.1: 作業用にダウンロードした PDF

7.3 ページ数の取得

PDF のページ数の取得だけでは実用的な自動化はできませんが、ページ数が分からないと困ることがあります。たとえば、ページの抽出でページ数を超えた数値を指定するとエラーになるからです。そのため、PDF の自動化プログラムにはページ数の取得が必要です。

PDF のページ数を取得するには、`pdf_length()`を使います。引数 `input` にファイルのパスを指定します。パスワード付きの PDF のときは `password` を指定します。

コード 7.4 (pdf-length.R) : PDF のページ数の取得

```
pdf_length(pdf_base)
## [1] 5
```

7.4 ページの分割・抽出

PDF のページの分割には `pdf_split()` を、抽出には `pdf_subset()` を使用します。 `pdf_split()` は PDF の全ページを 1 ページごとに分割します。 `pdf_subset()` は指定したページを抽出して 1 つのファイルにします。

`pdf_split()` では、引数 `input` に分割するファイルのパス、`output` に出力ファイル名の接頭辞を指定しますが、`output` は省略可能です。 パスワードが必要な場合は、`password` を指定します。返り値は分割後のファイル名です。

次のコードでは、`pdf_base` を 1 ページごとに分割し、分割後のファイル名を `pdf_spl` に代入しています。

コード 7.5 (pdf-split.R) : PDF の分割

```
pdf_spl <- pdf_split(pdf_base)
fs::path_file(pdf_spl) # ファイル名のみ
## [1] "base_1.pdf" "base_2.pdf" "base_3.pdf" "base_4.pdf" "base_5.pdf"
```

`pdf_subset()` では、抽出するページを `pages` で指定します。`pages` で指定したページのみを抽出した PDF が生成されます(図 7.2)。

コード 7.6 (pdf-subset.R) : PDF のページ抽出

```
pdf_sub <- pdf_subset(pdf_base, pages = c(1, 5))
fs::path_file(pdf_sub)
## [1] "base_output.pdf"
```

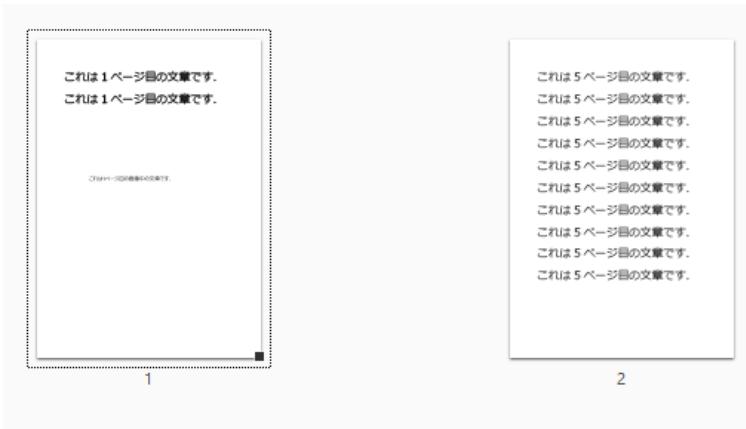


図 7.2: ページを抽出した PDF

`pdf_subset()`を使えば、引数 `pages` に指定したページで順序を入れ替えることもできます。次のコードでは、全ページを逆順、奇数ページのみ、奇数ページの逆順に入れ替えていきます。

コード 7.7 (pdf-subset-reverse.R) : PDF のページ順序の入れ替え

```
len <- pdf_length(pdf_base)
pdf_reverse <- pdf_subset(pdf_base, pages = len:1) # 逆順
odd_pages <- seq(from = 1, to = len, by = 2)          # 奇数ページのみ
pdf_odd <- pdf_subset(pdf_base, pages = odd_pages)
odd_rev <- sort(odd_pages, decreasing = TRUE)         # 奇数ページの逆順
pdf_odd_rev <- pdf_subset(pdf_base, pages = odd_rev)
```

ただし、`pdf_subset()`で同じページを重複して指定するとエラーになります。同じページを繰り返したい場合は、`pdf_subset()`か `pdf_split()`で必要なページを取り出してから、`pdf_combine()`で結合してください。

コード 7.8 (pdf-subset-dup.R) : ページの重複エラー

```
pdf_dup <- pdf_subset(pdf_base, pages = rep(1:3, 2)) # 重複はエラー
## Error in eval(expr, envir, enclos): empty PDF (page 3 (numbered from zero): object 3 0): duplicate page reference found; this would cause loss of data
```

PDF を 1 ページずつに分割する、あるいは抽出するページ番号や事前に決まっている作業はありませんかもしれません。ユーザが入力したページ番号を抽出する操作であれば、色々な場面で使えそうです。また、複数の PDF のうちユーザが指定した PDF のみを分割したいかもしれません。

以下では、作業ディレクトリ内の PDF を選択し、さらに指定したページを抽出する関数を紹介します。作成する関数 `subset_pdf()` の全体像は以下のとおりです。

- ・ 複数の PDF ファイルからファイルを選択して分割
 - ファイルの選択
 - ・ PDF ファイルの一覧を取得
 - ・ 選択肢の作成・提示

- ユーザ入力にもとづきファイルを選択
- ページの抽出
 - for ループでファイルごとの操作
 - PDF のページ数を取得
 - ページ入力の受付
 - ユーザ入力にもとづきページを抽出
 - 抽出したファイル名を返す

なお、次の関数の詳細はサポートページに記載しています。関数を使う際には、コード 7.9 で関数を読み込んでください。

- user_input() : ユーザからの入力を受け付ける関数
- eval_strings() : 文字列を数値として返す関数
- input_numbers() : ユーザ入力のページ数を数値に変換する関数
- gen_choices() : ファイルの一覧を選択肢として返す関数

コード 7.9 (pdf-source-extra.R) : ユーザからの入力関連の関数の読み込み

```
source("https://matutosi.github.io/r-auto/R/99__extra_funs.R")
```

コード 7.10 (pdf-subset-fun.R) : 複数の PDF ファイルからファイルを選択して分割する関数

```
subset_pdf <- function(){
  # ファイルの選択
  files <- fs::dir_ls(regexp = "\\.pdf$") # PDF ファイルの一覧取得
  if(length(files) == 0){
    message("PDF ファイルがありません")
    return(0)
  }
  if(length(files) > 1){
    choices <- gen_choices(files) # ファイルを選択肢に
    prompt <- "分割する PDF ファイルを選択してください\n"
    file_no <- input_numbers(prompt, choices)
    selected_files <- files[file_no]
  }
  # ページの抽出
  res <- list()
  for(file in selected_files){
    shell.exec(file)
    len <- pdf_length(file)
    prompt <-
      paste0("ファイル名：" , file, "\n",
             "ページ番号を指定してください。例：1,3,5-10\n",
             "
```

```

    "最大ページ数：" , len , "\n")
pages <- input_numbers(prompt)
res[[file]] <- pdftools::pdf_subset(file, pages)
}
return(res)
}

```

コード 7.11 (pdf-subset-exec.R) : 複数の PDF ファイルからファイルを選択して分割

```

subset_pdf()
## 分割する PDF ファイルを選択してください
## 1: a.pdf
## 2: b.pdf
## 2
## ファイル名：a_output.pdf
## ページ番号を指定してください。
## 例：1,3,5-10
## 最大ページ数：10
## 1,3,4
#   $a_output.pdf
# [1] "C:\\Users\\ユザーネーム\\a_output_output.pdf"

```

《応用》 マウスでファイルを選択する場合は、1.8 節を参照してください。

7.5 ページの結合

PDF を結合するには、`pdf_combine()`を使います。`input` に結合させたいファイル名をベクトルで指定します。次のコードでは、`pdf_split()`で 1 ページずつに分割したファイルを再度結合します。掘った穴を埋めるような作業ですが、練習だと思ってご了承ください。`pdf_length()`でページ数を確認すると結合できたことがわかります。

コード 7.12 (pdf-combine.R) : PDF の結合

```

pdf_spl |>                                # 分割した PDF
  purrr::map_int(pdf_length)      # 各 PDF のページ数
  ## [1] 1 1 1 1 1
pdf_com <- pdf_combine(pdf_spl) # 結合
fs::path_file(pdf_com)          # 結合したファイル名
## [1] "base_1_combined.pdf"
pdf_length(pdf_com)            # 結合した PDF のページ数
## [1] 5

```

会議資料などの準備で複数の PDF を 1 つに結合する作業をする場合は、対象の PDF を 1 つのディレクトリに集めてから `pdf_combine()` で結合すると良いでしょう。このとき、結合する順序を決め

ておく必要があります。一番簡単な方法は、ファイル名を 01_xxx.pdf のように連番を入れることです。ファイル名の順序にしたがって結合するのは容易です。

また、次のコードはユーザの入力でファイルの順序を指定する関数です。ファイル名の指定は、ページの分割・抽出(7.4 節を参照)と同じ関数を使っています。

コード 7.13 (pdf-combine-fun.R) : ディレクトリ内の PDF のうち指定したものと結合する関数

```
combine_pdf <- function() {  
  files <- fs::dir_ls(regexp = "\\.pdf$")  
  choices <- gen_choices(files)  
  prompt <- "結合するファイル番号を指定してください。例：2,5,1\n"  
  file_no <- input_numbers(prompt, choices)  
  files <- files[file_no]  
  pdf_combine(files)  
}
```

このコードをバッチモードで実行すると(5.3 節を参照)，バッチファイルと同じディレクトリにある PDF についてファイル名を指定して結合できます。

7.6 ページの回転

PDF のページの向きを回転させるには、qpdf:::pdf_rotate_pages()を使います。回転は angle(既定値は 90) と relative(既定値は FALSE) で指定します。angle は 90 度単位で指定し，relative = FALSE のときは元の状態に関わらず，angle = 0 で縦長に，angle = 90 で横長の方向に回転します(回転は時計回りの方向)。relative = TRUE のときは元の方向から時計回りに回転させます。引数 pages でページを指定すれば，該当ページのみ回転します(図 7.3)。

コード 7.14 (pdf-rotate.R) : PDF の回転

```
pdf_rtt <- pdf_rotate_pages(pdf_com, pages = c(1, 3))  
fs::path_file(pdf_rtt) # ファイル名のみ  
## [1] "base_1_combined_output.pdf"
```

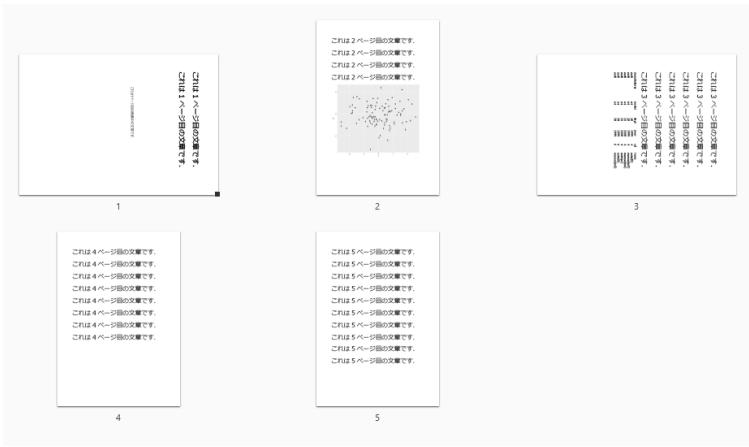


図 7.3: 指定ページを回転した PDF

7.7 圧縮・最適化

ふだん使用する PDF は圧縮されていることが多いですが、圧縮されていないときは `pdf_compress()` でファイルサイズを小さくできます。圧縮済みのファイルに対して `pdf_compress()` を使っても、ファイルサイズは変化しません。場合によってはファイルサイズが若干大きくなることがあります。

また、PDF ファイルを最適化していないと、ファイル内での情報がページの順序と一致していないことがあります。そのため、ファイルを開いたときに 1 ページ目がすぐには表示されない可能性があります。一方、最適化された PDF はページが順次表示されるため、表示を早めることができます。PDF の最適化は、`pdf_compress(linearize = TRUE)` と指定することで可能です。引数の `output` と `password` は必要に応じて指定します。

コード 7.15 (eval.R) : PDF の圧縮と最適化

```
pdf_compressed <- pdf_compress(pdf_base, linearize = TRUE)
```

あまり使わないとは思いますが、圧縮の反対の操作(decompress)をしたい場合は、次のようなオンラインサイトを使ってください。このサイトで `decompress` したファイルに対して、`pdf_compress()` を使うと圧縮したことを見ることができます。

<https://www.pdfyeah.com/decompress-pdf/>

7.8 文字列の抽出

`pdf_text()` を使うと、PDF に埋め込まれた文字列を抽出できます。返り値は、ページ単位での文字列ベクトルです。つまり、10 ページの PDF であれば、長さが 10 の文字列ベクトルです。ページ内の改行は `\n` です。もし、空白が沢山あって見にくいときは、`stringr` の `str_trim()` や `str_squish()` などの関数を使うと見やすくなります(3.7 節を参照)。

コード 7.16 (pdf-text.R) : PDF から文字列の抽出

```
text <-  
  pdf_spl[1:3] |> # 1-3 ページ  
  pdf_combine() |> # 結合  
  pdf_text()          # 文字列の抽出  
text |>  
  stringr::str_split("\n") # 改行(\n)で分割  
## [[1]]  
## [1] "これは 1 ページ目の文章です。" "これは 1 ページ目の文章です."  
## [3] ""  
##  
## [[2]]  
## [1] "これは 2 ページ目の文章です。" "これは 2 ページ目の文章です."  
## [3] "これは 2 ページ目の文章です。" "これは 2 ページ目の文章です."  
## [5] ""  
##  
## [[3]]  
## [1] "これは 3 ページ目の文章です."  
## [2] "これは 3 ページ目の文章です."  
## [3] "これは 3 ページ目の文章です."  
## [4] "これは 3 ページ目の文章です."  
## [5] "これは 3 ページ目の文章です."  
## [6] "これは 3 ページ目の文章です."  
## [7] "manufacturer model displ year cyl trans"  
## [8] "audi a4 1.8 1999 4 auto(15)"  
## [9] "audi a4 1.8 1999 4 manual(m5)"  
## [10] "audi a4 2.0 2008 4 manual(m6)"  
## [11] "audi a4 2.0 2008 4 auto(av)"  
## [12] "audi a4 2.8 1999 6 auto(15)"  
## [13] "audi a4 2.8 1999 6 manual(m5)"  
## [14] ""
```

【追加】 「コメント：この辺に抽出の例の画像があるとよいかも」ともらっていました。→ 画像でなくても、抽出したものがわかれれば良いと思いますので、抽出後の文字列の出力を入れました。

7.9 画像ファイルへの変換

PDF を 1 ページごとに画像ファイルに変換して書き込むには、`pdf_convert()`を使います。ファイル形式の引数である `format` の既定値は "png" ですが "jpeg" も可能です。

画像の解像度は `dpi` で指定します。既定値は 72 ですが、ちょっと荒いです(図 7.4)。必要に応じて 300 などと指定してください(図 7.5)。

また、`pdf_convert()` にページ数の多い PDF は実行にかなりの時間がかかります。場合によっては、処理しきれない可能性もあります。そのため、`pdf_split()` や `pdf_subset()` で必要なページのみを抽出してから実行することをお勧めします。次のコードでは、分割した PDF の 1 ページ目

と2ページ目を結合してから画像へ72dpi(既定値)と300dpで変換しています。300dpiの画像は次節でPDFファイルに変換しなおすために、パスをpngsという変数に代入しています。

コード7.17 (pdf-convert.R) : PDFを画像ファイルに変換

```
# pdf_convert(pdf_base, pages = 1:2) # かなり時間がかかる
pdf_combine(pdf_spl[1:2]) |>
  pdf_convert(filenames = paste0("072_", 1:2, ".png")) # 既定値の72dpi
## Converting page 1 to 072_1.png... done!
## Converting page 2 to 072_2.png... done!
## [1] "072_1.png" "072_2.png"
pngs <- pdf_combine(pdf_spl[1:2]) |>
  pdf_convert(filenames = paste0("300_", 1:2, ".png"), dpi = 300)
## Converting page 1 to 300_1.png... done!
## Converting page 2 to 300_2.png... done!
```



図7.4: 72dpiで変換した画像



図7.5: 300dpiで変換した画像

一定のファイルサイズ以内で、きれいな画像を求める場合は、forループやpurrr::map()を使って複数のdpiを試します。その後、生成されたファイルの情報をfs::dir_info()で取得して(1.4節を参照)，目的のサイズ内で最大サイズのファイルを使うと良いでしょう。画像サイズのファイルサイズを変更するコードは、11.12節を参考にしてください。

7.10 PDFの文字認識

pdf_ocr_text()とpdf_ocr_data()は、PDFファイルの内容を画像ファイルに一旦変換してから文字認識します。そのため、文字データであろうが画像であろうが、文字認識が可能です。もとの文字データが正しく文字認識されるとは限りません。PDFにある文字データを正確に抽出するにはpdf_text()を使ってください。なお、pdf_ocr_data()やpdf_ocr_text()で文字認識しても、もとのPDFファイルはそのままで変化はありません。OCRで認識した文字データは、返り値として得られるだけです。

文字認識には内部でtesseractパッケージを使っているため、あらかじめインストールしておきます。また文字認識に使用する言語のモデルをダウンロードします。日本語の場合は、

`tesseract_download(lang = "jpn")`とします。なお、言語モデルのダウンロードは初回だけです。

コード 7.18 (pdf-tesseract-install.R) : tesseract のインストール

```
install.packages("tesseract")
tesseract::tesseract_download(lang = "jpn")
```

通常の文字認識として使う場合は `pdf_ocr_text()` が適しており、高度な文字認識をするには `pdf_ocr_data()` を使います。2つの関数の違いは返り値の内容です。`pdf_ocr_text()` は文字列を返り値として返します。一方、`pdf_ocr_data()` は読み取り精度や文字の位置など結果の詳細をデータフレームとして返します。このデータフレームには、`word(単語)`, `confidence(信頼度)`, `bbox(画像領域の位置)` が含まれています。

コード 7.19 (pdf-ocr.R) : PDF 内の画像の文字認識

```
ocr_data <-
  pdf_ocr_data(pdf_spl[1], language = "jpn") |>
  magrittr::extract2(1) # [[1]]と同じ
  ## Converting page 1 to base_1_1.png... done!
head(ocr_data)
## # A tibble: 6 × 3
##   word  confidence bbox
##   <chr>     <dbl> <chr>
## 1 ゴ          69.5 622,736,752,915
## 2 これ        79.7 752,712,1334,921
## 3 は          95.4 1064,699,1266,967
## 4 1          95.1 1435,727,1533,912
## 5 ペー        92.9 1734,725,2048,904
## 6 ジ          93.0 2139,719,2360,920
pdf_ocr_text(pdf_spl[1], language = "jpn") |>
  stringr::str_split("\n") |>
  magrittr::extract2(1) |> # [[1]]と同じ
head()
## Converting page 1 to base_1_1.png... done!
## [1] "ごこれは 1 ページ目の文草です."
## [2] "これは 1 ページ目の文章です."
## [3] "1.050"
## [4] "1.025"
## [5] "> 1.000           これは 1 ページ目の画父中の文章です."
## [6] "0.975"
```

コード 7.19 の"ごこれは"の冒頭の"ご"や"画父中"("像中")のように、誤認識があります。"こご"の部分は同じ部分に対して複数の読み取りをしてしまった結果です。

読み取り精度の良い部分のみを取り出すには、次のコードのように `filter()` などを使います。

コード 7.20 (pdf-ocr-filter.R) : 精度の高い結果のみを抽出

```
word <-  
  ocr_data |>  
  dplyr::filter(confidence > 75) |>  
  `\$`(_,"word") |> # $word の取り出し  
  paste0(collapse = "") # 文字列の結合
```

こうすると、ある程度以上の精度の読み取り結果のみを抽出できます。

7.11 画像の抽出

pdftools には PDF から画像を抽出する関数がありませんので、Poppler(R とは別のアプリ)を直接使います。Windows で pdftools をインストールしても、Poppler の全機能は使えませんので、Poppler 自体をインストールしてください。Poppler をインストールしたディレクトリにはパスを通す(5.1.1 項を参照)ことをお勧めします。

画像の抽出用の pdfimages.exe を使って PDF に含まれる画像を抽出する関数を定義します。関数内では、fs パッケージ(第 1 章を参照)でパスの設定をして、system()(5.4 節を参照)で pdfimages.exe を実行しています。

コード 7.21 (pdf-extract-images-fun.R) : PDF に含まれる画像を抽出する関数

```
extract_images <- function(pdf, out = fs::path_temp(), bin_dir = ""){  
  f_name <-  
    fs::path_file(pdf) |>                                # ファイル名のみ  
    fs::path_ext_remove() |>                            # 拡張子の除去  
  out_dir <- fs::path(out, f_name) |>                # 出力ディレクトリ  
  dir_create(out_dir)  
  out_file <- fs::path(out_dir, f_name) |>          # 出力ファイル  
  bin <- "pdfimages"  
  if(bin_dir != ""){  
    bin <- fs::path(bin_dir, bin) |>                  # 実行ファイルのディレクトリ  
  }  
  cmd <- paste0(bin, " -all ", pdf, " ", out_file) |> # 実行コマンド  
  paste0("Running ", cmd) |>                         # 実行中の表示  
  message()  
  system(cmd, intern = TRUE) |>                      # コマンド実行  
  iconv("sjis", "utf8") |>                           # 文字化け対策  
  message()  
  return(out_dir)  
}
```

定義した関数を使って画像を抽出します。たとえば、図 7.6 の PDF からは図 7.7 のように画像部分が抽出されます。

コード 7.22 (pdf-extract-images.R) : PDF に含まれる画像の抽出

```
image_dir <- extract_images(pdf_base)
## 'Runnning pdfimages -all # (ファイル名以降省略)
## shell.exec(image_dir) # ディレクトリを表示
```

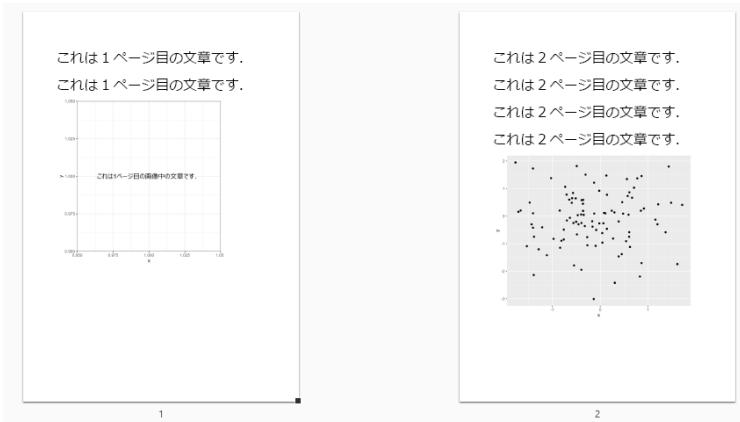


図 7.6: PDF 内の画像

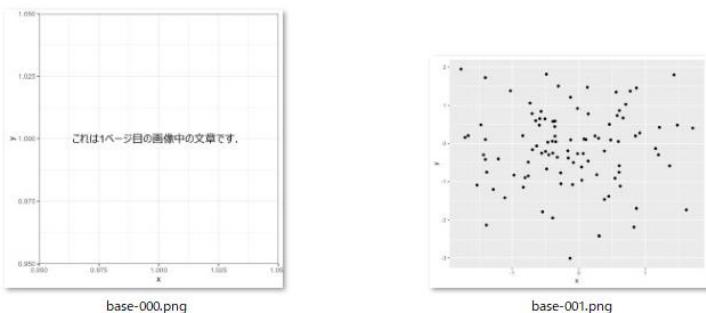


図 7.7: PDF から抽出した画像

7.12 重ね合わせ

PDF にページ番号や「資料 1」のような文字などを重ね合わせるには,
qpdf::pdf_overlay_stamp()を使います。

ページ番号を重ね合わせるためには、ページ番号だけの PDF を準備する必要があります。ここでは ggplot2 パッケージ(2.7 節を参照)を活用します。ggplot2 の関数の詳細は、Wickham (2012)(英語版は第 3 版出版予定)か以下の URL をご覧ください。

<https://ggplot2-book.org/>

まずは、縦横比を A4 サイズに合わせた xy 座標(横 210, 縦 297)に label で指定したページ番号を描画する関数を作成します。ここで用紙サイズは A4 版の縦書きですが、width と height で大きさを変更できます。また、下部の中央に設定したページ番号の位置は、x_pos と y_pos で変更

可能です。左下が原点ですので、`x_pos` と `y_pos` をそれぞれ `width` と `height` に近い数値に設定すると、右上の部分に文字を重ね合わせできます。

《応用》他の点もユーザのみなさんが使いやすいように修正して下さい。たとえば、「資料1」のような資料番号やロゴマークなどをヘッダーやフッター部分に付加することができます。また、PDFの分割・抽出(7.4節を参照)や結合(7.5節を参照)と合わせることで、必要な部分にだけ重ね合わせするように作業を自動化できます。

コード 7.23 (`pdf-plot-page-number.R`) : ページ番号だけのページを作成する関数

```
plot_page_number <- function(label, x_pos = width / 2, y_pos = 5,
                               size = 5, colour = "black",
                               width = 210, height = 297, ...){
  tibble::tibble(x_pos = x_pos, y_pos = y_pos, label = label) |>
    ggplot2::ggplot(ggplot2::aes(x_pos, y_pos, label = label)) +
      ggplot2::geom_text(size = size, colour = colour, ...) + # ページ数の描画
      # x 軸と y 軸の設定
      ggplot2::scale_x_continuous(limits = c(0, width), expand = c(0, 0)) +
      ggplot2::scale_y_continuous(limits = c(0, height), expand = c(0, 0)) +
      ggplot2::theme_void()
}
```

次に、`plot_page_number()`を利用して `n` ページ分の PDF を生成する関数を定義します。`size` でページ番号の文字サイズを変更できます。

コード 7.24 (`pdf-gen-page-numbers-fun.R`) : 複数ページ分のページ番号の PDF を生成する関数

```
gen_page_numbers <- function(n, x_pos = width / 2, y_pos = 5,
                               size = 5, colour = "black",
                               width = 210, height = 297, ...){
  pages <- seq(n)
  filename <- fs::path_temp(paste0(pages, ".pdf"))
  page_numbers <-
    purrr::map(pages, plot_page_number, # ページごとで繰り返し
               x_pos = x_pos, y_pos = y_pos,
               size = size, colour = colour,
               width = width, height = height, ...)
  purrr::walk2( # ページごとにファイルを書き込み
    filename, page_numbers,
    ggplot2::ggsave,
    width = width, height = height, units = "mm", bg = "transparent")
  return(unlist(filename))
}
```

ここではページ番号をわかりやすくするために、大きなページ番号を作成します(図 7.8)。なお、通常のページサイズであれば、size = 5 で良いでしょう。

コード 7.25 (pdf-gen-page-numbers.R) : ページ番号を大きく作成

```
pdf_pages <- gen_page_numbers(n = 10, size = 200, y_pos = 150)
```



図 7.8: 大きく作成したページ番号

ページ番号を付ける前に PDF を pdf_split() で 1 ページごとに分割します。分割したファイルとページ番号を pdf_overlay_stamp() で重ね合わせて、最後に pdf_combine() で結合すればページ番号を付加した PDF ができます。

ページ番号を追加する pdf_overlay_stamp() ですが、重ね合わせる文字は、引数 size で文字の大きさを、colour で文字の色をそれぞれ指定できます。

PDF の重ね合わせをすると、重なった部分の下側の内容は見えなくなります。以下の関数 add_page_numbers() では、ページ番号の上にもとの PDF を重ねわせています。重ね合わせの上下の順序を入れ替えたいときは backside = TRUE としてください。

【追加・変更点】 「透かし」のやり方が分かりましたので、以下の Tips に追加しました。

《Tips》 pdf_overlay_stamp() で重ね合わせをするときに、上側に重ねる PDF の背景色が透明でなければ、下側の PDF ファイルの内容は隠れて見えなくなります。コード 7.24 では、ggsave() で bg = "transparent" と指定することで、背景色を透明にしています。また、既存の PDF ファイルの背景を透明にするには、PDF ファイルを magick パッケージの image_read_pdf() で読み込み、image_transparent() で背景色を透明にしてから、image_write() で PDF に保存すると良いでしょう(11.18 節を参照)。また、上側が文字列のときは、文字列を薄い色にすれば、透かしに似たものになります。白黒であれば colour = grey(0.8) のように色を指定すると薄い灰色になります。なお、grey() は灰色の色指定をつくる関数で、grey(0) が黒、grey(1) が白です。

コード 7.26 (pdf-add-page-numbers-fun.R) : ページ番号を重ね合わせる関数

```
add_page_numbers <- function(path, y_pos = 5, size = 5,
                               colour = "black", backside = FALSE, ...){
```

```

pdf_spl <- pdftools::pdf_split(path) # 分割
n <- length(pdf_spl)                  # PDF のページ数
pdf_pages <-
  gen_page_numbers(n = n, y_pos = y_pos, # ページ番号の PDF 生成
                   size = size, colour = colour, ...)
if(backside){ # 上下入れ替え
  tmp <- pdf_pages
  pdf_pages <- pdf_spl
  pdf_spl <- tmp
}
pdf_paged <- purrr::map2_chr(
  pdf_pages, pdf_spl, qpdf::pdf_overlay_stamp) # ページ番号の重ね合わせ
pdf_com <- pdftools::pdf_combine(pdf_paged)      # PDF の結合
fs::file_delete(c(pdf_pages, pdf_paged))          # 不要な PDF の削除
return(pdf_com)
}

```

これで材料が揃いましたので、ページ番号を付けます(図 7.9)。ここでは見やすくするために、大きな数字を使っていますが、実際には適切なサイズで実行してください。

コード 7.27 (pdf-add-page-numbers.R) : ページ番号の重ね合わせ

```
pdf_paged <- add_page_numbers(pdf_base, size = 200, y_pos = 150, colour = "yellow")
```

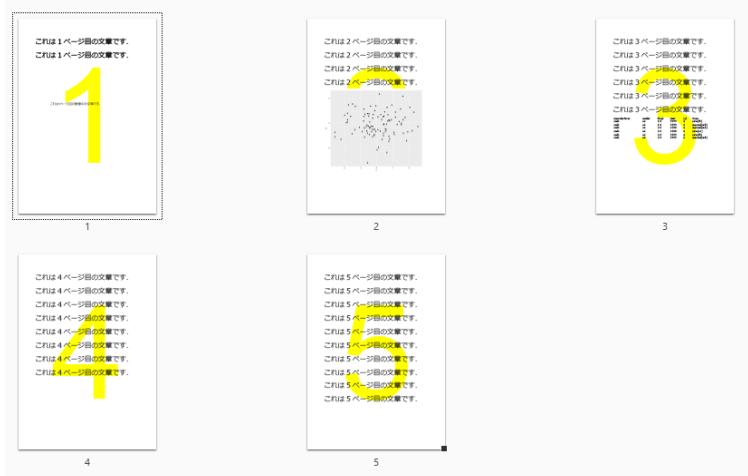


図 7.9: 大きなページ番号を付けた PDF

8 ワードの操作

ワードは文字のサイズを変更したり、図や表を入れたりできるので、文書作成には欠かせないアプリです。ただし、動作はあまり早くないため、テキストエディタを使う人には、ワードでの文書の作成や修正は面倒な作業です。ワードでも文字列の置換はできますが、文字列の検索や置換で高度な作業はできません。複数ファイルで同じ文字列を置換するには、同じ作業の繰り返しが必要です。

`officer` パッケージでは、正規表現(3.4 節を参照)を使えます。また、ワードでファイルを一つずつ開く必要はありませんので、ワード本体で実行するよりも高度な検索や置換を自動的かつ迅速に実行できます。`lubridate`(第4章を参照)と合わせて使えば、日付や曜日に関する作業も自動化できます。たとえば、日付と曜日の対応が正しいかの確認や、定期的に作成する月報や日報での日付・曜日の更新です。もちろん、複数ファイルを対象にしても、手間は増えません。

定期的な会議の資料を準備するときに、ワードのファイルを PDF に変換することがあるかもしれません。ファイルを一つずつ開いて、変換して、閉じてという作業は面倒なものです。

`RDCOMClient` パッケージを使えば複数ファイルの PDF のへの変換を自動化できます。変換後に `pdftools` で結合(7.5 節を参照)すれば一気に作業が完了します。なお、`RDCOMClient` パッケージでは `html`、リッチテキスト、テキストファイルへの変換も可能です。

`docx` 形式のファイルは文書の中身を `zip` 形式でまとめたものです。そのため、`zip` ファイルとして解凍すれば、ファイルを開かずにワードファイルの中の画像の抽出が自動化できます。本章ではその方法を紹介します。

8.1 ワードを操作するパッケージ

ワードを操作するには、`officer` パッケージと `RDCOMClient` パッケージを使います。`officer` はワードだけでなくパワーポイントを操作できます(第10章を参照)。`RDCOMClient` はワードだけでなく、エクセル(9.14 節を参照)やパワーポイント(10.14 節を参照)などを操作できます。なお、`RDCOMClient` からワードを操作するときには、ワードがインストールされている必要があります。

`officer` は CRAN からインストール可能です。

コード 8.1 (word-install.R) : `officer` のインストール

```
install.packages("officer")
```

`RDCOMClient` は CRAN には登録されていませんので GitHub からインストールします。`RDCOMClient` は `zip` ファイルでのインストールする方法とソースファイルからビルドしてインストールする方法があります。どちらかでインストールできれば構いません。

R のバージョンにあった `zip` ファイルがないときにはソースファイルからビルドします。ただし、事前に `RTools` が必要で、インストール自体にも時間がかかります。`Rtools` は以下の URL からインストールしてください。

<https://cran.r-project.org/bin/windows/Rtools/>

```
# zip ファイルでのインストール  
install.packages("RDCOMClient",  
                  repos = "http://www.omegahat.net/R", type = "win.binary")  
# ソースファイルからビルドしてインストール  
# install.packages("remotes") # remotes をインストールしていないとき  
# Rtools も必要  
remotes::install_github("omegahat/RDCOMClient")
```

インストールができれば library() で呼び出します。

コード 8.2 (word-library.R) : officer と RDCOMClient の呼び出し

```
library("officer")  
library("RDCOMClient")
```

8.2 ワードの読み込み

まず、作業用のファイルをダウンロードします。ダウンロードには curl パッケージを使いますので、必要に応じてインストールしてください。読者自身のファイルでも構いません。ただし、画像を抽出しますので、画像を含むファイルが良いでしょう。

コード 8.3 (word-download.R) : 作業用ファイルのダウンロード

```
# install.packages("curl")  
url <- "https://matutosi.github.io/r-auto/data/doc_1.docx"  
path_doc_1 <-  
  fs::path_file(url) |>  
  fs::path_temp()  
curl::curl_download(url, path_doc_1) # url から PDF をダウンロード  
# shell.exec(path_doc_1)
```

ダウンロードしたファイルの内容は、図 8.1 のとおりです。なお、ページ間の余白は非表示にしています。

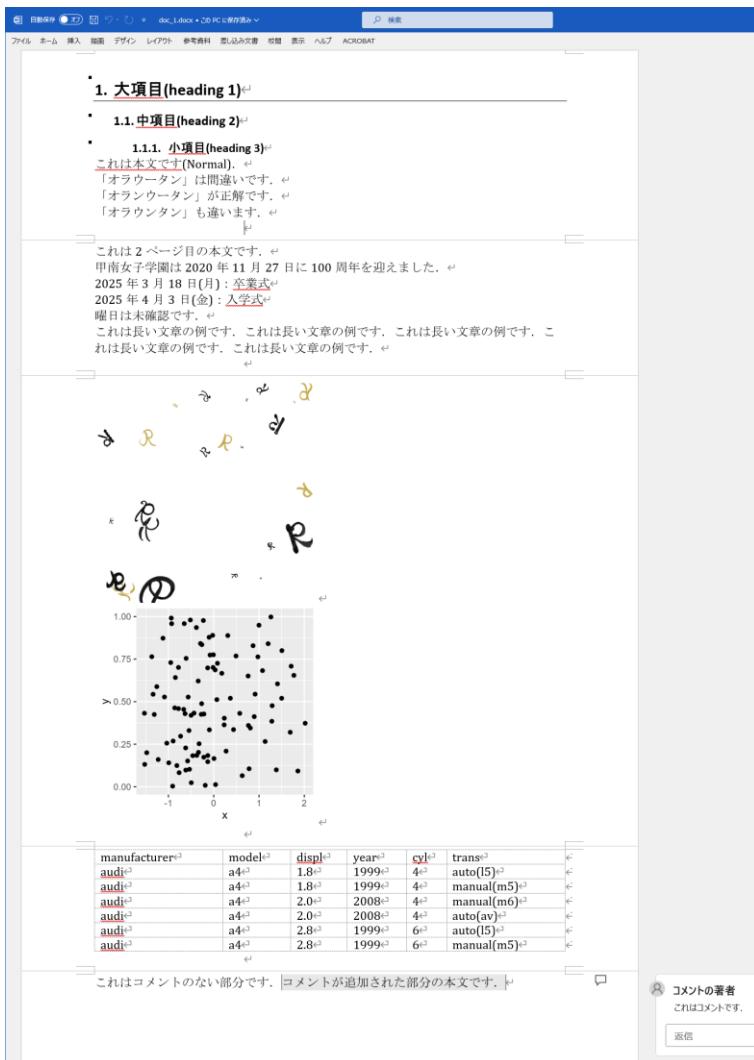


図 8.1: ダウンロードしたワードのファイル(ページ間の余白は非表示)

ワードのファイルを読み込むには `read_docx()`を使います。読み込んだものを表示させると、文書に含まれる書式(styles)とカーソル位置の内容(Content at cursor location)が表示されます。

コード 8.4 (word-read-docx.R) : ワードファイルの読み込み

```
doc_1 <- read_docx(path_doc_1)
doc_1
## rdocx document with 12 element(s)
##
## * styles:
##           Normal             heading 1             heading 2
##           "paragraph"         "paragraph"        "paragraph"
##           "heading 3 Default Paragraph Font" "character"      "character"
##           "paragraph"         "strong"          "centered"
##           "No List"          "numbering"       "paragraph"
##           "numbering"         "character"      "paragraph"
## (中略)
## * Content at cursor location:
##           row_id is_header cell_id      text col_span row_span content_type
## 1.1      1    TRUE      1 manufacturer      1      1   table cell
## 1.7      2   FALSE      1      audi      1      1   table cell
## 1.13     3   FALSE      1      audi      1      1   table cell
## 1.19     4   FALSE      1      audi      1      1   table cell
```

```
## 1.25      5    FALSE      1       audi      1      1  table cell
## (以下省略)
```

8.3 文書の書き出し

文書をワードのファイルとして書き出すには、`print()`を使います。`x`に保存するオブジェクトを、`target`にパスを指定します。

読み込んだワード文書をそのまま保存するだけですが、`doc_1`を一時ディレクトリに`doc_2.docx`として保存するには以下のようにします。

コード 8.5 (word-print.R) : 文書の書き出し

```
path_doc_2 <- fs::path_temp("doc_2.docx")
print(x = doc_1, target = path_doc_2)
# shell.exec(path_doc_2)
```

8.4 概要の表示と内容の取り出し

文書の概要を表示するには、`docx_summary()`を使います。ここでは、見やすく表示するために`tibble`に変換します。

コード 8.6 (word-docx-summary.R) : 概要の表示

```
doc_1 |>
  docx_summary() |>
  tibble::as_tibble() |>
  head()

## # A tibble: 6 × 11
##   doc_index content_type style_name text          level num_id row_id
##       <int>     <chr>      <chr>     <chr>        <dbl> <int> <int>
## 1       1 paragraph   heading 1 大項目(headin...     NA    NA    NA
## 2       2 paragraph   heading 2 中項目(headin...     NA    NA    NA
## 3       3 paragraph   heading 3 小項目(headin...     NA    NA    NA
## 4       4 paragraph     Normal    これは本文で...     NA    NA    NA
## 5       5 paragraph     Normal    「オラウータ...     NA    NA    NA
## 6       6 paragraph     Normal    「オランワー...     NA    NA    NA
## # ℹ 4 more variables: is_header <lgl>, cell_id <dbl>, col_span <dbl>,
## #   row_span <int>
```

データフレーム形式で文書の概要が表示されます。`content_type`には`paragraph`(段落)や`table cell`(表)などが表示されます。さらに書式が設定されているときは`style_name`列に表示されます。`text`列には本文が表示されます。

本文の概要はデータフレームですので、 dplyr(2.5 節を参照)の関数が使えます。見出しも含めた本文は content_type == "paragraph" とすれば取り出せます。

コード 8.7 (word-docx-summary-filter-paragraph.R) : 見出しを含む本文の取り出し

```
doc_1 |>
  docx_summary() |>
  tibble::as_tibble() |>
  dplyr::filter(content_type == "paragraph") |>
  dplyr::select(content_type, style_name, text) |>
  dplyr::filter(text != "")  
## # A tibble: 14 × 3  
##   content_type style_name text  
##   <chr>        <chr>      <chr>  
## 1 paragraph    heading 1  大項目(heading 1)  
## 2 paragraph    heading 2  中項目(heading 2)  
## 3 paragraph    heading 3  小項目(heading 3)  
## 4 paragraph    Normal     これは本文です(Normal).  
## 5 paragraph    Normal     「オラウータン」は間違います。  
## 6 paragraph    Normal     「オランウータン」が正解です。  
## 7 paragraph    Normal     「オラウンタン」も違います。  
## 8 paragraph    Normal     これは 2 ページ目の本文です。  
## 9 paragraph    Normal     甲南女子学園は 2020 年 11 月 27 日に 100 周年を迎...  
## 10 paragraph   Normal     2025 年 3 月 18 日(月) : 卒業式  
## 11 paragraph   Normal     2025 年 4 月 3 日(金) : 入学式  
## 12 paragraph   Normal     曜日は未確認です。  
## 13 paragraph   Normal     これは長い文章の例です。これは長い文章の例...  
## 14 paragraph   Normal     これはコメントのない部分です。コメントが追...
```

見出しを除いた本文のみ取り出すのには、 style_name == "Normal" とします。

コード 8.8 (word-docx-summary-filter-normal.R) : 見出しを除く本文の取り出し

```
doc_1 |>
  docx_summary() |>
  tibble::as_tibble() |>
  dplyr::filter(style_name == "Normal") |>
  dplyr::select(content_type, style_name, text) |>
  dplyr::filter(text != "")  
## # A tibble: 11 × 3  
##   content_type style_name text  
##   <chr>        <chr>      <chr>  
## 1 paragraph    Normal     これは本文です(Normal).  
## 2 paragraph    Normal     「オラウータン」は間違います。  
## 3 paragraph    Normal     「オランウータン」が正解です。  
## 4 paragraph    Normal     「オラウンタン」も違います。
```

```

## 5 paragraph Normal これは 2 ページ目の本文です。
## 6 paragraph Normal 甲南女子学園は 2020 年 11 月 27 日に 100 周年を迎...
## 7 paragraph Normal 2025 年 3 月 18 日(月)：卒業式
## 8 paragraph Normal 2025 年 4 月 3 日(金)：入学式
## 9 paragraph Normal 曜日は未確認です。
## 10 paragraph Normal これは長い文章の例です。これは長い文章の例...
## 11 paragraph Normal これはコメントのない部分です。コメントが追...

```

その他の部分も、該当する `content_type` や `style_name` を指定して取り出せます。

過去に編集したワードのファイルの変更点が分からなくなり、文字列を比較することがあります（3.8 節を参照）。そのときには、ワードから文字列を取り出す必要があります。ワードの内容を翻訳するときにもファイルから本文を取り出します。見出し、本文、あるいはその両方を選択して文字列を取り出せれば便利ですので、文字列を取り出す関数を作成します。取り出す条件を `condition` としてまとめて、`filter()` と `str_detect()` を組み合わせて抽出します。

コード 8.9 (`word-docx-extract-text-fun.R`) : ワードから文字列を抽出する関数

```

extract_docx_text <- function(docx, normal = TRUE, heading = TRUE, flatten = TRUE){
  if(sum(normal, heading) == 0){ # 両方とも FALSE のとき
    return("") # ""を返す
  }
  condition <- # 検索条件："normal|heading", "normal", "heading" のうち 1 つ
  c("Normal"[normal], "heading"[heading]) |>
    paste0(collapse = "|")
  text <- # 文字列
  docx |>
    officer::docx_summary() |>
    dplyr::filter(stringr::str_detect(style_name, condition)) |>
    dplyr::filter(text != "") |>
    dplyr::select(text)
  if(flatten) text <- text$text
  return(text)
}

```

定義した関数で文字列を抽出します。

コード 8.10 (`word-docx-extract-text.R`) : ワードから文字列の抽出

```

extract_docx_text(doc_1, heading = FALSE)
## [1] "これは本文です(Normal)."
## [2] "「オラウータン」は間違います。"
## [3] "「オランウータン」が正解です。"
## [4] "「オラウンタン」も違います。"
## [5] "これは 2 ページ目の本文です。"

```

```

## [6] "甲南女子学園は2020年11月27日に100周年を迎えました。"
## [7] "2025年3月18日(月)：卒業式"
## [8] "2025年4月3日(金)：入学式"
## [9] "曜日は未確認です。"
## [10] "これは長い文章の例です。これは長い文章の例です。これは長い文章の例で..."
## [11] "これはコメントのない部分です。コメントが追加された部分の本文です。"
extract_docx_text(doc_1, normal = FALSE, flatten = FALSE)
##          text
## 1 大項目(heading 1)
## 2 中項目(heading 2)
## 3 小項目(heading 3)

```

`write_tsv()`でテキストファイルに保存できます(図 8.2).

コード 8.11 (word-save-text.R) : ワードの内容をテキストとして保存

```

path_txt <- fs::path_temp("doc.txt")
doc_1 |>
  extract_docx_text(flatten = FALSE) |>
  readr::write_tsv(path_txt, col_names = FALSE)
# shell.exec(path_txt)

```

The screenshot shows a Microsoft Word document window with the title bar 'C:\Users\matutosi\AppData\Local\Temp\RtmpCyP2My\doc.txt [UTF-8] [LF] - 秀丸'. The document content is as follows:

```

(見出し無し)
- これは本文です(Normal). ↓
「オラウータン」は間違いです。 ↓
「オランウータン」が正解です。 ↓
「オラウンタン」も違います。 ↓
これは2ページ目の本文です。 ↓
甲南女子学園は2020年11月27日に100周年を迎えました。 ↓
2025年3月18日(月)：卒業式 ↓
2025年4月3日(金)：入学式 ↓
曜日は未確認です。 ↓
これは長い文章の例です。これは長い文章の例です。これは長い文
章の例です。これは長い文章の例です。これは長い文章の例です。 ↓
これはコメントのない部分です。コメントが追加された部分の本文
です。 ↓
[EOF]

```

図 8.2: ワードの内容を保存したテキストファイル

8.5 文字列の一括置換

本文の文字列の一括置換には、`body_replace_all_text()`を使います。また、ヘッダーの置換には`headers_replace_all_text()`を、フッターの置換には`footers_replace_all_text()`をそれぞれ使います。第1引数の`old_value`に置換前の文字列を、第2引数の`new_value`に置換後の文字列を指定します。

"オラウータン"と"オラウンタン"を個別に"オランウータン"に置換しても問題ありませんが、引数 `old_value` と `new_value` には正規表現(3.4 節を参照)が使えます。任意の 1 文字の意味の.(ドット)を使うと、"オラウ.タン"とすれば"オラウータン"と"オラウンタン"の両方を一括で置換できます。

コード 8.12 (word-replace-regexp-1.R) : 正規表現による置換

```
doc_1 |> # 置換前
  extract_docx_text(heading = FALSE) |>
  `[,`(_ , 1:5)
## [1] "これは本文です(Normal)."      "「オラウータン」は間違います。"
## [3] "「オランウータン」が正解です。" "「オラウンタン」も違います。"
## [5] "これは 2 ページ目の本文です。"
doc_1 <- # 正規表現による置換
  body_replace_all_text(doc_1, "オラウ.タン", "オランウータン")
doc_1 |> # 置換後
  extract_docx_text(heading = FALSE) |>
  `[,`(_ , 1:5)
## [1] "これは本文です(Normal)."      "「オランウータン」は間違います。"
## [3] "「オランウータン」が正解です。" "「オランウータン」も違います。"
## [5] "これは 2 ページ目の本文です。"
```

【相談】ワードで実行しそうな文字列の置換として、「"第 n 回"」の例を書いたのですが、不要であればその旨コメントください。あるいは、書くとすれば文字列の正規表現のところでどうか。

また、"第 n 回"(n は数字)を"n 回目"のようにしたいとき、"第"を削除して"回"を"回目"に置換することを考えます。"次第"から"第"が欠落したり、"回転"を"回目転"に置換したりします。正規表現で `body_replace_all_text("第(\d)回", "\1 回目")` としてマッチした部分を参照して使えます。"\d"は任意の数字を、"()"はマッチした部分のまとまりを、"\1"はマッチした部分の 1 つ目を表します。

コード 8.13 (word-replace-regexp-2.R) : 正規表現による置換(マッチ部分の参照)

```
str <- c(paste0("第", 1:3, "回"), "次第", "回転")
str
## [1] "第 1 回" "第 2 回" "第 3 回" "次第"   "回転"
stringr::str_replace_all(str, "第(\d)回", "\1 回目")
## [1] "1 回目" "2 回目" "3 回目" "次第"   "回転"
```

《Tips》A と B を入れ替えるとき、A を B に置換してから、B を A に置換すると、最初の A も B もすべて A になってしまいます。1 つ目の置換で A も B もすべて B になるためです。そのようなときは、中間変数を使ってください。A を C に置換して、次に B を A に置換して、最後に C を B に

置換します。ただし、中間変数である `c` の文字列は、文章内には出てこない文字列を使ってください。

8.6 文字の書式変更

文字の書式は、"Normal"や"heading 1"のように書式(styles)として保存されています。書式を変更することで、該当する部分のフォントサイズとフォントタイプを一括で変更できます。

"Normal", "heading 1", "heading 2", "heading 3"のフォントサイズとフォントタイプを変更するには次のようにします。

コード 8.14 (word-set-font.R) : フォントサイズとフォントタイプを変更

```
doc_1 <-  
  doc_1 |>  
  docx_set_paragraph_style(style_id = "Normal", style_name = "Normal",  
    fp_t = fp_text(font.size = 18, font = "Yu Gothic")) |>  
  docx_set_paragraph_style(style_id = "Titre1", style_name = "heading 1",  
    fp_t = fp_text(font.size = 40, font = "MS Gothic")) |>  
  docx_set_paragraph_style(style_id = "Titre2", style_name = "heading 2",  
    fp_t = fp_text(font.size = 30, font = "MS Mincho")) |>  
  docx_set_paragraph_style(style_id = "Titre3", style_name = "heading 3",  
    fp_t = fp_text(font.size = 20, font = "UD デジタル 教科書体 NK-R"))  
print(x = doc_1, target = path_doc_1)
```

ワードで確認すると、フォントが変更されていることがわかります(図 8.3)。

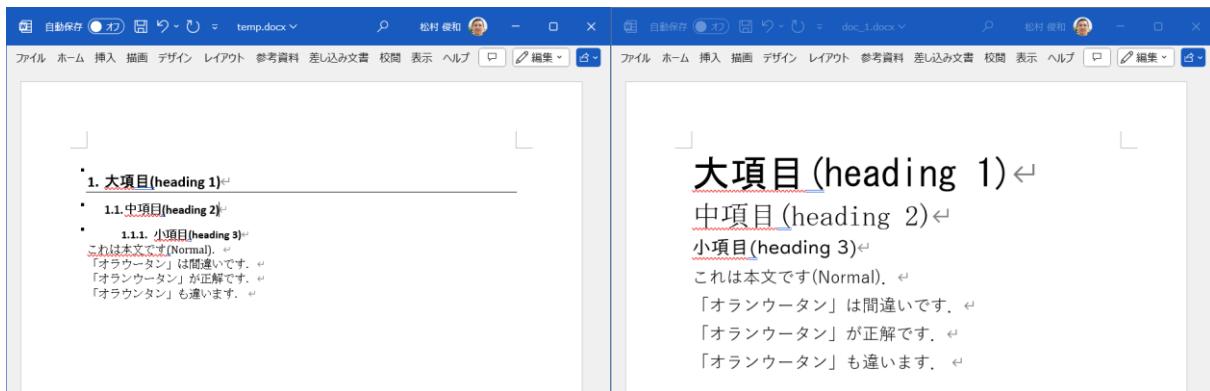


図 8.3: フォントを変更したファイル(左: 変更前, 右: 変更後)

8.7 ページ設定

ページの大きさと向き(縦・横), 余白の大きさを設定するには, `page_size()`と`page_mar()`を使います。`page_size()`はページの大きさと向きを, `page_mar()`は余白の大きさをそれぞれ指定します。`page_size()`では, `width`と`height`をインチで指定します。幅 21.0cm で高さ 29.7cm の

A4 サイズにするには, `width = 21.0/2.54, height = 29.7/2.54` とします. 縦長の向きは `orient = "portrait"` とし, 横長は `orient = "landscape"` とします. `page_mar()` は上・下・左・右の余白, ヘッダーとフッター, 緹じ代の大きさをインチで指定します.

`page_size()` と `page_mar()` で指定したものをもとに, `prop_section()` と `body_set_default_section()` で設定します.

以下では, ページサイズはそのまま横向きに, 余白は 0.4 インチ(約 1cm), ヘッダーとフッターの位置はそれぞれ約 0.5cm, 緹じ代なしに設定します(図 8.4).

コード 8.15 (word-page.R) : ページ設定

```
size <- page_size(orient = "landscape") # 横向き
mar <- 0.4                                # 1 インチ: 約 1cm
margins <- page_mar(mar, mar, mar, mar, # 順に下上右左の余白
                     mar/2, mar/2,      # ヘッダーとフッターの位置
                     0)                  # 緹じ代
ps <- prop_section(page_size = size, page_margins = margins)
doc_1 <- body_set_default_section(doc_1, value = ps)
print(x = doc_1, target = path_doc_1)
```

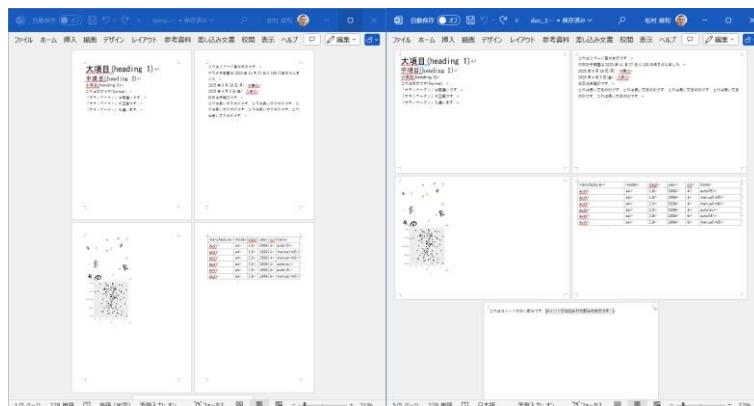


図 8.4: ページ設定を変更したファイル(左: 変更前, 右: 変更後)

ワードの既定値の設定では余白が大きすぎることがあります. 一括で余白を狭くしたい時に役立つでしょう.

8.8 文書の相互変換

ワードの文書を PDF やテキストファイルとして保存したいことがあると思います. RDCOMClient パッケージを使ってワードを操作すれば, ワードから別の形式に変換できます. また, 逆に PDF などからワードへの変換も可能です.

ワードから PDF など別形式に変換するには, RDCOMClient の `COMCreate()` によってワードを操作します. `COMCreate()` で作成したオブジェクトのメソッド(関数)を使うと, ファイルを開いたり保存したりできます.

次のコードでは、docx, PDF, XPS(XMLベースのマークアップ言語), HTML, rtf(リッチテキスト), txtを相互変換できる関数を定義します。なお、Rではディレクトリを"/"で表示します。RDCOMClientパッケージの関数では"/"はエラーになるため、normalizePath()で"\\"に変換します。

コード 8.16 (word-convert-fun.R) : ワードと各種形式との相互変換の関数

```
convert_docs <- function(path, format){  
  if (fs::path_ext(path) == format){ # 拡張子が入力と同じとき  
    return(invisible(path)) # 終了  
  }  
  format_no <- switch(format, # 拡張子ごとに番号に変換  
    docx = 16, pdf = 17,  
    xps = 19, html = 20, rtf = 23, txt = 25)  
  path <- normalizePath(path) # Windows の形式に変換  
  converted <-  
    fs::path_ext_set(path, ext = format) |> # 変換後の拡張子  
    normalizePath(mustWork = FALSE)  
  # ワードの操作  
  wordApp <- RDCOMClient::COMCreate("Word.Application")  
  wordApp[["Visible"]] <- FALSE # TRUE: ワードの可視化  
  wordApp[["DisplayAlerts"]] <- FALSE # TRUE: 警告の表示  
  doc <- wordApp[["Documents"]]$Open(path, # ファイルを開く  
    ConfirmConversions = FALSE)  
  doc$SaveAs2(converted, FileFormat = format_no) # 名前をつけて保存  
  doc$close()  
  return(converted)  
}
```

定義した関数の実行前に、RDCOMClientの呼び出します。呼び出さないとエラーが発生するためです。関数を実行するとPDFに変換できます(図8.5)。なお、PDFに変換したファイルは、手作業での変換と同様の表示のずれは発生します。

コード 8.17 (word-convert.R) : ワードと各種形式との相互変換

```
library(RDCOMClient) # 無いと関数実行時にエラーが出る  
path_pdf <- convert_docs(path_doc_1, "pdf")  
fs::path_file(path_pdf)  
## [1] "doc_1.pdf"  
# shell.exec(path_pdf)
```

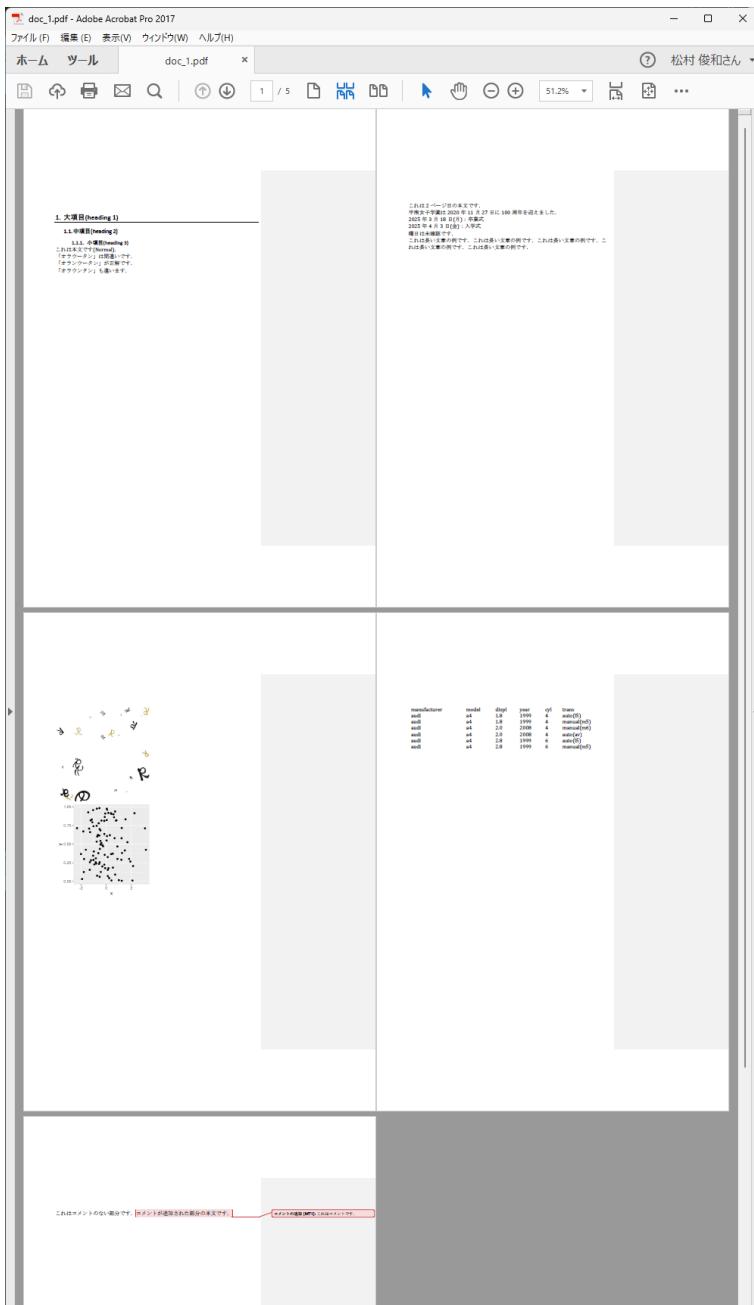


図 8.5: ワードから変換した PDF

8.9 文書の作成

`officer` でワードの文書を新規作成することも可能です。テキストファイルがあれば、それをもとにワードの文書を作成できます。タイトルなどの書式の設定、画像や表の挿入ができます。`ggplot2`(2.7節を参照)で作成した図も使えます。最初から最後までを R で作成することはないかもしれません、文書の概要を作成することはできるでしょう。

`read_docx()`を引数なしで実行すると、文書を新規作成できます。オブジェクトを表示すると、使用できる書式が `styles` で出力されます。`empty document` となっているので、空の文書だと分かります。

コード 8.18 (`word-new-docx.R`) : 文書の新規作成

```

doc_2 <- read_docx()
doc_2
## rdocx document with 1 element(s)
##
## * styles:
##      Normal          heading 1          heading 2
##      "paragraph"      "paragraph"
##      heading 3 Default Paragraph Font
##      "paragraph"      "character"
##      No List         strong
##      "numbering"     "character"
##      table_template Light List Accent 2
##      "table"          "table"
##      Titre 2 Car    Titre 3 Car
##      "character"     "character"
##      Table Caption   Table Professional
##      "paragraph"      "table"
##      toc 2           Balloon Text
##      "paragraph"      "paragraph"
##      reference_id    graphic title
##      "character"     "paragraph"
##
## * empty document

```

文書の内容としてパラグラフを追加するには、`body_add_par()`を使います。`value`に追加したい文字列を、`style`に書式を指定します。改ページを入れるには、`body_add_break()`を使います。オブジェクトを表示させると、`Content at cursor location`として最後に追加した内容が表示されます。

コード 8.19 (word-body-add-par.R) : 文書にパラグラフを追加

```

doc_2 <-
doc_2 |>
body_add_par(value = "大項目(heading 1)", style = "heading 1") |>
body_add_par(value = "中項目(heading 2)", style = "heading 2") |>
body_add_par(value = "小項目(heading 3)", style = "heading 3") |>
body_add_par(value = "これは本文です(Normal). ", style = "Normal") |>
body_add_par(value = "「オラウータン」は間違えです。", style = "Normal") |>
body_add_par(value = "「オランウータン」が正解です。", style = "Normal") |>
body_add_par(value = "「オラウンタン」も違います。", style = "Normal") |>
body_add_break() |>
body_add_par(value = "これは2ページ目の本文です", style = "Normal")
doc_2
## rdocx document with 7 element(s)
## * styles:
## (中略) 1つ前のコードと同じ
## * Content at cursor location:
##   level num_id          text style_name content_type
##   1     NA      NA これは2ページ目の本文です      Normal      paragraph

```

`docx_summary()`で文書の概要を表示すると、入力したものが入っているのが確認できます。

コード 8.20 (word-body-add-par-summary.R) : 作成中の文書の概要

```
docx_summary(doc_2) |>
  tibble::as_tibble() # 見やすくするために tibble に変換
## # A tibble: 8 × 6
##   doc_index content_type style_name text          level num_id
##       <int>      <chr>     <chr>    <chr>
## 1 1         paragraph heading 1 "大項目(heading 1)"     NA    NA
## 2 2         paragraph heading 2 "中項目(heading 2)"     NA    NA
## 3 3         paragraph heading 3 "小項目(heading 3)"     NA    NA
## 4 4         paragraph Normal    "これは本文です(Norm...
## 5 5         paragraph Normal    "「オラウータン」は...
## 6 6         paragraph Normal    "「オランウータン」...
## 7 7         paragraph Normal    "「オラウンタն」も...
## 8 8         paragraph <NA>      ""                  NA    NA
```

文書作成に body_add_par() とするのは面倒です。そこで文字列をまとめて入力する関数を作成します。関数内ではワードに対して順次文字列を追加するので、順次処理の関数 purrr::reduce() を使います(2.9 を参照)。

コード 8.21 (word-insert-text-fun.R) : 文字列をまとめて入力する関数

```
insert_text <- function(docx, str, style = "Normal"){
  docx <-
    str |> # str を順番に
    purrr::reduce(officer::body_add_par, style = style, .init = docx)
  return(docx)
}
```

定義した関数を使って文章を入力します。

コード 8.22 (word-insert-text.R) : 文字列をまとめて入力

```
text <-
  c("これは 2 ページ目の本文です。",
    "甲南女子学園は 2020 年 11 月 27 日に 100 周年を迎えました。",
    "2025 年 3 月 18 日(月)：卒業式",
    "2025 年 4 月 3 日(金)：入学式",
    "曜日は未確認です。",
    paste0(rep("これは長い文章の例です。 ", 5), collapse = ""))
  )
doc_2 <- insert_text(doc_2, text)
docx_summary(doc_2) |>
  tibble::as_tibble() |> # tibble に変換
```

```
dplyr::select(-c(content_type, level, num_id)) |> # 列を除去
tail(8) # 最後の 8 行のみ
## # A tibble: 8 × 3
##   doc_index style_name text
##       <int>      <chr>    <chr>
## 1         7 Normal   "「オラウンタン」も違います。"
## 2         8 <NA>        ""
## 3         9 Normal   "これは 2 ページ目の本文です。"
## 4        10 Normal  "甲南女子学園は 2020 年 11 月 27 日に 100 周年を迎えた。"
## 5        11 Normal  "2025 年 3 月 18 日(月) : 卒業式"
## 6        12 Normal  "2025 年 4 月 3 日(金) : 入学式"
## 7        13 Normal  "曜日は未確認です。"
## 8        14 Normal  "これは長い文章の例です。これは長い文章の例です。"
```

ワードの文章として保存してから開くと、文書が入力されていることがわかります(図 8.6).

コード 8.23 (word-doc-2-text-print.R) : ワードの保存

```
print(doc_2, target = path_doc_2)
# shell.exec(path_doc_2)
```



図 8.6: 文書を入力したワード

実際の作業では、テキストファイルを `readLines()` や `readr::read_lines()` で読み込んでから、`insert_text()` を使うとよいでしょう。

8.10 図表の追加

画像ファイルの追加には `body_add_img()` を、 `ggplot2` での作図には `body_add_gg()` を、 表には `body_add_table()` をそれぞれ使います。 `body_add_img()` はファイルのパスおよび `width(幅)` と `height(高さ)` を指定します。 幅と高さの単位はインチ(約 2.54cm)です。 `body_add_gg()` は `ggplot2` での作図オブジェクトを、 `body_add_table()` にはデータフレームを指定します。

コード 8.24 (word-boy-add-img.R) : 文書への図表の追加

```
img <- "https://matutosi.github.io/r-auto/data/r_gg.png" # 画像
gg_point <- # ggplot
  tibble::tibble(x = rnorm(100), y = runif(100)) |>
  ggplot2::ggplot(ggplot2::aes(x, y)) +
```

```

ggplot2::geom_point()
mpg_tbl <- head(mpg[,1:6])                                # 表
doc_2 <-
  doc_2 |>
  body_add_break() |>                                     # 改ページ
  body_add_img(img, width = 3, height = 3) |>            # 画像の追加
  body_add_gg(gg_point, width = 3, height = 3) |>        # ggplot の追加
  body_add_break() |>
  body_add_table(mpg_tbl)                                    # 表の追加

```

一旦保存してから開くと、図と表が入っているのがわかります(図 8.7).

コード 8.25 (word-fig-print.R) : 図と表を追加後の保存

```

print(doc_2, target = path_doc_2)
# shell.exec(path_doc_2)

```

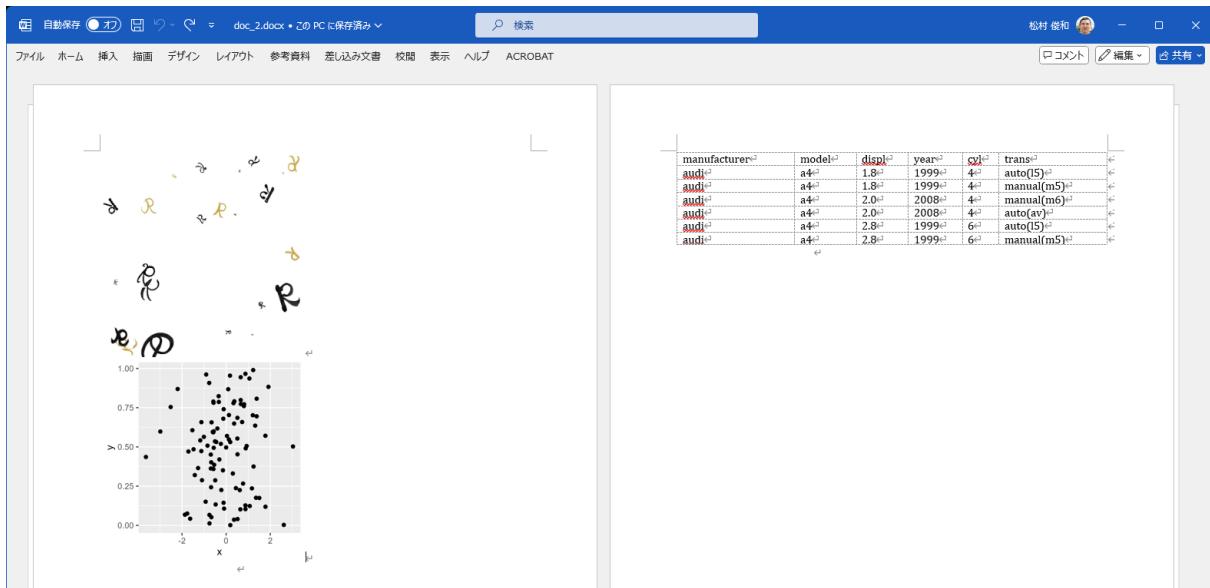


図 8.7: 図と表を入力したワード

8.11 コメントの追加と抽出

ワードにはコメントを書き込める機能があります。 `officer` では `run_comment()` でコメントを作成できます。引数 `cmt` に `block_list()` の形でコメントを指定します。コメントを追加する場所は、引数 `run` に `ftext()` で指定します。その他、コメントの著者、追加日、イニシャルを指定できます。

コード 8.26 (word-run-comment.R) : コメントの追加

```

comment <- run_comment(
  cmt = block_list("これはコメントです。"))

```

```

run = ftext("コメントが追加された部分の本文です。"),
author = "コメントの著者",
date = Sys.Date(),
initials = "MT"
)
par <- fpar("これはコメントのない部分です。", comment)
doc_2 <-
  doc_2 |>
  body_add_break() |>
  body_add_fpar(value = par, style = "Normal")

```

コード 8.27 (word-doc-2-comment-print.R) : コメントを追加後の保存

```

print(doc_2, target = path_doc_2)
# shell.exec(path_doc_2)

```

保存してから開くとコメントが入っています(図 8.8).

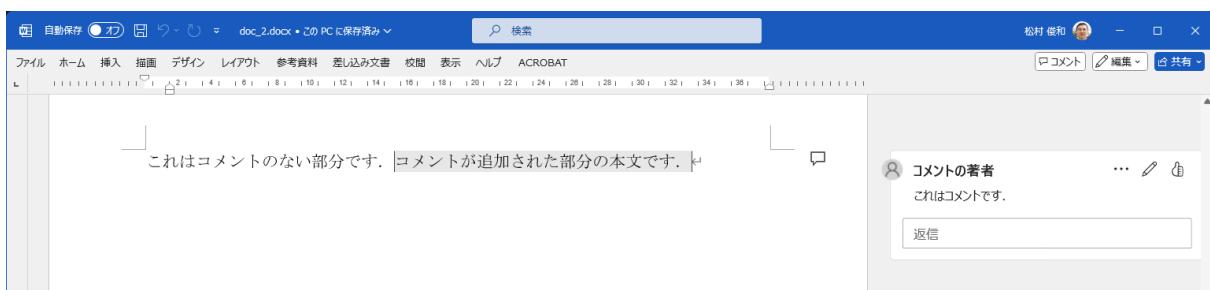


図 8.8: コメント付きの文書を入力したワード

コメントを抽出するには, `docx_comments()`を使います。データフレームとしてコメントの一覧を抽出できます。なお、ファイルが大きいと、コメントの抽出に数分以上かかることがあります。

コード 8.28 (word-docx-comment.R) : コメントの抽出

```

comment <-
  docx_comments(doc_2) |>
  tibble::as_tibble() |>
  print()
## # A tibble: 1 × 7
##   comment_id author      initials date  text  para_id commented_text
##   <chr>     <chr>      <chr>    <chr> <lis> <list>   <list>
## 1 0         コメントの著... MT       2024... <chr> <chr>   <chr> [1]>

```

取り出したコメントをエクセルとして保存するには、`openxlsx::write.xlsx()`を使います(9.4 節を参照)。今回のコメントであればそのまま保存できますが、コメントやその本文に複数の文字列が入っていて、いわゆるリスト列になっていることがあります。そのときは、そのままでは保存で

きません。そこで `tidy::unnest_longer()` でリスト列を解除してからコメントごとに結合します。

ただし、コメントに全く同じ文字列が入っているときは、貼り付けないようにするために、文字列を比較して異なるときのみ貼り付ける関数を定義します。

コード 8.29 (word-cumulative-paste-fun.R) : 文字列を比較して異なるときのみ貼り付ける関数

```
cumulative_paste <- function(x, y){  
  if(x == y){      # x と y が同じなら  
    x                #   x のまま  
  }else{           # x と y が異なれば  
    paste0(x, y) #   x と y を貼り付け  
  }  
}
```

次のコードを実行するとコメントを取り出して、列幅を整理したエクセルに保存できます(図 8.9)。

コード 8.30 (word-docx-comment-write.R) : コメントの保存

```
comment_path <- fs::path_temp("comment.xlsx")  
comment |>  
  tidy::unnest_longer(-comment_id) |>  
  dplyr::summarise(  
    .by = c(comment_id, author, initials, date), # コメントでグループ化  
    text = reduce(text, cumulative_paste),  
    commented_text = reduce(commented_text, cumulative_paste)) |>  
  openxlsx::write.xlsx(comment_path)  
wb <- openxlsx::loadWorkbook(comment_path)          # 読み込み  
openxlsx::setColWidths(wb, 1, cols = 1:7, width = "auto") # 列幅の変更  
openxlsx::saveWorkbook(wb, comment_path, overwrite = TRUE) # 保存  
# shell.exec(comment_path)
```

	A	B	C	D	E	F	G
1	comment_id	author	initials	date	text	para_id	commented_text
2	0	コメントの著者	MT	2024-06-10	これはコメントです。	NA	コメントが追加された部分の本文です。
3							
4							

図 8.9: エクセルに保存したコメント

8.12 画像の抽出

ワードのファイルに含まれる画像を単独のファイルとして保存するには次のような作業が必要です。

ワードで画像を右クリックしてから、「図として保存」を選択し、ファイル名を指定する必要があります(図 8.10)。

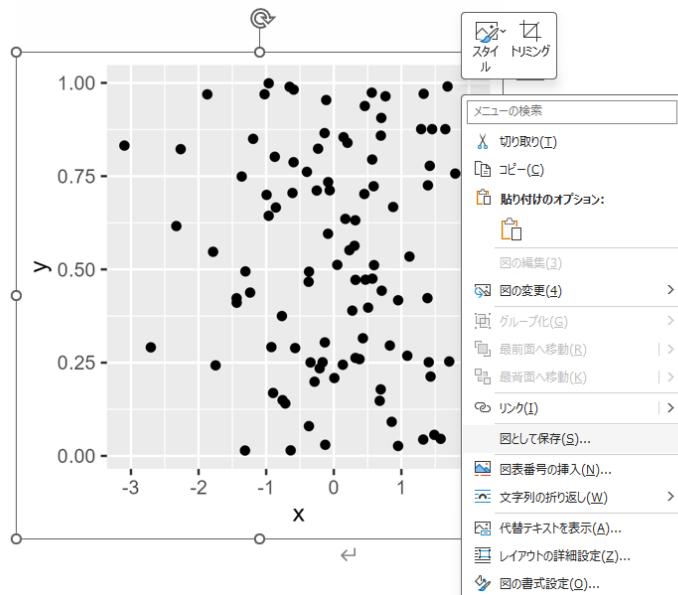


図 8.10: 手作業での画像の保存

ファイルに含まれる画像が 1 つや 2 つだけであれば、手作業でも時間はかかりません。しかし、ファイルに含まれる画像が多ければ時間がかかりますし、作業漏れもあります。

ワードファイルは文書の内容を記した複数のファイルを zip 圧縮して、拡張子を docx にしたもので(エクセルとパワーポイントも同様です)。そのためファイルを zip で解凍すれば、文章に含まれる画像ファイルを取り出すことができます。つまり以下の手順で実行可能です。

- ワードの拡張子を docx から zip に変更
- ファイルを右クリック
- 全てを展開で解凍
- 解凍してできたディレクトリにある word/media の画像を保存

doc_1.docx をもとに作業すると、word/media ディレクトリには画像含まれていることがわかります(図 8.11)。

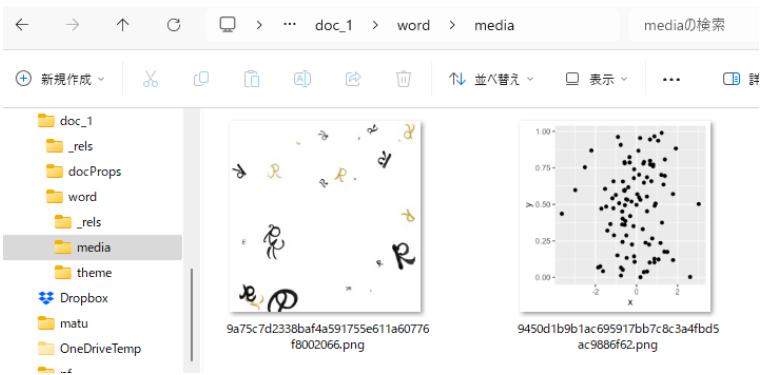


図 8.11: word/media ディレクトリの画像ファイル

この作業もファイルが少なければ問題有りませんが、多くのファイルになると手間がかかります。作業として単純なものですので、Rで関数化できます。

ファイル名の変更やファイルのコピーなどがありますので、fs パッケージ(第1章を参照)を多く使います。zip ファイルの解凍は unzip()を使います。

次の関数では、path のディレクトリの全てのワードファイルから、全ての画像を抽出します。抽出した画像のファイル名は、「ワードのファイル名_画像のファイル名」です。なお、画像のファイル名はワードが自動的に生成したもので、ワードに入れる前の画像のファイル名とは異なります。

コード 8.31 (word-extract-docx-img-fun.R) : ディレクトリ内のワードから画像を抽出する関数

```
extract_docx_imgs <- function(path) {
  docxs <- fs::dir_ls(path, regexp = "\\\\\\.docx$") # ワードの一覧
  zips <-
    docxs |>
      fs::path_file() |> # ファイル名のみ
      fs::path_ext_set("zip") |> # 拡張子を zip に変更
      fs::path_temp() # 一時ファイル
  fs::file_copy(docxs, zips, overwrite = TRUE) # docx を zip として複製
  zip_dirs <- fs::path_ext_remove(zips) # 拡張子の除去
  purrr::walk2(zips, zip_dirs,
    \x, y){ unzip(zipfile = x, exdir = y) }) # 解凍
  images <- purrr::map(zip_dirs, extract_imgs) # 画像の抽出
  images <-
    unlist(images) |>
    fs::file_move(path) # 画像の移動
  return(images)
}
```

コード 8.32 (word-extract-img-fun.R) : ディレクトリから画像を抽出する関数

```

extract_imgs <- function(zip_dir) {
  img_dir <- fs::path(zip_dir, "word/media") # 画像ディレクトリ
  if(fs::dir_exists(img_dir)){ # ディレクトリの有無の確認
    img_files <- fs::dir_ls(img_dir) # 画像ファイル
    # 変更後の画像ファイル
    img_files_new <- paste0(zip_dir, "_", fs::path_file(img_files))
    # 画像ファイルの複写
    fs::file_copy(img_files, img_files_new, overwrite = TRUE)
    return(img_files_new)
  }else{
    return(fs::path())
  }
}

```

作成した関数を実行すると、画像が抽出できます(図 8.12).

コード 8.33 (word-extract-docx-img.R) : ディレクトリ内のワードから画像を抽出

```

dir <- fs::dir_create(fs::path_temp(), "images") # ディレクトリの作成
fs::file_copy(c(path_doc_1, path_doc_2), dir) # ファイルを複写
imgs <- extract_docx_imgs(dir) # ワードから画像を抽出
fs::path_file(imgs) # 抽出した画像のファイル名
## [1] "doc_1_9a75c7d2338baf4a591755e611a60776f8002066.png"
## [2] "doc_1_bfd7286710bb064e3c259f8bc229017f1e24a821.png"
## [3] "doc_2_9a75c7d2338baf4a591755e611a60776f8002066.png"
## [4] "doc_2_bfd7286710bb064e3c259f8bc229017f1e24a821.png"
# shell.exec(dir)

```

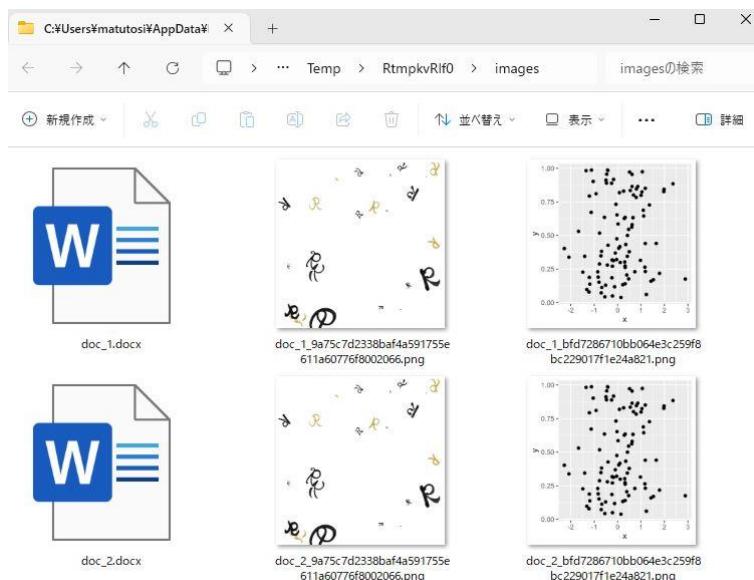


図 8.12: ワードと抽出した画像ファイル

8.13 文書内の日付の修正と更新

日付関連の関数は、第4章で定義したものを使用するので、以下のコードで読み込みます。関数の詳細は第4章をご覧ください。

コード 8.34 (word-date-fun.R) : 日付関連の関数の読み込み

```
source("https://matutosi.github.io/r-auto/R/04_date_funs.R")
```

コード 8.35 (word-update-dates.R) : ワードの文書内の日付の修正

```
text <- extract_docx_text(doc_1) # 文字列の抽出
dates_before <- # 日付の抽出
  extract_date_ish(text) |>
  unlist()
dates_after <- update_wday(dates_before) # 正しい曜日
tibble::tibble(dates_before, dates_after) # 置換文字列の一覧
## # A tibble: 3 × 2
##   dates_before      dates_after
##   <chr>            <chr>
## 1 2020 年 11 月 27 日(金) 2020 年 11 月 27 日(金)
## 2 2025 年 3 月 18 日(火) 2025 年 3 月 18 日(火)
## 3 2025 年 4 月 3 日(木) 2025 年 4 月 3 日(木)
doc_1 <-
  purrr::reduce2(.x = dates_before, .y = dates_after,
                 .f = body_replace_all_text, .init = doc_1, fixed = TRUE)
print(doc_1, path_doc_1)
# shell.exec(path_doc_1)
# for ループのとき
# for(i in 1:length(dates_before)){
#   docx_1 <- docx_1 |>
#     body_replace_all_text(dates_before[i], dates_after[i], fixed = TRUE)
# }
```

上のコードを実行すると、ワードの文書内の日付を修正できます(図 8.13)。

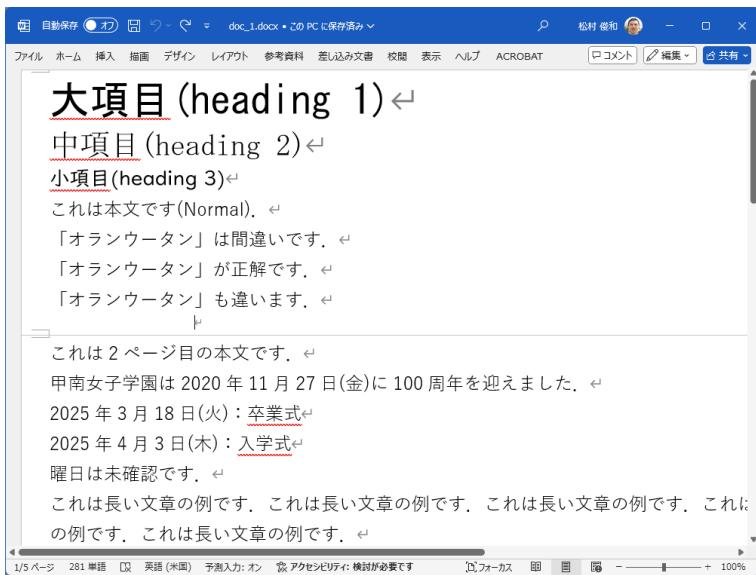


図 8.13: 日付を修正した文書

4月の第1木曜日のように、1年後の同月の同じ位置に更新するには、次のようにします。

コード 8.36 (word-dates-next-yr.R) : 日付の1年後の同じ位置への更新

```
dates_next_yr <-  
  dates_before |>  
  lubridate::ymd() |>  
  same_pos_next_yr()  
tibble::tibble(dates_before, dates_next_yr) # 置換文字列の一覧  
## # A tibble: 3 × 2  
##   dates_before      dates_next_yr  
##   <chr>            <chr>  
## 1 2020 年 11 月 27 日(金) 2021 年 11 月 26 日(金)  
## 2 2025 年 3 月 18 日(火) 2026 年 3 月 17 日(火)  
## 3 2025 年 4 月 3 日(木) 2026 年 4 月 2 日(木)  
doc_1 <-  
  purrr::reduce2(.x = dates_before, .y = dates_next_yr,  
                 .f = body_replace_all_text, .init = doc_1, fixed = TRUE)
```

9 エクセルの操作

表計算アプリのエクセルは色々なことができて便利です。毎日のように事務作業でエクセルを使う人は多いと思いますが、色々なことができるがゆえに苦労も多いです。

たとえば、列幅の設定、オートフィルタの設定、ウィンドウ枠の固定の作業を新しいファイルを作るたびにしていいでしょうか。一回ごとの作業はほんのわずかでも、何度も繰り返していれば、合計すると膨大な時間になります。また、内容を見やすくするために、罫線や背景色などを使うこともあるでしょう。その際に「セルの内容を目で確認・マウスで範囲選択・セルの書式設定を変更」という手作業が必要です。手作業が終わってから、資料の内容が変更になって、最初からやり直した経験があるかもしれません。一度だけならまだしも、何度も同じ作業を繰り返すのは正直なところ辛いです。

条件付き書式設定では、背景色を設定するだけでなく重複の確認やデータバーなど便利な機能もあります。これらを自動化できればエクセルの活用方法は広がるでしょう。

PDFへの変換も RDCOMClient パッケージで可能ですので、本章ではその方法を紹介します。`pdftools` と合わせて使えば(7.5 節を参照)、会議資料の作成に役立つでしょう。

本章では、このようなエクセルの操作を R で自動化する方法を紹介します。特定の条件に従った操作であれば、何度も同じことを簡単に実行可能です。具体的には次のことを紹介します。

- ・ エクセルのファイルの読み込みと書き込み
- ・ ピボットテーブルのような集計表の作成
- ・ セルの列幅やオートフィルタの設定
- ・ 内容に合わせた罫線の描画や強調表示
- ・ ヘッダー・フッターや印刷レイアウトの設定

これらを自動化すれば、時間短縮が可能なのはもちろんですが、目視やマウスによる操作による間違いや漏れもありません。

9.1 エクセルを操作するパッケージ

本章では、主に `openxlsx` パッケージを使用するので、インストールして呼び出しておきます。また、ファイルの読み込みと書き込みで使用する `readxl` もインストールして呼び出します。

コード 9.1 (excel-install.R) : `openxlsx` と `readxl` のインストール

```
install.packages("openxlsx")
install.packages("readxl")
```

コード 9.2 (excel-library.R) : `openxlsx` と `readxl` の呼び出し

```
library(readxl)
library(openxlsx)
```

9.2 ファイルの読み込み

ファイルの読み込みと書き込みでは、`openxlsx` 以外に `readxl` と `readr` パッケージの関数を使います。ファイルの形式や R での形式によって使用する関数が異なります(表 9.1)。`readr` は `tidyverse` に含まれているので、追加のインストールは不要です。

表 9.1: ファイルの読み込みの主な関数

ファイルの形式	R での形式	パッケージ	関数
テキスト	データフレーム	readr	<code>read_csv()</code> , <code>read_tsv()</code> , <code>read_delim()</code>
エクセル	データフレーム	readxl	<code>read_xls()</code> , <code>read_xlsx()</code> , <code>read_excel()</code>
エクセル	データフレーム	openxlsx	<code>readWorkbook()</code>
エクセル	ワークブック	openxlsx	<code>loadWorkbook()</code>

`csv` や `tsv` などのテキストファイルをデータフレームとして読み込むには、`read_csv()` や `read_tsv()` を使います。区切り文字を指定して読み込むときには、`readr::read_delim()` を使います。

コード 9.3 (excel-readr-read.R) : csv などの読み込み

```
file_csv <- readr::readr_example("mtcars.csv")
readr::read_csv(file_csv, show_col_types = FALSE) # csv(カンマ区切り)
# readr::read_tsv(ファイル名)                      # tsv(タブ区切り)
# readr::read_delim(ファイル名, delim = ",")       # delim: 区切り文字
```

エクセル形式のファイルを読み込むにはいくつかの方法があります。読み込んだ後の操作によって使用するパッケージや関数が異なります。エクセルの個々のシートの内容をデータフレームとして読み込むときには、`readxl` の `read_xls()`, `read_xlsx()`, `read_excel()` や `openxlsx` の `readWorkbook()` を使います。

`read_xls()` と `read_xlsx()` はそれぞれ、`xls` のみと `xlsx` のみの読み込みが可能です。そのため、両方に対応している `read_excel()` を使うのが便利です。`sheet` にはシート番号かシート名を指定します。`sheet` の指定がなければ 1 つ目のシートのデータを読み込みます。

コード 9.4 (excel-readxl-read.R) : `read_excel()` によるデータフレームとしての読み込み

```
path <- readxl::readxl_example("datasets.xlsx")
iris <- readxl::read_excel(path, sheet = "iris") # シート名
iris |>
```

```

head(3)
## # A tibble: 3 × 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##       <dbl>      <dbl>      <dbl>      <dbl> <chr>
## 1         5.1        3.5        1.4       0.2  setosa
## 2         4.9        3.0        1.4       0.2  setosa
## 3         4.7        3.2        1.3       0.2  setosa
mtcars <- readxl::read_excel(path, sheet = 2) # 番号

```

readWorkbook()を使うと read_xlsx()と同様にエクセルのファイルをデータフレームとして読み込みます。

コード 9.5 (excel-read.R) : readWorkbook()によるデータフレームとしての読み込み

```

readWorkbook(path) |> head(3) # データフレームとしての読み込み
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1         5.1        3.5        1.4       0.2  setosa
## 2         4.9        3.0        1.4       0.2  setosa
## 3         4.7        3.2        1.3       0.2  setosa
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1         5.1        3.5        1.4       0.2  setosa
## 2         4.9        3.0        1.4       0.2  setosa
## 3         4.7        3.2        1.3       0.2  setosa

```

エクセルのファイルをワークブックとして読み込むときには openxlsx パッケージの loadWorkbook()を使います。ワークブックとして読み込むと、列幅や罫線を引くなどの作業ができます。

コード 9.6 (excel-load-wb.R) : ワークブックとしての読み込み

```

wb <- loadWorkbook(path) # ワークブックとしての読み込み
wb
## A Workbook object. # 空白行を省略して表示
## Worksheets:
##  Sheet 1: "iris"
##  Sheet 2: "mtcars"
##  Sheet 3: "chickwts"
##  Sheet 4: "quakes"
##  Worksheet write order: 1, 2, 3, 4
##  Active Sheet 1: "iris"
##          Position: 1

```

openxlsx や readxl の関数は、パスワード付きのエクセルのファイルには対応していません。パスワード付きのファイルを開くには、excel.link パッケージの xl.read.file()を使います。excel.link は CRAN からインストール可能です。なお、エクセルがインストールされている必要があります。また、日本語は NA に変換されるようです。

コード 9.7 (excel-excel.R) : excel.link パッケージのインストールと呼び出し

```
install.packages("excel.link")
library(readxl)
```

ファイル名とパスワードを指定することで、パスワード付きのエクセルのデータを読み込むことができます。

コード 9.8 (excel-read-with-password.R) : パスワード付きのエクセルファイルを開く疑似コード

```
excel.link::xl.read.file("ファイル名.xlsx", password = "パスワード")
```

同じ書式でデータが複数のシートに入っているとき、一気にデータを読み込むと便利です。そのための関数は、コード 2.3 に `read_all_sheets()`としてまとめているので参考にしてください。

9.3 ウェブからのデータの読み込み

【追加】 Google ドライブと OneDrive からのデータのダウンロード(擬似コード)を追加しました。

ウェブのフォームを使って情報収集することができます。GoogleForm のときは Google ドライブに、MicrosoftForms のときは OneDrive にそれぞれ結果のデータが保存されます。1回だけであれば、手作業でファイルをダウンロードしても問題ありません。しかし、途中経過を集計するときや、繰り返して実施するときには手作業でのダウンロードは面倒です。そのようなときには、以下のようにデータをダウンロードすると手間が削減できます。

コード 9.9 (excel-read-goolgedrive.R) : Google ドライブからのファイルのダウンロード(擬似コード)

```
install.packages("googledrive")
library(googledrive)
googledrive::drive_auth("YOURNAME@gmail.com") # 認証画面でパスワード等を入力
sheet <- googledrive::drive_find(pattern = "検索文字列", type = "spreadsheet")
path <- "DIRECORY/FILE_NAME.csv"
googledrive::drive_download(sheet$name, path = path, type = "csv", overwrite = TRUE)
# 上書きするとき
```

コード 9.10 (excel-read-onedrive.R) : OneDrive からのファイルのダウンロード(擬似コード)

```
install.packages("Microsoft365R")
library(Microsoft365R)
odb <- Microsoft365R::get_business_onedrive(tenant = "YOUR_COMPANY.0R.JP") # 認証
odb$list_files()
```

```

src <- "FILE_NAME"
dest <- "DIRECORY/FILE_NAME" # 認証画面でパスワード等を入力
odbc$download_file(src = src, dest = dest, overwrite = TRUE) # 上書きするとき

```

9.4 ファイルの書き込み

ファイルの書き込みでは、ファイルの形式やオブジェクトの形式によって使用する関数が異なります(9.2)。データフレームを csv や tsv などのテキストファイルとして書き込むには、それぞれ write_csv() や write_tsv() を使います。区切り文字を指定して書き込むときには、readr::write_delim() を使います。

表 9.2: ファイルの書き込みの主な関数

オブジェクトの形式	ファイルの形式	パッケージ	関数
データフレーム	テキスト	readr	write_csv(), write_tsv(), write_delim()
データフレーム	エクセル	openxlsx	write.xlsx()
ワークブック	エクセル	openxlsx	saveWorkbook()

コード 9.11 (excel-write-txt.R) : csv などの書き込み

```

readr::write_csv(mtcars, "mtcars.csv")           # csv(カンマ区切り)
readr::write_tsv(mtcars, "mtcars.tsv")          # tsv(タブ区切り)
readr::write_delim(iris, "iris.txt", delim = ";") # delim: 区切り文字
ls("package:readr") |>                      # 他にも色々とある
  stringr::str_subset("write") # 詳細はヘルプ参照

```

データフレームをエクセル形式で書き込むには、openxlsx パッケージの write.xlsx() を使います。

コード 9.12 (excel-write-df.R) : データフレームのエクセル形式での書き込み

```

file_wb <- fs::path_temp("workbook.xlsx")
write.xlsx(iris, file_wb)

```

データフレームを split() で分割(2.8 節を参照)したときは、分割したものがそれぞれ別のシートに書き込まれます。たとえば、iris のデータを Species で分割したときには、setosa, versicolor, virginica の 3 つのシートが自動的につくられて、データもそれぞれのシートに書き込まれます(図 9.1)。

コード 9.13 (excel-write-df-split.R) : 分割したデータフレームのエクセルのシートごとの書き込み

```
iris |>
  split(iris$Species) |>
  write.xlsx(file_wb)
```

	A	B	C	D	E
1	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
2	5.1	3.5	1.4	0.2	setosa
3	4.9	3	1.4	0.2	setosa
4	4.7	3.2	1.3	0.2	setosa
5	4.6	3.1	1.5	0.2	setosa
6	5	3.6	1.4	0.2	setosa
7	5.4	3.9	1.7	0.4	setosa
8	4.6	3.4	1.4	0.3	setosa
9	5	3.4	1.5	0.2	setosa
10	4.4	2.9	1.4	0.2	setosa
11	4.9	3.1	1.5	0.1	setosa
12	5.4	3.7	1.5	0.2	setosa
13	4.8	3.4	1.6	0.2	setosa
14	5.0	3.0	1.4	0.1	setosa

図 9.1: 分割したデータフレームを書き込んだエクセルのシート

読み込んだエクセルのワークブックを、ワークブックそのものとして書き込むときには `saveWorkbook()` を使います。

コード 9.14 (excel-write.R) : ワークブックの書き込み

```
saveWorkbook(wb, file_wb)
```

【注意】 `saveWorkbook()` の第 1 引数に指定できるのは、ワークブックのオブジェクトのみで、データフレームを指定するとエラーになります。データフレームをエクセル形式で書き込むときは、`write.xlsx()` を使ってください。

9.5 ピボットテーブル

エクセルには集計のための機能としてピボットテーブルがあります。R では同様の機能を提供するパッケージとして、`pivottabler`, `pivotea`, `tidyr` があります。ここでは `pivottabler` と `pivotea` の関数について説明します。`tidyr` の関数である `pivot_longer()` と `pivot_wider()` は 2.4 節を参照してください。

コード 9.15 (excel-pivot-packages.R) : パッケージのインストール

```
install.packages("pivottabler")
install.packages("pivotea")
```

コード 9.16 (excel-pivot-library.R) : パッケージの呼び出し

```
library(pivottabler)
library(pivotea)
```

エクセルのピボットテーブルと同様に、区分ごとのデータの個数・平均・合計などを計算するときには、`pivottabler::qpvt()`を使用します。`rows` と `columns` には、それぞれ行と列に配置する項目を文字列で、`calculations` には計算させる内容を指定します。次のコードのでは、`rows = c("=", "color")`という特殊な方法で行(`rowss`)を指定しています。このうち、"="は`calculations` の計算結果を行に並べて表示させる指定です。"color"の前に指定しており、`price` と `n` の区別があって、その中で `color` を区分します。`calculations` では、"price" = "`mean(price) |> round()`"で `price` の平均値を四捨五入して `price` という項目で表示させます。`n()` はデータの個数の計算です。

コード 9.17 (excel-pivot-qpvt.R) : ピボットテーブルの作成

```
head(diamonds)
## # A tibble: 6 × 10
##   carat cut      color clarity depth table price     x     y     z
##   <dbl> <ord>    <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal    E     SI2      61.5    55    326  3.95  3.98  2.43
## 2  0.21 Premium  E     SI1      59.8    61    326  3.89  3.84  2.31
## 3  0.23 Good     E     VS1      56.9    65    327  4.05  4.07  2.31
## 4  0.29 Premium  I     VS2      62.4    58    334  4.2   4.23  2.63
## 5  0.31 Good     J     SI2      63.3    58    335  4.34  4.35  2.75
## 6  0.24 Very Good J     VVS2     62.8    57    336  3.94  3.96  2.48
pt_diamonds <-
  pivottabler::qpvt(diamonds,
  rows = c( "=", "color" ), # "=" : 結果(price, n)の表示場所・順序の指定を使う
  columns = "cut",
  calculations = c("price" = "mean(price) |> round()", "n" = "n()"))
pt_diamonds
## # A tibble: 10 × 9
##   cut      Fair  Good  Very Good Premium Ideal  Total
##   <ord>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 D        4291  3405  3470  3631  2629  3170
## 2 E        3682  3424  3215  3539  2598  3077
## 3 F        3827  3496  3779  4325  3375  3725
## 4 G        4239  4123  3873  4501  3721  3999
## 5 H        5136  4276  4535  5217  3889  4487
## 6 I        4685  5079  5256  5946  4452  5092
## 7 J        4976  4574  5104  6295  4918  5324
## 8 Total    4359  3929  3982  4584  3458  3933
## 9 n        D      163   662   1513  1603  2834  6775
## 10 F       312   909   2164  2331  3826  9542
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 11 G      314   871   2299  2924  4884  11292
## 12 H      303   702   1824  2360  3115  8304
## 13 I      175   522   1204  1428  2093  5422
## 14 J      119   307    678   808   896   2808
## 15 Total  1610  4906  12082 13791 21551 53940
```

《Tips》 行と列や `row` と `column` で、どちらが縦なのかどちらが横なのか分からなくなることはないでしょうか。著者は次のように覚えています。行と列では漢字の右側を見ます。行は横向きの二重線があり、列は縦向きの二重線があります。ですので、行は横向きの並びで、列は縦向き

の並びです。rowとcolumnについては、行とrowは発音が「ぎょう」と「ろう」と似ているので同じものであると覚えるのが良いかもしれません。

ピボットテーブルの結果はデータフレームではないので、そのままでは書き込まれません。ピボットテーブルの結果、ここではpt_diamondsのオブジェクトに含まれている関数asDataFrame()を使ってデータフレームに変換します。その後、tidyverseの関数(tibbleは2.3節を参照、tidyrは2.4節を参照)でさらに変換し、最後に、write_tsv()でtsvとして書き込みます。Windowsでは、shell.exec()で関連付けしたアプリでファイルを開くことができます(5.2節を参照)。

コード9.18 (excel-pivot-save.R) : ピボットテーブルの書き込み

```
df_diamonds <-  
  pt_diamonds$asDataFrame() |>  
  tibble::rownames_to_column("color") |>          # 行名を列に  
  tidyrr::separate(color, into = c("calc", "color")) # 結果と色を別の列に  
file_diamonds <- fs::path_temp("df_diamonds.tsv")  
readr::write_tsv(df_diamonds, file_diamonds)  
# shell.exec(file_diamonds) # 関連付けアプリで開く
```

また、ピボットテーブル中の関数renderPivot()を使うと、RStudioではビューアに、Rでは既定のブラウザに表示されます(図9.2)。

コード9.19 (excel-pivot-show.R) : ピボットテーブルの表示

```
pt_diamonds$renderPivot() # ビューア(RStudio)やブラウザ(R)で表示
```

	Fair	Good	Very Good	Premium	Ideal	Total	
price	D	4291	3405	3470	3631	2629	3170
n	E	3682	3424	3215	3539	2598	3077
carat	F	3827	3496	3779	4325	3375	3725
clarity	G	4239	4123	3873	4501	3721	3999
color	H	5136	4276	4535	5217	3889	4487
cut	I	4685	5079	5256	5946	4452	5092
depth	J	4976	4574	5104	6295	4918	5324
table	Total	4359	3929	3982	4584	3458	3933
price	D	163	662	1513	1603	2834	6775
n	E	224	933	2400	2337	3903	9797
carat	F	312	909	2164	2331	3826	9542
clarity	G	314	871	2299	2924	4884	11292
color	H	303	702	1824	2360	3115	8304
cut	I	175	522	1204	1428	2093	5422
depth	J	119	307	678	808	896	2808
table	Total	1610	4906	12082	13791	21551	53940

図9.2: ピボットテーブルのRStudioでの表示

数値の合計や個数のような計算結果ではなく、セルの中に文字列そのものを表示させるとときには `pivotea::pivot()` を使います。具体的には、授業の時間割のように、セルの中に授業科目名や教員名を入れるときです。また、顧客名簿の整理、イベントでの役割分担、参加者名簿の作成のように文字列を整理したいことがあります。そのようなときに `pivot()` は活用できます。

【追加】 上のように、社会人読者向けに対応しそうな状況を少しだけ追加しました。

次のコードでは、大学の時間割データを使います。`subject(科目)` の文字列が長いので、`stringr::str_sub()` で最初と最後の各 2 文字の合計 4 文字に短縮します(3.6 節を参照)。また、`wday(曜日)` はそのままでは月・火…とは並びませんので、見た目は悪いですが 1 や 2 などを追加します。本来は因子(`factor`)を使うべきところですが、作業用としてはこれでも問題ないでしょう。

`row` 行に表示する "grade"(学年)と "hour"(時限)を、`col` に列に表示する "wday"(曜日)を、`val` にセルに表示する値として "subj" を指定しています。さらに、`split` で "semester"(学期)を指定して全体を分割します。

コード 9.20 (excel-pivot-pivotea.R) : 文字列の入った表の作成

```
url <- "https://matutosi.github.io/r-auto/data/timetable.csv"
csv <- fs::path_temp("timetable.csv")
curl::curl_download(url, csv) # url から PDF をダウンロード
syllabus <- readr::read_csv(csv, show_col_types = FALSE) |>
  dplyr::mutate(subj = paste0(stringr::str_sub(subject, 1, 2), stringr::str_sub(subject,
  -2, -1))) |>
  dplyr::mutate(wday = stringr::str_replace(wday, "月", "1 月")) |>
  dplyr::mutate(wday = stringr::str_replace(wday, "火", "2 火")) |>
  dplyr::mutate(wday = stringr::str_replace(wday, "水", "3 水")) |>
  dplyr::mutate(wday = stringr::str_replace(wday, "木", "4 木")) |>
  dplyr::mutate(wday = stringr::str_replace(wday, "金", "5 金"))
timetable <- syllabus |>
  pivotea::pivot(row = c("grade", "hour"), col = "wday", val = c("subj",
  "teacher")), split = "semester")
head(timetable[[1]])
file_timetable <- fs::path_temp("timetable.xlsx")
write.xlsx(timetable, file_timetable) # シート別に書き込み
# shell.exec(file_timetable)
```

書き込んだファイルを開くとセルの中に科目名が文字列として入っているのが確認できます(図 9.3)。

	A	B	C	D	E	F	G	H
1	semester	grade	hour	1月	2火	3水	4木	5金
2	first		1	1 健康習A_藤英語話A_中ハウグ論_村上			健康習A_橋本	
3	first		1		1 健康習A_橋本		健康習C_伊	
4	first		1			生態入門_森本	健康習C_伊	
5	first		1	2 健康習A_山情報タA_足立		化学基礎_英語語		
6	first		1	2 健康習A_藤井			健康習A_高橋	
7	first		1	3 経済概論_高橋	健康習C_伊	化学基礎_生涯ツ		
8	first		1	3 健康習C_山田	生物基礎_	健康習C_松本		
9	first		1	3 食品品学_西村		健康習C_藤井		
10	first		1	4 健康習A_山生活習A_橋	健康習A_伊	健康習A_松本		
11	first		1	4 生活習A_高生物基礎_		健康習C_藤井		
12	first		1	4 生活習A_山下		生活入門_林		
13	first		1	4 生活習A_森本				
14	first		1	4 生活習A_村上				
15	first		1	4 生活習A_林				
16	first		1	5 生命入門_小林				
17	first		2	1 カラ実習_ハウ・B_藤 ファス論_高住生ン論_ト	レ科:			
18	first		2	1		環境イ		
19	first		2	2 カラ実習_ハウ・B_藤 フート論_山口		トレギ		
20	first		2	3 生涯科学_衣生ン論_山下		食生実:		
21	first		2	3 服飾化論_山下				
22	first		2	4 地球境論_小林		食生実:		
23	first		3	1 食品倫論_食品工学_服飾実習_林		フート!		
24	first		3	2 社会障論_高橋	服飾実習_ファン論_オレク技			
25	first		3	2		生活演習_森本		
26	first		3	3 イント論_藤原		生活習A_橋伴居居!		

図 9.3: 文字列の入った表の作成をエクセルで表示

9.6 列幅の設定

エクセルのワークブックの列幅を変更するには、`setColWidths()`を使います。 `wb` にワークブックのオブジェクト、`sheet` にシート番号またはシート名、`cols` に列幅を設定する列番号、`width` には列幅を指定します。`width = 6` のように 1つだけ指定すると、全ての列幅が 6 になります。`cols` の列番号と同じ長さであれば、各列の幅をそれぞれ設定できます。たとえば、`cols = 1:3`、`width = c(2, 4, 6)` とすると、A 列は 2、B 列は 4、C 列は 6 の幅になります。また、`width = "auto"` とすると、自動的に各列の最大幅に設定されます(図 9.4)。列幅が分からぬときは `width = "auto"` とするのが良いでしょう。ただし、フォント等の関係でエクセルでの操作のように幅がピッタリとはならないことがあります。

コード 9.21 (excel-width-sheet1.R) : 列幅の設定

```
wb <- loadWorkbook(file_timetable) # ワークブック読み込み
setColWidths(wb, 1, cols = 1:10, width = "auto") # 列幅の変更
saveWorkbook(wb, file_timetable, overwrite = TRUE) # ワークブックの書き込み
```

A	B	C	D	E	F	G
1	semester	grade	hour	1月	2火	3水
2	first	1	1	健康習A_藤井	英語話A_中村	4木
3	first	1	1			ハウグ論_村上
4	first	1	1			健康習A_橋本
5	first	1	2	健康習A_山田	情報タA_足立	生態入門_森本
6	first	1	2	健康習A_藤井		化学基礎
7	first	1	3	経済概論_高橋		健康習C_伊藤
8	first	1	3	健康習C_山田		生物基礎_小林
9	first	1	3	食品品学_西村		健康習C_
10	first	1	4	健康習A_山田	生活習A_橋本	健康習A_伊藤
11	first	1	4		生活習A_高橋	生物基礎_小林
12	first	1	4		生活習A_山下	健康習C_
13	first	1	4		生活習A_森本	生活入門
14	first	1	4		生活習A_村上	
15	first	1	4		生活習A_林	
16	first	1	5	生命入門_小林		
17	first	2	1	カラ実習_林	ハウ・B_藤田	ファス論_高橋
18	first	2	1			住生ン論
19	first	2	2	カラ実習_林	ハウ・B_藤田	フート論_山口
20	first	2	3	生涯科学_藤井	衣生ン論_山下	
21	first	2	3	服飾化論_山下		
22	first	2	4	地球境論_小林		
23	first	3	1	食品倫論_西村	食品工学_西村	服飾実習_林
24	first	3	2	社会障論_高橋		ファン論
25	first	3	2			生活演習
26	first	3	3		イント論_藤原	生活習A

図 9.4: 自動で列幅を設定したシート

上のコードでは、1つのシートだけで幅を自動的に設定しましたが、1シートだけでなく全シートで実行したいものです。同じ作業の自動的な繰り返しは、プログラムの得意分野です。コードを書く前に、全体の流れを確認します。ここでは次の順序で作業すれば良さそうです。

- ・ ワークブックの読み込み
- ・ シートごとに
 - データのある列番号の取得
 - 列幅の自動設定
- ・ ワークブックの書き込み

次のコードでは、for ループを使ってシートごとに列番号を取得して、cols に代入します。ループ以外は1つのシートでの設定と同じです。

コード 9.22 (excel-width-fun.R) : 列幅を変更する関数

```
set_col_width_auto <- function(wb_path){ # wb_path: ワークブックのパス(文字列)
  wb <- openxlsx::loadWorkbook(wb_path) # 読込み
  for(sheet in sheets(wb)){ # シートごと
    cols <- # 列番号
    openxlsx::readWorkbook(wb, sheet) |>
      ncol() |>
      seq() # seq(5)で1:5と同じ
    openxlsx::setColWidths(wb, sheet, cols, width = "auto") # 列幅の設定
  }
  openxlsx::saveWorkbook(wb, wb_path, overwrite = TRUE) # 書き込み
}
```

【追加】複数ワークブックの説明を追加しました。

次のコードを実行すると、全てのシートで列幅を自動で設定します。今回のワークブックはシートが5つだけですが、もっと多くのシートの場合は自動化でかなりの労力が削減できます。さらに、ディレクトリ内の全てのワークブックについて列幅を自動的に調整することも可能です。その場合は、`fs::dir_ls(regexp = "\\.xlsx")`(1.4)でディレクトリ内のエクセルファイルの一覧を取得してから、`for` ループで処理します。

コード 9.23 (excel-width-all.R) : 列幅の変更

```
set_col_width_auto(file_timetable) # 関数実行
```

9.7 オートフィルタの設定

エクセルのシートにオートフィルタを設定するには、`addFilter()`を使います(図 9.5)。基本的な使い方は、列幅を設定する `setColWidths()`と同じです。

コード 9.24 (excel-autofilter.R) : オートフィルタの設定

```
wb <- loadWorkbook(file_timetable)
addFilter(wb, sheet = 1, rows = 1, cols = 1:10)
saveWorkbook(wb, file_timetable, overwrite = TRUE) # ワークブックの書き込み
```

	A	B	C	D	E	F	
1	semes	gra	hc	1月	2火	3水	4木
2	first	1	1	健康習A_藤井	英語話A_中村	ハウグ論_村上	
3	first	1	1			健康習A_橋本	
4	first	1	1			生態入門_森本	
5	first	1	2	健康習A_山田	情報タA_足立		化学基礎
6	first	1	2	健康習A_藤井			
7	first	1	3	経済概論_高橋		健康習C_伊藤	化学基礎
8	first	1	3	健康習C_山田		生物基礎_小林	健康習C_
9	first	1	3	食品品学_西村			健康習C_
10	first	1	4	健康習A_山田	生活習A_橋本	健康習A_伊藤	健康習A_
11	first	1	4		生活習A_高橋	生物基礎_小林	健康習C_
12	first	1	4		生活習A_山下		生活入門
13	first	1	4		生活習A_森本		
14	first	1	4		生活習A_村上		
15	first	1	4		生活習A_林		
16	first	1	5	生命入門_小林			
17	first	2	1	カラ実習_林	ハウ・B_藤田	ファス論_高橋	住生ン論
18	first	2	1				
19	first	2	2	カラ実習_林	ハウ・B_藤田	フート論_山口	
20	first	2	3	生涯科学_藤井	衣生ン論_山下		
21	first	2	3	服飾化論_山下			
22	first	2	4	地球境論_小林			
23	first	3	1	食品価論_西村	食品工学_西村	服飾実習_林	
24	first	3	2	社会障論_高橋		服飾実習_林	ファン論
25	first	3	2				生活演習
26	first	3	3		イント論_藤原		生活習A

図 9.5: オートフィルタを設定したシート

自動的な列幅の設定と同様に、`for` ループを使ったコードを書けば全シートに対してオートフィルタを設定可能です。ただし、`for` ループのコードは長くて可読性が低いので、`purrr::map()`を使います。まず、ワークブックの全シートに対して指定した関数を実行する関数を作ります。

コード 9.25 (excel-autofilter-map-fun.R) : 全シートに同じ関数を実行する関数

```
map_wb <- function(wb, fun, ...){  
  res <-  
    openxlsx::sheets(wb) |>      # シート名を取得  
    purrr::map(fun, wb = wb, ...) # fun(wb = wb, sheet = sheet)のように受け取る  
  return(invisible(res)) # 非表示で返す  
}
```

《Tips》 map_wb()の引数の最後の...は、任意の引数を受け取るためのものです。...で受け取った引数を purrr::map(...)に渡すことができます。map_wb()は fun に関数を引数として受け取りますが、受け取った関数の引数は場合によって異なります。そのため、引数をあらかじめ決めることができませんので、...としておくと様々な引数を受け取ることができます。

addFilter()と setColWidths()をそのまま使っても良いのですが、コードがちょっと長くなりそうです。そのため、短くするための糖衣関数(ラッパー関数)を定義します。シートごとの列数を取得する関数も合わせてつくります。

コード 9.26 (excel-autofilter-wrapper-fun.R) : コードを簡潔にするための糖衣関数

```
# addFilter()の糖衣関数  
add_filter <- function(wb, sheet, rows = 1){  
  cols <- cols_wb_sheet(wb, sheet)  
  openxlsx::addFilter(wb, sheet, rows = rows, cols = cols)  
}  
  
# setColWidths()の糖衣関数  
set_col_width <- function(wb, sheet, width = "auto", ...){  
  cols <- cols_wb_sheet(wb, sheet)  
  openxlsx::setColWidths(wb, sheet, cols = cols, width = width, ...)  
}  
  
# wb と sheet での列数を取得  
cols_wb_sheet <- function(wb, sheet){  
  openxlsx::readWorkbook(wb, sheet) |>  
  ncol() |> seq()  
}
```

これで材料が揃いました。作成した関数を実行すると全てのシートでオートフィルタが設定できることが確認できます(図 9.6)。

コード 9.27 (excel-autofilter-fun.R) : 全シートでのオートフィルタと列幅の設定

```
wb <- loadWorkbook(file_timetable)  
map_wb(wb, add_filter)  
map_wb(wb, set_col_width)  
saveWorkbook(wb, file_timetable, overwrite = TRUE)
```

	A	B	C	D
1	semes	gra	hc	1月
2	first	1	1	健康習A_藤井
3	first	1	1	
4	first	1	1	
5	first	1	2	健康習A_山田
6	first	1	2	健康習A_藤井
7	first	1	3	経済概論_高橋
8	first	1	3	健康習C_山田
9	first	1	3	食品品学_西村
10	first	1	4	健康習A_山田
11	first	1	4	
12	first	1	4	
13	first	1	4	
14	first	1	4	

	A	B	C	D
1	semes	gra	hc	1月
2	second	1	1	健康習D_藤井
3	second	1	1	
4	second	1	1	
5	second	1	2	健康習B_藤井
6	second	1	2	生活済学_高橋
7	second	1	3	健康習D_山田
8	second	1	3	生化基礎_小林
9	second	1	3	
10	second	1	4	健康習B_山田
11	second	1	4	生命入門_小林
12	second	1	4	
13	second	1	4	
14	second	1	4	

	A	B	C	D
1	semes	gra	hc	他
2	通年	3	9	レク実習_橋本
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				

図 9.6: オートフィルタを全てのシートに自動で設定

《Tips》 ここでは、完成した関数をいきなり実行しています。最初からうまくいくことはほとんどありませんので、自分でプログラムを書くときには少しずつ動作確認してください。

9.8 ウィンドウ枠の固定

エクセルでは、スクロールすると行名や列名が見えなくなります。ウィンドウ枠を固定すると行名や列名が常に表示されます(図 9.7)。ウィンドウ枠の固定には、freezePane()を使います。これまでと同様に、ワークブック名とシート名を指定します。1行目あるいは1列目のみを固定したいときは、それぞれ firstRow = TRUE あるいは firstCol = TRUE と指定します。また、固定しない最初の行や列を指定することも可能で、firstActiveRow = 5, firstActiveCol = 3 とすると、5行目と3列目以降が動く行と列になり、4行目までと2列目までが固定されます。

A	F	G	H	I	J
1	semes	3水	4木	5金	
11	first	生物基礎_小林	健康習C_藤井		
12	first		生活入門_林		
13	first				
14	first				
15	first				
16	first				
17	first	ファス論_高橋	住生ン論_村上	トレ科学_清水	
18	first			環境イ論_森本	
19	first	フート論_山口		トレグ論_清水	
20	first			食生実習_山口	
21	first				
22	first			食生実習_山口	
23	first	服飾実習_林		フート論_大西	
24	first	服飾実習_林	ファン論_林	レク技A_橋本	
25	first		生活演習_森本		
26	first		生活習A_橋本		
27	first		生活習A_高橋		
28	first		生活習A_山下		
29	first		生活習A_山口		
30	first		生活習A_森本		
31	first		生活習A_西村		
32	first		生活習A_村上		
33	first		生活習A_藤原		
34	first		生活習A_林		
35	first				

図 9.7: ウィンドウ枠を固定したシート

コード 9.28 (excel-freezepanel.R) : 個別のシートでのウィンドウ枠の固定

```
freezePane(wb, 1, firstRow = TRUE, firstCol = TRUE) # 1行目と1列目を固定
```

オートフィルタの設定と同様に、全てのシートで1行目と1列目を固定できるようにする関数を定義します。

コード 9.29 (excel-freezepanel-fun.R) : ウィンドウ枠を固定する関数

```
# freezePane()の糖衣関数
freeze_pane <- function(wb, sheet){
  openxlsx::freezePane(wb, sheet, firstRow = TRUE, firstCol = TRUE)
}
```

関数が定義できたので、全てのシートにウィンドウ枠を固定します。loadWorkbook()で一旦ワークブックを読み込み直してから、map_wb()で全シートのウィンドウ枠を固定します。

コード 9.30 (excel-freezepanel-all.R) : 全てのシートでのウィンドウ枠の固定

```
wb <- loadWorkbook(file_timetable)
map_wb(wb, freeze_pane)
saveWorkbook(wb, file_timetable, overwrite = TRUE)
```

なお、本来はloadWorkbook()でワークブックを再度読み込まなくてもウィンドウ枠の固定ができるほしいところです。しかし、openxlsxのバージョン4.2.5.2の仕様では、再読み込みをしないとwarningが出るとともに、データのある行が非表示になりますので、ご注意ください。

9.9 罫線

エクセルでは、罫線を引くことも結構多い作業です。1種類の線だけなら簡単ですが、セルの内容に応じて二重線・太線・点線・1点鎖線などを分けて引くのは面倒です。また、どこに線を引くのかを目視で確認していると、間違いが発生しやすいです。ここでは、基本的な線の引き方をまず説明し、その後でセルの内容に応じた線を引く方法を紹介します。

罫線を引くには、2つの段階を経ます。

- createStyle()で書式の定義
- addStyle()で設定した書式をセルに追加

練習として設定可能な罫線の一覧を作成します。罫線の位置(border)をセルの下("bottom")とします。罫線の種類(style)は"thin"から"slantDashDot"までの13種類があります(図9.8)。これらをもとに、createStyle()で罫線の位置と種類についてスタイル(styles)を設定します。なお、purrr::map()とpurrr::iwalk()は繰り返しをする関数です(2.8節を参照)。

次に、createWorkbook()でワークシートを作成し、writeData()で罫線の種類の文字列(style)を書き込んでいます。最後に、addStyle()で罫線を適用して、saveWorkbook()でワークブックを書き込みます。以下のコードを実行すると、図9.8のように罫線を引いたエクセルができます。

コード 9.31 (excel-border-style.R) : 設定可能な罫線の一覧作成

```
border <- "bottom"
style <-
  c("thin", "medium", "dashed", "dotted", "thick", "double",
    "hair", "mediumDashed", "dashDot", "mediumDashDot",
    "dashDotDot", "mediumDashDotDot", "slantDashDot")
styles <-
  style |>
    purrr::map(\(x){ createStyle(border = border, borderStyle = x) })
wb <- createWorkbook()          # ワークブックを作成
addWorksheet(wb, 1, zoom = 200) # シートを追加
writeData(wb, sheet = 1, style) # データ書き込み
file_border <- fs::path_temp("border.xlsx")
styles |>
  purrr::iwalk(\(.x, .y){
    addStyle(wb, 1,           # 罫線のスタイルを適用
              style = .x,      # .x: style[[i]], i は 1 から n まで
              rows = .y, cols = 1) # .y: i
  })
saveWorkbook(wb, file_border, overwrite = TRUE) # 書き込み
# shell.exec(file_border)
```

	A	B
1	thin	
2	medium	
3	dashed	
4	dotted	
5	thick	
6	double	
7	hair	
8	mediumDashed	
9	dashDot	
10	mediumDashDot	
11	dashDotDot	
12	mediumDashDotDot	
13	slantDashDot	
14		

図 9.8: 設定可能な罫線の一覧

データのあるところ全てに罫線を引くなら、次のような関数を定義しておきます。 基本的な考え方は、これまでと同じで `map_wb()` で全てのシートに適用可能なように、糖衣関数を作成します。

罫線の位置と種類は場合によって変えることが想定されるので、引数で変更可能なようにします。また、`createStyle()`で設定したスタイルを指定できるようにします。

あとは、各シートの行数と列数を取得して、その範囲に罫線のスタイルを設定します。

コード 9.32 (`excel-border-fun.R`) : 単純な罫線を引く関数

```
set_border <- function(wb, sheet,
                        border = "Bottom", borderStyle = "thin",
                        style = NULL){
  if(is.null(style)){
    style <- createStyle(border = border, borderStyle = borderStyle)
  }
  rows <- rows_wb_sheet(wb, sheet) # シートごとの行数
  cols <- cols_wb_sheet(wb, sheet) # シートごとの列数
  addStyle(wb, sheet, style = style, rows = rows, cols = cols,
           gridExpand = TRUE, # rows と cols の組み合わせで範囲を拡張
           stack = TRUE)      # 既存の書式に追加
}
```

コード 9.33 (`excel-rows.R`) : シートごとの行数を取得する関数

```
rows_wb_sheet <- function(wb, sheet) {
  rows <- openxlsx::readWorkbook(wb, sheet) |>
    nrow() |>
    magrittr::add(1) |>
    seq() # +1: 列名の1行を追加
}
```

関数が定義できたので、先程の各種罫線のワークブックに罫線を引きます。罫線を引いて、列幅を自動で設定したのが図 9.9 です。列幅が若干足りませんが、フォントによるものだと思われます。正確に修正するには手作業が必要ではあるものの、通常に使う分には問題ないでしょう。

コード 9.34 (`excel-border-all.R`) : データの範囲に罫線を引く

```
map_wb(wb, set_border, border = c("bottom", "right"))
map_wb(wb, set_col_width)
saveWorkbook(wb, file_border, overwrite = TRUE)
```

	A
1	thin
2	medium
3	dashed
4	dotted
5	thick
6	double
7	hair
8	mediumDashed
9	dashDot
10	mediumDashDot
11	dashDotDot
12	mediumDashDotDot
13	slantDashDot
14	

図 9.9: データの範囲に引いた罫線

全てに同じ罫線を引くのではなく、セルの内容に応じて罫線の種類を使い分けて引きたいときもあるでしょう。そのときは罫線の位置を目視で確認しながら、マウスやカーソルキーで範囲選択して、罫線を引く必要があります。罫線を引く操作の方法は色々とありますが、どれも結構面倒です。

罫線を引く条件が決まっているれば自動化が可能です。Rでコードを書けば、同じ作業を何度も手間なくできます。ここでは、次のような条件で線を引きます。

- ・ 特定の列でセルの中身が上のセルと異なったら太線で区別
- ・ セルに特定の文字があれば、その列の右側に二重線
- ・ データの範囲の外枠に 1 点鎖線

今回の時間割では次のように線を引く設定です。

- ・ 学年(grade)の区別で太線
- ・ 時限(hour)の列の右側に二重線
- ・ 時間割の外枠に 1 点鎖線

上の条件の 3 つのうちでは、学年の区別の判定は工夫が必要です。`dplyr::lag()`は、ベクトルをずらす関数です。ずらした内容と元の内容とを比較して、一致しないところは学年が異なるところ、つまり学年の区別にあたります。データフレームの 1 行目は、タイトル行ですので、`c(1, ...)`で区別にも追加するとともに、データ全体も 1 つ足します。これで、学年の区別として罫線を引くべき位置がわかります。

また、`border_between_categ()`でカテゴリの区別で罫線を引けるようにします。さらに、`set_border()`を拡張して、罫線を引く行と列を指定できるように再定義します。

コード 9.35 (excel-border-condition-fun.R) : セルの内容の区別で罫線を引く関数

```
# 区別として罫線を引く位置
new_categ_rows <- function(wb, sheet, col_name, include_end = FALSE){
  df <- openxlsx::readWorkbook(wb, sheet)
  old_categ <- df[[col_name]] != dplyr::lag(df[[col_name]]) # lag: 1つずらす
  old_categ <- c(1, which(old_categ)) # 1: タイトル行
  if(include_end){ # 最後のカテゴリの終わりを含める
    old_categ <- c(old_categ, nrow(df) + 1) # nrow(df)+1: 最終行の次
  }
  new_categ <- old_categ + 1 # タイトルの分として+1 する
  return(new_categ)
}

# 学年別で太線を引く
border_between_categ <- function(wb, sheet, categ){
  style <- createStyle(border = "top", borderStyle = "thick") # 書式
  rows <- new_categ_rows(wb, sheet, categ) # 範囲
  set_border(wb, sheet, rows = rows, style = style) # 設定
}

# set_border()の拡張版
set_border <- function(wb, sheet, rows = NULL, cols = NULL,
                      border, borderStyle, style = NULL,
                      gridExpand = TRUE){
  if(is.null(style)){
    style <- createStyle(border = border, borderStyle = borderStyle)
  }
  if(is.null(rows)){ rows <- rows_wb_sheet(wb, sheet) }
  if(is.null(cols)){ cols <- cols_wb_sheet(wb, sheet) }
  addStyle(wb, sheet, style = style, rows = rows, cols = cols,
           gridExpand = TRUE, # rows と cols の組み合わせで範囲を拡張
           stack = TRUE) # 既存の書式に追加
}
```

上記の関数を実行して罫線を引きます(図 9.10).

コード 9.36 (excel-border-condition-grade.R) : 学年の区別で太線を引く

```
wb <- loadWorkbook(file_timetable)
map_wb(wb, border_between_categ, categ = "grade")
saveWorkbook(wb, file_timetable, overwrite = TRUE)
```

A	B	C	D	E	F	G	
1	semes	gra	hc	1月	2火	3水	4木
2	first	1	1	健康習A_藤井	英語話A_中村	ハウグ論_村上	
3	first	1	1			健康習A_橋本	
4	first	1	1			生態入門_森本	
5	first	1	2	健康習A_山田	情報タA_足立		化学基礎
6	first	1	2	健康習A_藤井			
7	first	1	3	経済概論_高橋		健康習C_伊藤	化学基礎
8	first	1	3	健康習C_山田		生物基礎_小林	健康習C_
9	first	1	3	食品品学_西村			健康習C_
10	first	1	4	健康習A_山田	生活習A_橋本	健康習A_伊藤	健康習A_
11	first	1	4		生活習A_高橋	生物基礎_小林	健康習C_
12	first	1	4		生活習A_山下		生活入門
13	first	1	4		生活習A_森本		
14	first	1	4		生活習A_村上		
15	first	1	4		生活習A_林		
16	first	1	5	生命入門_小林			
17	first	2	1	カラ実習_林	ハウ・B_藤田	ファス論_高橋	住生ン論
18	first	2	1				
19	first	2	2	カラ実習_林	ハウ・B_藤田	フート論_山口	
20	first	2	3	生涯科学_藤井		衣生ン論_山下	
21	first	2	3	服飾化論_山下			
22	first	2	4	地球境論_小林			
23	first	3	1	食品儀論_西村	食品工学_西村	服飾実習_林	
24	first	3	2	社会障論_高橋		服飾実習_林	ファン論
25	first	3	2				生活演習
26	first	3	3		イント論_藤原		生活習A

図 9.10: 学年の区分として追加した太線

次に、セルの内容が指定した文字列と合致したときに、罫線を引く関数を定義します。

`colnames(df) == str` で列名と指定した文字列が一致しているかを判定して、その番号を `which()` で取得し、その列の右に罫線を引きます。

コード 9.37 (excel-content-cols-fun.R) : 文字列に合致する列番号を取得する関数

```
content_cols <- function(wb, sheet, str){
  df <- openxlsx::readWorkbook(wb, sheet)
  which(colnames(df) == str)
}
```

コード 9.38 (excel-border-condition-hour-fun.R) : セルの文字列に合わせて罫線を引く関数

```
border_between_contents <- function(wb, sheet, border = "right",
                                      borderStyle = "double", str){
  style <- createStyle(border = border, borderStyle = borderStyle) # 書式
  cols <- content_cols(wb, sheet, str = str) # 範囲
  set_border(wb, sheet, cols = cols, style = style) # 設定
}
```

定義した関数を実行して、罫線を引きます(図 9.11)。

コード 9.39 (excel-border-condition-hour.R) : hour 列の右に二重線を引く

```
wb <- loadWorkbook(file_timetable)
map_wb(wb, border_between_contents, str = "hour")
saveWorkbook(wb, file_timetable, overwrite = TRUE)
```

A	B	C	D	E	F	G	
1	semes	gra	hd	1月	2火	3水	4木
2	first	1	1	健康習A_藤井	英語話A_中村	ハウ グ論_村上	
3	first	1	1			健康習A_橋本	
4	first	1	1			生態入門_森本	
5	first	1	2	健康習A_山田	情報タA_足立		化学基礎
6	first	1	2	健康習A_藤井			
7	first	1	3	経済概論_高橋		健康習C_伊藤	化学基礎
8	first	1	3	健康習C_山田		生物基礎_小林	健康習C_
9	first	1	3	食品品学_西村			健康習C_
10	first	1	4	健康習A_山田	生活習A_橋本	健康習A_伊藤	健康習A_
11	first	1	4		生活習A_高橋	生物基礎_小林	健康習C_
12	first	1	4		生活習A_山下		生活入門
13	first	1	4		生活習A_森本		
14	first	1	4		生活習A_村上		
15	first	1	4		生活習A_林		
16	first	1	5	生命入門_小林			
17	first	2	1	カラ実習_林	ハウ・B_藤田	ファス論_高橋	住生ン論
18	first	2	1				
19	first	2	2	カラ実習_林	ハウ・B_藤田	フート論_山口	
20	first	2	3	生涯科学_藤井		衣生ン論_山下	
21	first	2	3	服飾化論_山下			
22	first	2	4	地球境論_小林			
23	first	3	1	食品儀論_西村	食品工学_西村	服飾実習_林	
24	first	3	2	社会障論_高橋		服飾実習_林	ファン論
25	first	3	2				生活演習
26	first	3	3		イント論_藤原		生活習A

図 9.11: hour 列の右に二重線を追加

最後に、データの外枠に罫線を引く関数を定義します。ここでは、難しいことはありませんが、上下左右を間違いなく定義します。

コード 9.40 (excel-border-condition-frame-fun.R) : データの外枠に罫線を引く関数

```
border_frame <- function(wb, sheet, borderStyle = "mediumDashDot") {
  # 書式
  style_t <- createStyle(border = "top", borderStyle = borderStyle)
  style_b <- createStyle(border = "bottom", borderStyle = borderStyle)
  style_l <- createStyle(border = "left", borderStyle = borderStyle)
  style_r <- createStyle(border = "right", borderStyle = borderStyle)
  # 範囲
  rows <- rows_wb_sheet(wb, sheet)
  cols <- cols_wb_sheet(wb, sheet)
  rows_t <- 1
  rows_b <- max(rows)
  cols_l <- 1
  cols_r <- max(cols)
  # 書式の適用
  set_border(wb, sheet, rows = rows_t, cols = cols, style = style_t) # 上
  set_border(wb, sheet, rows = rows_b, cols = cols, style = style_b) # 下
  set_border(wb, sheet, rows = rows, cols = cols_l, style = style_l) # 左
  set_border(wb, sheet, rows = rows, cols = cols_r, style = style_r) # 右
}
```

定義した関数で外枠に 1 点鎖線を引きます。

コード 9.41 (excel-border-condition-frame.R) : データの外枠に 1 点鎖線を引く

```
wb <- loadWorkbook(file_timetable)
map_wb(wb, border_frame)
saveWorkbook(wb, file_timetable, overwrite = TRUE)
```

A	F	G	H	I	J
1 semes	3水	4木	5金		
2 first	ハウグ論_村上		健康習A_橋本		
3 first	健康習A_橋本		健康習C_谷口		
4 first	生態入門_森本				
5 first		化学基礎_吉田	英語語A_岡田		
6 first			健康習A_谷口		
7 first	健康習C_伊藤	化学基礎_吉田	生涯ツ論_橋本		
8 first	生物基礎_小林	健康習C_松本			
9 first		健康習C_藤井			
10 first	健康習A_伊藤	健康習A_松本			
11 first	生物基礎_小林	健康習C_藤井			
12 first		生活入門_林			
13 first					
14 first					
15 first					
16 first					
17 first	ファス論_高橋	住生ン論_村上	トレ科学_清水		
18 first			環境イ論_森本		
19 first	フート論_山口		トレグ論_清水		
20 first			食生実習_山口		
21 first					
22 first			食生実習_山口		
23 first	服飾実習_林		フート論_大西		
24 first	服飾実習_林	ファン論_林	レク技A_橋本		
25 first		生活演習_森本			
26 first		生活習A_橋本	住居居中_村ト		

図 9.12: データの外枠に 1 点鎖線を追加

これで各種の罫線を引く関数が完成しました。意図どおりに動作していることも確認できました(図 9.12)。

関数の定義には、正直なところ少し時間がかかります。それでも、1 回定義すれば何度でも自動的に同じことができますし、間違いもありません。上記の関数をもとにして読者の使いやすいように関数を修正してください。

9.10 条件付き書式設定による強調表示

エクセルのデータを見やすくするために、文字列や数値に応じて色を付けることがあります。その場合、個別に色付けをしていいでしょうか。目視での確認には手間がかかり、漏れや間違いのもとです。検索して作業するのも手間がかかります。エクセルの条件付書式設定を使えば、一定の条件でエクセルが判別してくれるので非常に便利です。ただし、条件が多くなると書式の設定に多くの手間がかかります。

このようなときには、`conditionalFormatting()`で条件付き書式設定をしましょう。罫線を引くのと同様に`createStyle()`でセルの文字色や背景色の書式を定義できます。`createStyle()`の引数`bgFill`には"yellow"や"green"などの色名や"#FFFF00"(黄色)のようなカラーコードが使用できます。文字色や背景色を変更する以外にも、色を段階的に変化させるカラースケールやセル内の棒グラフであるデータバーを使うことができます。数値を文字としてだけでなく、視覚的に分かりやすく表現できます。

次のコードでは、条件付き書式設定で背景色を変更する関数を定義しています。既定値では、strings で指定する文字列の背景色を黄色("#FFFF00")にします。

コード 9.42 (excel-bg-color-fun.R) : 条件付き書式設定による背景色を変更する関数

```
set_bg_color <- function(wb, sheet, color = "#FFFF00", strings){
  bg_color <- openxlsx::createStyle(bgFill = color)
  for(str in strings){ # string の数だけ繰り返し
    openxlsx::conditionalFormatting(wb, sheet,
      rows = rows_wb_sheet(wb, sheet),
      cols = cols_wb_sheet(wb, sheet),
      rule = str, style = bg_color, type = "contains")
  }
}
```

次のコードを実行すると、上で定義した関数を利用して、衣、食、住が含まれるセルの色を黄色に着色します(図 9.13)。

コード 9.43 (excel-bg-color.R) : 条件付き書式設定による背景色の変更

```
map_wb(wb, set_bg_color, color = "yellow", strings = c("衣", "食", "住"))
saveWorkbook(wb, file_timetable, overwrite = TRUE)
# shell.exec(file_timetable)
```

	A	B	C	D	E	F	
1	semen	gra	hc	1月	2火	3水	4木
2	first	1	1	健康習A_藤井	英語話A_中村	ハウグ論_村上	
3	first	1	1			健康習A_橋本	
4	first	1	1			生態入門_森本	
5	first	1	2	健康習A_山田	情報タA_足立		化学基礎
6	first	1	2	健康習A_藤井			
7	first	1	3	経済概論_高橋		健康習C_伊藤	化学基礎
8	first	1	3	健康習C_山田		生物基礎_小林	健康習C_
9	first	1	3	食品品学_西村			健康習C_
10	first	1	4	健康習A_山田	生活習A_橋本	健康習A_伊藤	健康習A_
11	first	1	4		生活習A_高橋	生物基礎_小林	健康習C_
12	first	1	4		生活習A_山下		生活入門
13	first	1	4		生活習A_森本		
14	first	1	4		生活習A_村上		
15	first	1	4		生活習A_林		
16	first	1	5	生命入門_小林			
17	first	2	1	カラ実習_林	ハウ・B_藤田	ファス論_高橋	住生ン論
18	first	2	1				
19	first	2	2	カラ実習_林	ハウ・B_藤田	フート論_山口	
20	first	2	3	生涯科学_藤井	衣生ン論_山下		
21	first	2	3	服飾化論_山下			
22	first	2	4	地球壇論_小林			
23	first	3	1	食品衛論_西村	食品工学_西村	服飾実習_林	
24	first	3	2	社会障論_高橋		服飾実習_林	ファン論
25	first	3	2				生活演習
26	first	3	3		イント論_藤原		生活習A

図 9.13: 条件付き書式設定で変更した背景色

conditionalFormatting()の引数の type には、"expression"(既定値), "colourScale", "dataBar", "duplicates", "beginsWith", "endsWith", "topN", "bottomN", "contains", "notContains"を指定できます。

"expression"では、エクセルのセルに条件式を指定するときのような不等号が使えます。ただし、等しいときは "=="(エクセルでは=)を、等しくないときは"!="(エクセルでは<>)を使います。たとえば、セルの値が0より大きい条件とするなら rule = ">0", A1 のセルの値が10以下を条件とするなら rule = "\$A\$1<=0"とします。このときの rule は指定範囲の最初のセルです。そのため、指定する全範囲で A1 を参照するときには、このように絶対参照である\$をつける必要があります。もし、A1 とすると相対参照になります。

"colourScale"と".databar"は、それぞれ色を段階的に変化させるカラースケールとセル内の棒グラフを表示させるデータバーを表示します。たとえば、セルの中に1から10までの数値があるとします。type = "colourScale", style = c("blue", "white"), rule = c(1, 10)とすると、1のところは濃い青色、5のところは青色、10のところは白色の背景色になります。type = ".databar", style = c("yellow")とするとセル幅の最大値とした黄色の棒グラフがセル内に表示されます。

"duplicates"は指定した範囲内で重複するものを、"topN"と"bottomN"はそれぞれ上位と下位N個のものについて条件付き書式設定できます。"beginsWith"と"endsWith"はそれぞれ開始と終了の文字列を、"contains"と"notContains"は文字列を含む・含まないものを指定できます。

次のコードでは、色々な条件付き書式設定をしています(図9.14)。typeによって指定する引数が少し異なります。また、"colourScale"と".databar"のみstyleの色を文字列で指定可能です。他のtypeでは、createStyle()で生成したStyleオブジェクトをstyleに指定します。styleを省略したときは既定値の createStyle(fontColour = "#9C0006", bgFill = "#FFC7CE") (文字は赤みがかった黒、背景は薄い桃色)が使われます。詳細な設定は、?conditionalFormattingでヘルプを参照してください。

コード9.44 (excel-conditionals.R) : 色々な条件付き書式設定の例

```
val <- 1:10
str <- stringr::fruit[val]
df <- tibble::tibble(
  equal_3 = val, colourScale = val, databar = val, top5 = val, bottom3 = val,
  duplicates = letters[sample(1:9, 10, replace = TRUE)],
  beginsWith_a = str, endsWith_e = str, contains_p = str, notContains_c = str)
file_cond <- fs::path_temp("conditional.xlsx")
write.xlsx(df, file_cond)
wb <- loadWorkbook(file_cond)
rows <- 2:11
conditionalFormatting(wb, 1, cols = 1, rows = rows,
  type = "expression", rule = "==3") # 3と同じ
conditionalFormatting(wb, 1, cols = 2, rows = rows,
  type = "colourScale", style = c("blue", "white"), # カラースケール
  rule = c(0, 10))
conditionalFormatting(wb, 1, cols = 3, rows = rows,
  type = ".databar", style = c("yellow")) # データバー
```

```

conditionalFormatting(wb, 1, cols = 4, rows = rows,
  type = "topN", rank = 5)                                # 上位 5 つ
conditionalFormatting(wb, 1, cols = 5, rows = rows,
  type = "bottomN", rank = 3)                               # 下位 3 つ
conditionalFormatting(wb, 1, cols = 6, rows = rows,
  type = "duplicates")                                    # 重複
conditionalFormatting(wb, 1, cols = 7, rows = rows,
  type = "beginsWith", rule = "a")                         # a で始まる
conditionalFormatting(wb, 1, cols = 8, rows = rows,
  type = "endsWith", rule = "e")                           # e で終わる
conditionalFormatting(wb, 1, cols = 9, rows = rows,
  type = "contains", rule = "p")                            # p を含む
conditionalFormatting(wb, 1, cols = 10, rows = rows,
  type = "notContains", rule = "c")                         # c を含まない
saveWorkbook(wb, file_cond, overwrite = TRUE)

```

	A	B	C	D	E	F	G	H	I	J	K
1	equal_3	colourScale	dataBar	top5	bottom3	duplicates	beginsWith_a	endsWith_e	contains_p	notContains_c	
2	1	1	1	1	1 g	apple	apple	apple	apple	apple	
3	2	2	2	2	2 e	apricot	apricot	apricot	apricot	apricot	
4	3	3	3	3	3 e	avocado	avocado	avocado	avocado	avocado	
5	4	4	4	4	4 i	banana	banana	banana	banana	banana	
6	5	5	5	5	5 i	bell pepper					
7	6	6	6	6	6 h	bilberry	bilberry	bilberry	bilberry	bilberry	
8	7	7	7	7	7 b	blackberry	blackberry	blackberry	blackberry	blackberry	
9	8	8	8	8	8 a	blackcurrant	blackcurrant	blackcurrant	blackcurrant	blackcurrant	
10	9	9	9	9	9 g	blood orange					
11	10	10	10	10	10 f	blueberry	blueberry	blueberry	blueberry	blueberry	

図 9.14: 色々な条件付き書式設定の例

9.11 ヘッダー・フッター

ヘッダーとフッターを設定するには、`setHeaderFooter()`を使います(図 9.15)。引数 `header` にはヘッダーの、`footer` にはフッターの内容を指定します。内容は、`c("左の内容", "中央の内容", "右の内容")` の 3 項目からなる文字列ベクトルで指定します。何も入力しないときは、`NA` とします。最初のページの指定には、`firstHeader` と `firstFooter` を、偶数ページには `evenHeader` と `evenFooter` を使用します。また、専用の記号を使ってページ番号なども使うことができます(表 9.3)。なお、図 9.15 では次節で見やすく設定したものと比較するため、あえて縦長のページ設定にしています。

表 9.3: ヘッダーとフッター指定の記号と意味

記号	意味
&[Page]	ページ番号
&[Pages]	全ページ数
&[Date]	現在の日付

記号	意味
&[Time]	現在時刻
&[Path]	ファイルのパス
&[File]	ファイル名
&[Tab]	シート名

コード 9.45 (excel-header.R) : ヘッダーとフッターの設定

```
wb <- loadWorkbook(file_timetable)
header <- c("&[Date] &[Time]", "甲南女子大学の時間割", "&[File] &[Tab]")
footer <- c(NA, "&[Page] / &[Pages]", NA)
map_wb(wb, setHeaderFooter, header = header, footer = footer)
saveWorkbook(wb, file_timetable, overwrite = TRUE)
```

甲南女子大学の時間割 timetable.xlsx first			
semester	grade	hour	月
first	1	1	健康習A_藤井
first	1	1	英語話A_中村
first	1	1	ハウグ論_村上
first	1	2	健康習A_山田
first	1	2	情報タ_足立
first	1	3	経済概論_高橋
first	1	3	健康習C_伊藤
first	1	3	生態入門_森本
first	1	3	食品品学_西村
first	1	4	健康習A_山田
first	1	4	生活習A_橋本
first	1	4	健康習A_伊藤
first	1	4	生活習A_高橋
first	1	4	生物基礎_小林
first	1	4	生活習A_山下
first	1	4	生活習A_森本
first	1	4	生活習A_村上
first	1	4	生活習A_林
first	1	5	生命入門_小林
first	2	1	カラ実習_林
first	2	1	ハウ・B_藤田
first	2	2	ハウ・B_藤田
first	2	2	ファス論_高橋
first	2	3	生涯科学_藤井
first	2	3	フート論_山口
first	2	3	服飾化論_山下
first	2	4	衣生ン論_山口
first	2	4	地球境論_小林
first	3	1	食品衛論_西村
first	3	1	食品工学_西村
first	3	2	服飾実習_林
first	3	2	社会障論_高橋
first	3	3	イント論_藤原
first	3	3	食品演習_山口
first	3	3	食品実習_山口
first	3	3	食品実習_山口
first	3	3	食品実習_山口
first	3	4	食品演習_山口
first	3	4	食品実習_山口
first	4	3	卒業実A_橋本
first	4	3	卒業実A_高橋
first	4	3	卒業実A_山下
first	4	3	卒業実A_山口
first	4	3	卒業実A_森本
first	4	3	卒業実A_西村
first	4	3	卒業実A_村上
first	4	3	卒業実A_藤原
first	4	3	卒業実A_林

図 9.15: 設定したヘッダーとフッター

9.12 ページ設定

印刷ページの向き・拡大率・余白などは `pageSetup()` で設定できます(表 9.4, 図 9.16)。

表 9.4: `pageSetup()` の主な引数と内容

引数	内容	値(意味, 単位など)
<code>orientation</code>	印刷向き	"landscape"(横),"portrait"(縦)
<code>scale</code>	拡大率	数値(%)
<code>left,right,top,bottom</code>	左右上下の余白	数値(インチ)
<code>header,footer</code>	ヘッダーとフッターの幅	数値(インチ)
<code>fitToWidth,fitToHeight</code>	幅・高さに合わせる	TRUE/FALSE
<code>paperSize</code>	用紙サイズ	整数(既定値は 9 で A4 版)
<code>printTitleRows,printTitleCol</code>	行・列タイトル	整数のベクトル(列番号)

ここでは横向き、幅に合わせて(横 1 ページ)、1 行目をタイトルにするようにページを設定します。まずは例にならって、糖衣関数を定義します。

コード 9.46 (excel-page-setup-fun.R) : 横向き等に設定する糖衣関数

```
page_setup <- function(wb, sheet, ...){  
  pageSetup(wb, sheet, ...)  
}
```

定義した関数を使って、ページ設定します。

コード 9.47 (excel-page-setup.R) : 横向き等にページを設定する

```
map_wb(wb, page_setup,  
       orientation = "landscape", # 横向き,  
       fitToWidth = TRUE,          # 幅に合わせる(横 1 ページ)  
       printTitleRows = 1)         # 1 行目をタイトルに  
saveWorkbook(wb, file_timetable, overwrite = TRUE)  
# shell.exec(file_timetable)
```

甲南女子大学の時間割									
semester	grade	月	火	水	木	金	土	日	休
first	hour 1	健康栄養A_難井	英語会話A_中村	ハウジ論_村上					健康栄養A_難井
first	1	1			健康栄養A_赤本				健康栄養A_谷口
first	1	1			生物実習A_森本				
first	1	1	健康栄養A_山田	情報タクニズム					化学基礎A_吉田
first	1	2	健康栄養A_森井						英語会話A_岡田
first	1	3	経済政策_赤崎						健康栄養A_谷口
first	1	3	健康栄養A_山田						生物基礎A_松本
first	1	3	食品工学_西村						健康栄養A_松本
first	1	4	健康栄養A_山田	生活習A_森本	健康栄養A_伊藤				健康栄養A_森井
first	1	4		生活習A_赤崎					生物基礎A_森井
first	1	4		生活習A_山下	生物基礎A_小林				生活習A_林
first	1	4			生活習A_山下				
first	1	4							
first	1	4							
first	1	5	生命人門A_小林						
first	2	1	万力A_美智子	ハウ・ビ_森田	フックス論_高橋	住生・演_村上			トシヒ学_清水
first	2	1							環境・地_森本
first	2	2	カワハ家実_林	ハウ・ビ_森田	フート論_山口				トレーニング_清水
first	2	3	生物学_森井		衣生・演_山下				食育実習_山口
first	2	3	開拓化論_山下						
first	2	4	地図推論_小林						食育実習_山口
first	3	1	食品工学_西村	食品工学_西村	脇野実習_林				フード論_大西
first	3	2	社会論_高橋		脇野実習_林	ファ・演_林			フレクター_橋本
first	3	2				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3	イント論_藤原			生活習A_赤崎			
first	3	3	食育実習_山口			生活習A_山下			
first	3	3	食育実習_山口			生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first	3	3				生活習A_山下			
first	3	3				生活習A_森本			
first	3	3				生活習A_赤崎			
first</									

図 9.16: 横向き等に設定したページ

9.13 改ページの設定

改ページの位置が思い通りにならないときは、任意のところに改ページを設定できます。今回の時間割であれば、学年の区切りで改ページすることを考えましょう。その場合は、単純に学年の区切りで改ページを設定します。

改ページの設定には `pageBreak()` を使います。 `pageBreak()` の引数は、 `wb` と `sheet` 以外に、 `i` と `type` があります。 `i` は改ページ位置の行か列、 `type` は既定値が "row" (行、 水平方向の改ページ) で、 "column" (列、 垂直方向の改ページ) にも設定できます。

ます、学年の区切りは罫線(9.9 参照)で作成した関数 `new_categ_rows()`を使って取得します。

コード 9.48 (excel-breaks-grade-fun.R) : 改ページのための区切りを取得する関数

```
breaks <- function(wb, sheet, col_name){  
  res <-  
    new_categ_rows(wb, sheet, col_name,  
                  include_end = TRUE)[-1] |> # [-1] : タイトル行の区切りを削除  
    magrittr::subtract(1) # subtract(1)は-1と同じ : 区切りの上で改ページするため  
}  
## 動作確認  
## breaks(wb, 1, col_name = "grade")
```

次に、全シートで改ページを設定する関数を定義します。 基本的にはこれまでと同じように引数を渡すだけの関数です。

コード 9.49 (excel-page-breaks-grade-fun.R) : 学年ごとに改ページを設定する関数

```

page_break <- function(wb, sheet, type = "row", col_name) {
  brks <- breaks(wb, sheet, col_name)
  openxlsx::pageBreak(wb, sheet, i = brks, type)
}

```

定義した関数を使って学年別で改ページを設定します。次のコードではこれまでとは別の名前 (timetable_breaks.xlsx) で書き込みます。この後に別の位置で改ページの設定をしたいのですが、pageBreak()は改ページ位置の追加のみで解除ができないためです。また、ワークブックも別の変数(wb_breaks)としています。

コード 9.50 (excel-page-breaks-grade.R) : 学年ごとに改ページを設定する

```

wb_breaks <- wb
map_wb(wb_breaks, page_break, type = "row", "grade")
file_timetable_breaks <-
  fs::path_temp("timetable_breaks.xlsx") # 別名で書き込み
saveWorkbook(wb_breaks, file_timetable_breaks, overwrite = TRUE)

```

改ページプレビューで表示すると学年ごとに改ページが設定できたことを確認できます(図 9.17)。

	A	B	C	D	E	F	G	
1	semei	gra	h	1月	2火	3水	4木	5金
2	first	1	1	健康習A_藤井	英語話A_中村	ハウグ論_村上		健康習
3	first	1	1			健康習A_橋本		健康習
4	first	1	1			生態入門_森本		
5	first	1	2	健康習A_山田	情報タA_足立		化学基礎_吉田	英語認
6	first	1	2	健康習A_藤井			健康習C_伊藤	健康習
7	first	1	3	経済概論_高橋			化学基礎_吉田	生涯
8	first	1	3	健康習C_山田			健康習C_松本	生涯
9	first	1	3	食品品学_西村			健康習C_藤井	
10	first	1	4	健康習A_山田	生活習A_橋本	健康習A_伊藤	健康習A_松本	
11	first	1	4		生活習A_高橋	生物基礎_小林	健康習C_藤井	
12	first	1	4		生活習A_山下		健康習C_山下	
13	first	1	4			生活習A_森本	生活入門_林	
14	first	1	4			生活習A_村上		
15	first	1	4			生活習A_林		
16	first	1	5	生命入門_小林				
17	first	2	1	カラ実習_林	ハウ・B_藤田	ファス論_高橋	住生ン論_村上	トレチ
18	first	2	1					環境
19	first	2	2	カラ実習_林	ハウ・B_藤田	フート論_山口		トレク
20	first	2	3	生涯科学_藤井	衣生ン論_山下			食生業
21	first	2	3	服飾化論_山下				
22	first	2	4	地球地論_小林				食生業
23	first	3	1	食品面論_西村	食品工学_西村	服飾実習_林		フート
24	first	3	2	社会隣論_高橋		服飾実習_林		レク
25	first	3	2					レク
26	first	3	3		イント論_藤原			
27	first	3	3		食品演習_山口			
28	first	3	3		食品実習_山口			
29	first	3	3					
30	first	3	3					
31	first	3	3					
32	first	3	3					
33	first	3	3					
34	first	3	3					
35	first	3	4		食品演習_山口			
36	first	3	4		食品実習_山口			
37	first	4	3			卒業研究A_橋本		
38	first	4	3			卒業研究A_高橋		
39	first	4	3			卒業研究A_山下		
40	first	4	3			卒業研究A_山口		
41	first	4	3			卒業研究A_森本		
42	first	4	3			卒業研究A_西村		
43	first	4	3			卒業研究A_村上		
44	first	4	3			卒業研究A_藤原		
45	first	4	3			卒業研究A_林		

図 9.17: 学年ごとに 1 ページの設定

学年ごとのページにできましたが、やたらと改ページばかりで紙がもったいないです。できれば複数の学年が1ページに入ることを許容しつつ、1つの学年が複数ページにまたがらないようにしたいところです。これには、もう少し作業が必要です。具体的には、1ページあたりの最大行数を決めておき、この最大行数と学年の区切りとを比較して改ページの位置を決めます。

コード9.51 (excel-page-compare-breaks-fun.R) : 最大行数と改ページ位置を比較する関数

```
compare_page_break <- function(breaks, page_size = 30){  
  page_size_next <- page_size # 最大行数の初期値  
  page_breaks <- NULL # 結果を返す変数  
  while(length(breaks) > 1){ # 改ページが複数  
    is_under_page_brekas <- which(breaks < page_size_next) # 最大行数と比較  
    if(length(is_under_page_brekas) == 0){ # 最大行数以上  
      message("Page breaks is OVER Page Size!")  
      return(breaks)  
    }  
    index <- max(is_under_page_brekas) # 最大値の位置  
    page_breaks <- c(page_breaks, breaks[index]) # 改ページの追加  
    page_size_next <- breaks[index] + page_size # 次の最大行数  
    if(length(breaks) == index){ # 改ページ終了  
      return(page_breaks)  
    }  
    breaks <- breaks[(index + 1):length(breaks)] # 残りの改ページ  
  }  
  is_under_page_brekas <- which(breaks < page_size_next) # 最大行数と比較  
  if(length(is_under_page_brekas) == 0){ # 最大行数以上  
    message("Page breaks is OVER Page Size!")  
  }  
  return(c(page_breaks, breaks))  
}
```

改ページを breaks、ページの最大行数を 40 行として動作確認します。

コード9.52 (excel-page-compare-breaks.R) : 最大行数と改ページ位置を比較

```
# 動作確認  
breaks <- c(10, 20, 30, 50, 60, 65, 85, 90, 120)  
compare_page_break(breaks, page_size = 40)  
## [1] 30 65 90 120
```

ここでもワークブック内の全シートに適用するための関数を設定します。page_size は手作業でするので、横向きと縦向きで何行ぐらいいが1ページに収まるのか、事前に把握する必要があります。

コード 9.53 (excel-page-compare-add-breaks-fun.R) : 最大行数と改ページ位置を比較した改ページを設定する関数

```
add_page_break <- function(wb, sheet, col_name, page_size = 30){
  breaks <-
    new_categ_rows(wb, sheet, col_name,
      include_end = TRUE)[-1] |> # [-1]: タイトル行の区切りを削除
    magrittr::subtract(1)      # subtract(1)は-1と同じ: 区切りの上で改ページ
  breaks <- compare_page_break(breaks, page_size)
  openxlsx::pageBreak(wb, sheet, breaks)
}
```

関数の定義が終わったので、ページ内に複数学年を許容して改ページを設定します。改ページプレビューや印刷プレビューで、思い通りの改ページ位置に設定したことを確認できます(図 9.18)。

コード 9.54 (excel-page-compare-add-breaks.R) : ページ内に複数学年を許容して改ページを設定

```
map_wb(wb, add_page_break, col_name = "grade", page_size = 30)
saveWorkbook(wb, file_timetable, overwrite = TRUE)
```

	A	B	C	D	E	F	G	H
1	semestr	grd	h	1月	2火	3水	4木	5金
2	first	1	1	健康習A_藤井	英語論A_中村	ハウグ論_村上		健康習A_橋本
3	first	1	1			健康習A_橋本		健康習C_谷口
4	first	1	1			生態入門_森本		
5	first	1	2	健康習A_山田	情報タA_足立		化学基礎_吉田	英語論A_岡田
6	first	1	2	健康習A_藤井		健康習C_伊藤	化学基礎_吉田	健康習A_谷口
7	first	1	3	経済概論_高橋		生物基礎_小林	健康習C_松本	生涯ツ論_橋本
8	first	1	3	健康習C_山田			健康習A_藤井	
9	first	1	3	食品品学_西村			健康習A_伊藤	健康習A_松本
10	first	1	4	健康習A_山田	生活習A_橋本	生活習A_山下	健康習C_藤井	健康習C_松本
11	first	1	4		生活習A_高橋	生物基礎_小林	健康習A_山下	生活入門_林
12	first	1	4					
13	first	1	4					
14	first	1	4					
15	first	1	4					
16	first	1	5	生命入門_小林				
17	first	2	1	力ラ実習_林	ハウ・B_藤田	ファス論_高橋	住生ン論_村上	トレ科学_清水
18	first	2	1					環境ア論_森本
19	first	2	2	力ラ実習_林	ハウ・B_藤田	フト論_山口		トレグ論_清水
20	first	2	3	生涯科学_藤井	衣生ン論_山下			食生東習_山口
21	first	2	3	脳筋化論_山下				
22	first	2	4	地球境論_小林				食生東習_山口
23	first	3	1	食品衛論_西村	食品工学_西村	服飾実習_林		フト論_大西
24	first	3	2	社会障論_高橋		服飾実習_林		フレク接A_橋本
25	first	3	2					
26	first	3	3		イント論_藤原			
27	first	3	3		食品實習_山口			
28	first	3	3		食品実習_山口			
29	first	3	3					
30	first	3	3					
31	first	3	3					
32	first	3	3					
33	first	3	3					
34	first	3	3					
35	first	3	4		食品実習_山口			
36	first	3	4		食品実習_山口			
37	first	4	3			卒業究A_橋本		
38	first	4	3			卒業究A_高橋		
39	first	4	3			卒業究A_山下		
40	first	4	3			卒業究A_山口		
41	first	4	3			卒業究A_森本		
42	first	4	3			卒業究A_西村		
43	first	4	3			卒業究A_村上		
44	first	4	3			卒業究A_藤原		
45	first	4	3			卒業究A_林		

図 9.18: ページ内に複数学年を許容した改ページの設定

9.14 PDFへの変換

【追加】 エクセルの PDF への変換を追加しました。

RDCOMClient パッケージを使うと、アプリとしてのエクセルを操作できます。これをを利用して、エクセルを PDF に変換する関数を定義します。なお、RDCOMClient は、8.1 節を参考にしてインストールしてください。また、エクセルがインストールされている必要があります。

format_no は保存形式によって決まっており、PDF のときは 57 です。3 で txt, 6 で csv として保存できますが、`readr::write_delim()` や `readr::write_csv()` を使ったほうが簡単です。そのため、ここではエクセルから PDF に変換する関数を定義します。

コード 9.55 (excel-excel2pdf-fun.R) : エクセルを PDF に変換する関数

```
xlsx2pdf <- function(path){  
  format_no <- 57  
  path <- normalizePath(path)  
  converted <-  
    fs::path_ext_remove(path) |>  
    normalizePath(mustWork = FALSE)  
  xlsxApp <- RDCOMClient::COMCreate("Excel.Application") # エクセルの操作  
  xlsx <- xlsxApp$workbooks()$Open(path) # ファイルを開く  
  xlsx$SaveAs(converted, FileFormat = format_no) # 指定形式で保存  
  xlsx$close() # エクセルを閉じる  
  converted <- fs::path_ext_set(converted, "pdf") # 拡張子の設定  
  return(converted)  
}
```

関数を実行すると、ページ設定や改ページは設定したどおりの状態でエクセルを PDF に変換できます(図 9.19)。

コード 9.56 (excel-excel2pdf.R) : エクセルの PDF への変換

```
library(RDCOMClient)  
path_pdf <- xlsx2pdf(file_timetable)  
fs::path_file(path_pdf)  
## [1] "timetable.pdf"  
# shell.exec(path_pdf)
```

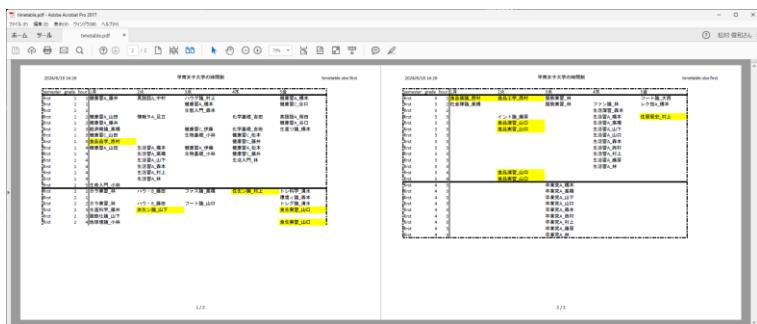


図 9.19: エクセルから変換した PDF

10 パワーポイントの操作

プレゼンや会議の資料作成にパワーポイントでよく使われますが、作成時には、文字や図型の大きさや位置の調整にマウスでの細かな作業が必要です。デザイン性を求めるのならマウスによる細かな調整は必要です。しかし、事務的な資料であれば同じ形式で理解しやすい資料にするのが重要です。その時には手作業でスライドを複写・修正するのは時間と手間がかかります。`officer`を使えば、使用する画像や表を指定するだけで、自動的に全く同じ形式・配置のスライドを作ることができます。

配布されたパワーポイントから文字列・図・表を取り出すこともできます。取り出した文字列・図・表は別のスライドへの使い回しができます。

また、`RDCOMClient`パッケージをつかえば、パワーポイントからPDF・画像・動画への変換も自動化できます。ワードやエクセルもRでPDFに変換できますので、会議用資料の元ファイルが異なる形式であっても、一連の操作としてPDFへの変換をRで自動化できます。

10.1 パワーポイントを操作するパッケージ

パワーポイントの操作には、`officer`パッケージを使います。

コード 10.1 (`powerpoint-install.R`) : `officer` のインストール

```
install.packages("officer")
```

インストールができれば `library()` で呼び出します。

コード 10.2 (`powerpoint-library.R`) : `officer` の呼び出し

```
library("officer")
```

10.2 ファイルの新規作成・読み込み

パワーポイントの資料の新規作成と読み込みは `read_pptx()` を使います。引数に何も指定しないと新規に作成されます。既存のファイルを読み込むときには、引数にファイルのパスを指定します。なお、`read_pptx()` で作成あるいは読み込んだものは、`rpptx` というクラスで、パワーポイントのオブジェクトです。

まず、資料を新規作成します。

コード 10.3 (`powerpoint-read.R`) : パワーポイントの新規作成

```
library(officer)
pp <- read_pptx()
# 既存ファイルの読み込み
```

```
# path <- "PATH/TO/POWERPOINT/FILE.pptx"
# pp <- read_pptx(path)
```

コード 10.4 (powerpoint-print.R) : パワーポイントの内容表示

```
pp
##  pptx document with 0 slide(s)
## Available layouts and their associated master(s) are:
##          layout      master
## 1      Title Slide Office Theme
## 2 Title and Content Office Theme
## 3   Section Header Office Theme
## 4      Two Content Office Theme
## 5      Comparison Office Theme
## 6     Title Only Office Theme
## 7      Blank Office Theme
```

新規作成したパワーポイントの内容を表示すると、`0 slide(s)`となっており、スライドが1枚も含まれていないことがわかります。また、使用可能な `layout` と `master` が表示されます。

10.3 レイアウトの確認

新規作成したパワーポイントにスライドを追加するには、レイアウトの名称などが必要です。これらは `layout_properties()` で確認できますが、スライドでの配置までは分かりません。そこで、実際のスライドに配置して確認します。

次のコードを実行すると、作業ディレクトリに `layout.pptx` が作成されます。関数の詳細は、次の項以降で説明しますので、細かなことは気にせず実行してください。ファイルを開くとレイアウトを確認できます。タイトルの位置などにレイアウトの名称を表示しています(図 10.1)。

コード 10.5 (powerpoint-layout.R) : レイアウトの確認

```
pp <- read_pptx()
layout_name <-
  layout_properties(pp)$name |>
  unique() # レイアウト名
for(ln in layout_name){ # レイアウトごとに
  pp <- add_slide(pp, layout = ln) # スライドを追加
  ph_label <-
    layout_properties(pp, layout = ln)$ph_label |>
    unique() # プレイスホルダーの一覧
  for(pl in ph_label){ # プレイスホルダーごとに
    val <- pl # プレイスホルダーのラベル
    if(stringr::str_detect(pl, "Title|タイトル")){
      val <- paste0(val, ":", ln)
    }
  }
```

```

loc <- ph_location_label(ph_label = pl)
pp <- ph_with(pp, value = val, location = loc) # プレイスホルダーを追加
}
if(ln == "Blank"){
  pp <- ph_with(pp, value = ln, location = ph_location())
}
}

path <- fs::path_temp("layout.pptx")
print(pp, target = path)
# shell.exec(path)

```

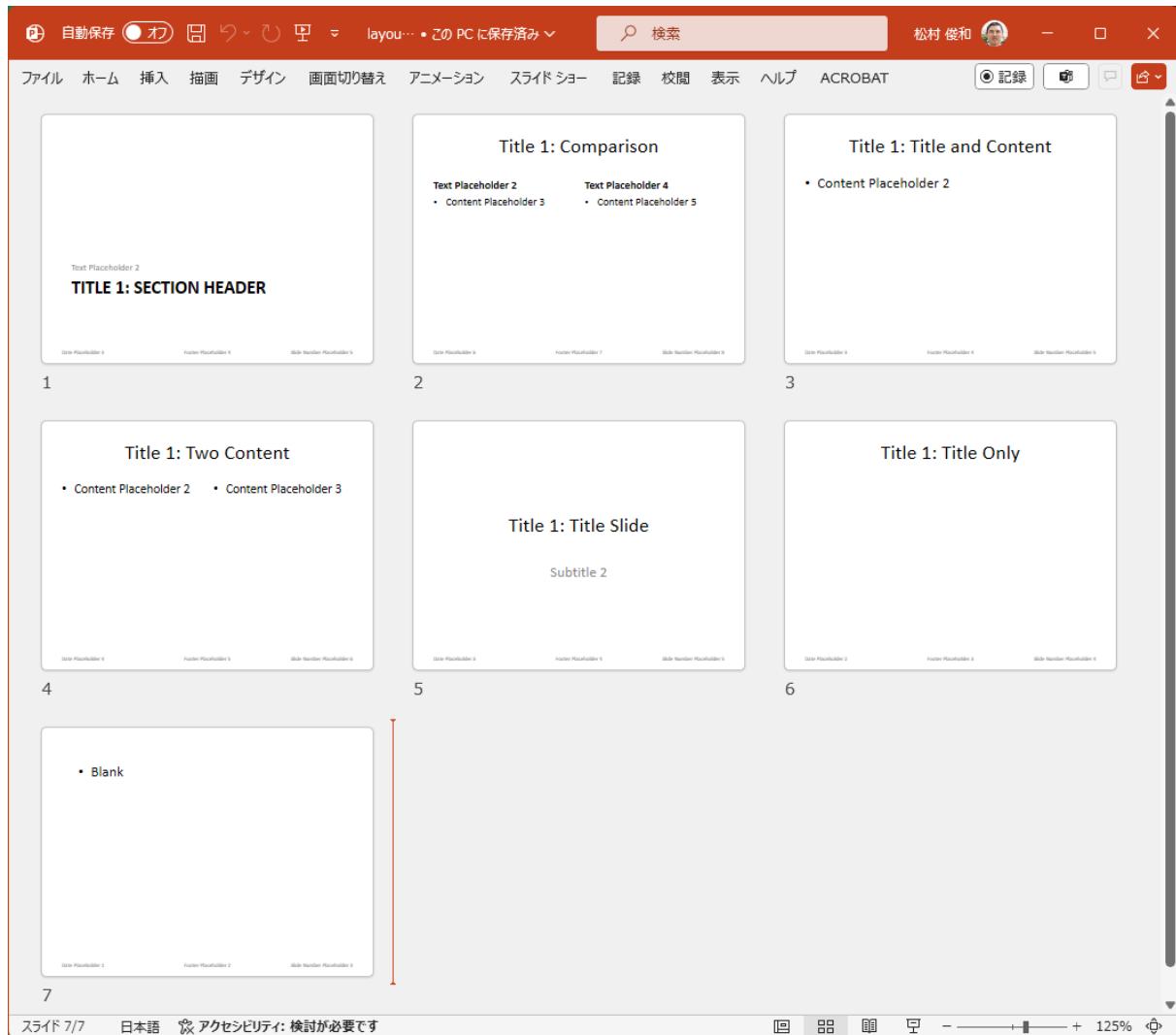


図 10.1: レイアウトの一覧

10.4 スライドの追加

ここからは、一からスライドを作るので、改めてファイルを新規作成します。その後で各種スライドを追加します。

特に設定せずに使えるレイアウトには、セクション見出し("Section Header"), 比較("Comparison"), タイトルとコンテンツ("Title and Content"), 2つのコンテンツ("Two

`Content`), タイトルスライド(`"Title Slide"`), タイトルのみ(`"Title Only"`), 白紙(`"Blank"`)があります(図 10.1).

スライドの追加には, `add_slide()`を使います. `add_slide()`の引数には, パワーポイントのオブジェクト, `layout`, `master`を指定します. `layout`には, `"Section Header"`などの文字列を指定します. `master`は, 既定値の`"Office Theme"`のままで構いません.

ここでは, レイアウトとして 2 つのコンテンツ(`"Two Content"`)のレイアウトでスライドを追加します.

コード 10.6 (powerpoint-add-slide.R) : スライドの追加

```
pp <- read_pptx() # 新規作成
layout <- "Two Content"
pp <- add_slide(pp, layout = layout)
path <- fs::path_temp("temp.pptx")
print(pp, target = path)
# shell.exec(path)
```

追加しただけでは, 中身がないので白紙のスライドです(図 10.2).

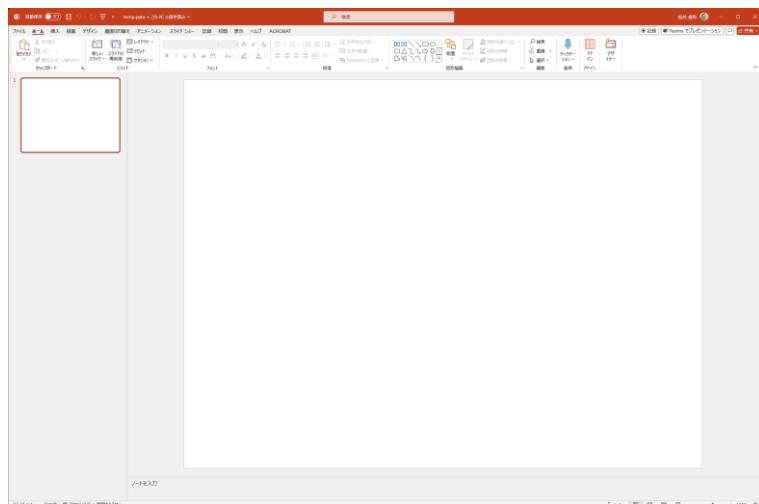


図 10.2: 追加したスライド

10.5 文字列の追加

内容を追加するには, `ph_with()`を使います. 引数 `value` に入力したい内容を, `location` に位置を指定します. レイアウト(図 10.1)で表示されている文字列を `ph_location_type(type = "title")` や `ph_location_label(ph_label = "Content Placeholder 2")` のように指定します. タイトルの位置に入力するときには `location = ph_location_type(type = "title")` とし, Content Placeholder 2 の位置に入力するには, `location = ph_location_label(ph_label = "Content Placeholder 2")` とします.

コード 10.7 (powerpoint-ph-with.R) : 内容の追加

```

pp <- ph_with(pp, value = "R による自動化の方法",
              location = ph_location_type(type = "title"))
pp <- ph_with(pp, paste0("手作業", 1:5),
              location = ph_location_label(
                ph_label = "Content Placeholder 2"))
pp <- ph_with(pp, paste0("自動化", 1:5),
              location = ph_location_label(
                ph_label = "Content Placeholder 3"))
print(pp, target = path)
# shell.exec(path)

```

上のコードを実行して内容を確認すると、タイトルと左右に項目が入力されているのがわかります(図 10.3)。

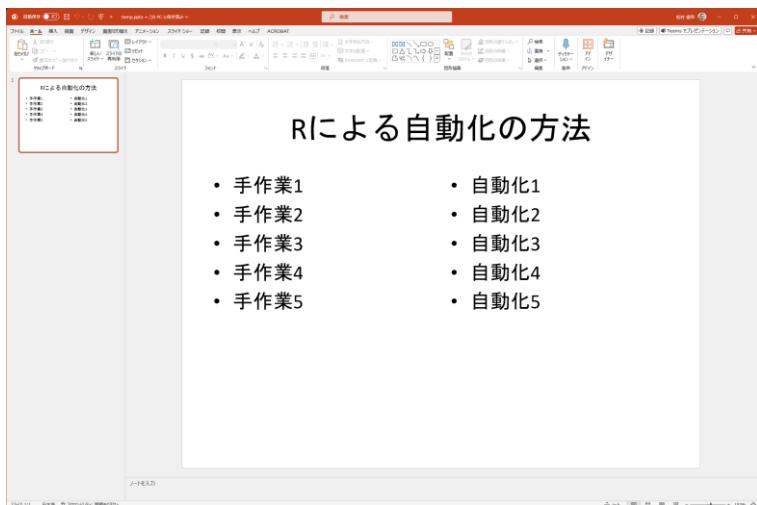


図 10.3: 内容を追加したスライド

10.6 箇条書きの追加

`unordered_list()`を使うと箇条書きを作成できます。引数 `level_list` に箇条書きの水準, `str_list` に箇条書きの文字列, `style` にフォントの書式などを指定します。なお, `fp_text` の引数 `font.size = 0` はそれぞれの水準での既定値のフォントサイズの意味です。`style` を省略すると、既定値の書式が設定されます。

箇条書きの内容は、`ph_with()`の引数 `value` に指定します。次のコードを実行すると、2 ページ目に色付きの箇条書きが、3 ページ目に黒色の箇条書きを追加できます(図 10.4)。

コード 10.8 (powerpoint-list.R) : パワーポイントへの箇条書きの追加

```

ul <-
  unordered_list(
    level_list = c(1, 2, 3),
    str_list = c("大項目", "中項目", "小項目"),
    style = list(fp_text(color = "red", font.size = 0),

```

```

        fp_text(color = "blue", font.size = 0),
        fp_text(color = "green", font.size = 0))
pp <- add_slide(pp)
pp <- ph_with(x = pp, value = ul, location = ph_location_type(type = "body"))
ul <- unordered_list(level_list = c(1, 2, 3),
                      str_list = c("大項目", "中項目", "小項目"))
pp <- add_slide(pp)
pp <- ph_with(x = pp, value = ul, location = ph_location_type(type = "body"))
print(pp, target = path)
# shell.exec(path)

```



図 10.4: 箇条書きを追加したスライド

箇条書きを追加するとき, `level_list` と `str_list` の対応が正しくないと間違った箇条書きになってしまいます。この指定は面倒なので、文字列から自動的に対応を作ってくれる関数を定義します。

ここでは Markdown 記法と似た書式を使います。つまり、"-"を箇条書きの記号とし、"-"の数を箇条書きの水準とします。つまり上のコードの場合は、"-大項目", "--中項目", "---小項目"と表します。1つの文字列で書くときには、区切り文字を使って項目を分けます。区切り文字は、ふだん使わない文字であれば何でも構いませんが、既定値は";"とします。

コード 10.9 (powerpoint-str2ul-fun.R) : 文字列を箇条書きに変換する関数

```

str2ul <- function(str, sep = ";", symbol = "-"){
  if(length(str) == 1){                                # 1 つの文字列のとき
    str <-
    str |>
    stringr::str_split_1(pattern = sep) |> # 区切り文字で分割
    stringr::str_subset("^.+\$")                # 空文字("")以外
  }
  str_list <- stringr::str_remove(str, paste0("^", symbol, "*")) # 記号の除去
  level_list <-
  str |>

```

```

stringr::str_extract(paste0("^", symbol, "*")) |> # 記号の抽出
  stringr::str_count(symbol)                      # 箇条書きの水準
ul <- unordered_list(str_list = str_list,
                      level_list = level_list)
return(ul)
}

```

定義した関数を使うと、箇条書きを追加できます(図 10.5)。対応関係を気にしなくても楽に箇条書きを追加できます。

コード 10.10 (powerpoint-str2ul.R) : 文字列の箇条書きに変換してスライドに追加

```

pp <- add_slide(pp)
str <- c("-大項目;-中項目;-大項目;-中項目;-中項目;---小項目;---小項目")
ul <- str2ul(str, sep = ";")
pp <- ph_with(x = pp, value = ul, location = ph_location_type(type = "body"))
print(pp, target = path)
# shell.exec(path)

```

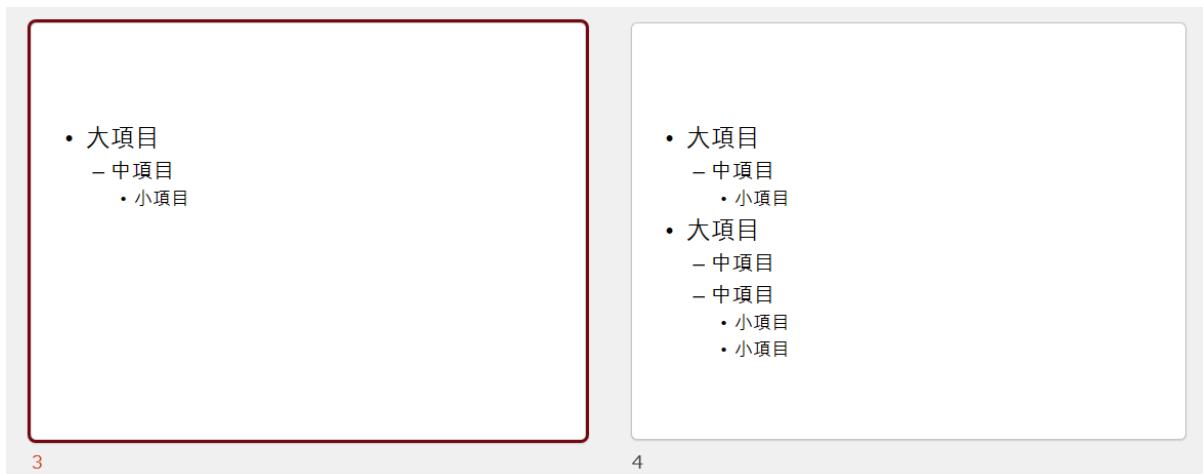


図 10.5: 関数を利用して箇条書きを追加したスライド

10.7 表の追加

`ph_with()`で表を追加することも可能です。"Title and Content"のレイアウトで新しいスライドを追加し、`iris` データの表を追加します。

コード 10.11 (powerpoint-ph-with-table.R) : 表の追加

```

layout <- "Title and Content"
pp <- add_slide(pp, layout = layout)
pp <- ph_with(pp, value = "みんな大好き iris データ",
              location = ph_location_type(type = "title"))

```

```

pp <- ph_with(pp, head(iris),
              location = ph_location_label(ph_label = "Content Placeholder 2"))
print(pp, target = path)
# shell.exec(path)

```

みんな大好きirisデータ

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

autofitで整形したirisデータ

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

5

6

個別に指定したirisデータ

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 10.6: 表を追加したスライド

データフレームそのままだと、あまり見た目がよくありません(図 10.6 の左上)。
`flextable::flextable()`で整形すると整形してスッキリできます。 `flextable` の `autofit()`を使うと表のデータに合わせた大きさになります(図 10.6 の右上)。`width()`, `height_all()`, `hrule()`を組み合わせて使うと、個別に幅や高さを指定できます(図 10.6 の左下)。

コード 10.12 (powerpoint-ph-with-flextable.R) : 整形した表の追加

```

# install.package("flextable") # 必要に応じてインストール
pp <- add_slide(pp, layout = layout)
# 表のデータに合わせる
pp <- ph_with(pp, value = "autofit で整形した iris データ",
              location = ph_location_type(type = "title"))
ft <- flextable::flextable(head(iris))
ft <- flextable::autofit(ft)
pp <- ph_with(pp, ft,
              location = ph_location_label(ph_label = "Content Placeholder 2"),
              )

```

```

# 個別に指定
pp <- add_slide(pp, layout = layout)
pp <- ph_with(pp, value = "個別に指定した iris データ",
              location = ph_location_type(type = "title"))
ft <- flextable::width(ft, width = 1.8)
ft <- flextable::height_all(ft, height = 0.75)
ft <- flextable::hrule(ft, rule = "exact", part = "all")
pp <- ph_with(pp, ft,
              location = ph_location_label("Content Placeholder 2")
            )
print(pp, target = path)
# shell.exec(path)

```

データの内容や行数・列数が定型の表であれば、`flextable`で個別に大きさを指定することで、いくつでも全く同じ大きさの表を作れます。定期的に作成する定型の表であれば自動化する利点が大きいでしょう。

表の内容や行数・列数が異なるときは、`officer`からの操作だけでは完全に希望どおりの大きさの表にすることは難しいので、細かな調整はパワーポイントでしてください。とはいえ、表をたくさん使うときは追加作業を自動化すれば作業が楽にできます。

10.8 画像の追加

`ph_with()`で、`png`などの画像ファイルを追加することができます。引数 `value` に `external_img("画像のパス")` として指定します。画像の大きさの指定方法は、若干複雑です。

`ph_with(use_loc_size = TRUE)` で指定すると、スライドや指定位置の大きさが基準になります。`use_loc_size = TRUE` だけだと、指定位置の大きさになります。追加で `location = ph_location_fullsize()` とすると、画像をスライド全体の大きさに合うように拡大・縮小します。

`ph_with(use_loc_size = FALSE)` とすると、画像の大きさが基準になり、`external_img()` で大きさを指定します。さらに、`guess_size = TRUE` とすると、画像のもとの大きさが使われます。`width` と `height` で数値を指定するとその大きさになります。指定する数値の単位の既定値はインチ(`unit = "in"`)ですが、`unit = "cm"` や `unit = "mm"` も使えます。

なお、画像を自動的に拡大・縮小するときには、縦横比は変化します。そのため、縦や横に間延びした図になることがあります。

次のコードを実行すると、スライド全体に合わせた画像(図 10.7 の左上)、指定位置全体の画像(右上)、画像のもとの大きさの画像(左下)、幅と高さを指定した画像(図 10.7 の右下)が追加されます。

コード 10.13 (powerpoint-ph-with-img.R) : スライドに画像を追加

```

url <- "https://matutosi.github.io/r-auto/data/r_gg.png"
path_img <- fs::path_temp("r_gg.png")
curl::curl_download(url, path_img) # url から PDF をダウンロード
# ph_location_fullsize() : スライド全体
pp <- add_slide(pp)
pp <- ph_with(pp,
              value = external_img(path_img),
              location = ph_location_fullsize())
pp <- ph_with(pp, value = "ph_location_fullsize() : \n スライド全体",
              location = ph_location_type(type = "title"))
# use_loc_size = TRUE : 指定位置全体
pp <- add_slide(pp)
pp <- ph_with(pp,
              value = external_img(path_img),
              location = ph_location_type(type = "body"), use_loc_size = TRUE)
pp <- ph_with(pp, value = "use_loc_size = TRUE : \n 指定位置全体", # \n で改行
              location = ph_location_type(type = "title"))
# guess_size = TRUE : 画像のもとの大きさ
pp <- add_slide(pp)
pp <- ph_with(pp,
              value = external_img(path_img, guess_size = TRUE),
              location = ph_location_type(type = "body"), use_loc_size = FALSE)
pp <- ph_with(pp, value = "guess_size = TRUE : \n 画像のもとの大きさ",
              location = ph_location_type(type = "title"))
# width と height で大きさ指定
pp <- add_slide(pp)
pp <- ph_with(pp,
              value = external_img(path_img, width = 5, height = 5),
              location = ph_location_type(type = "body"),
              use_loc_size = FALSE)
pp <- ph_with(pp, value = "width と height で\n大きさ指定",
              location = ph_location_type(type = "title"))
print(pp, target = path)
# shell.exec(path)

```

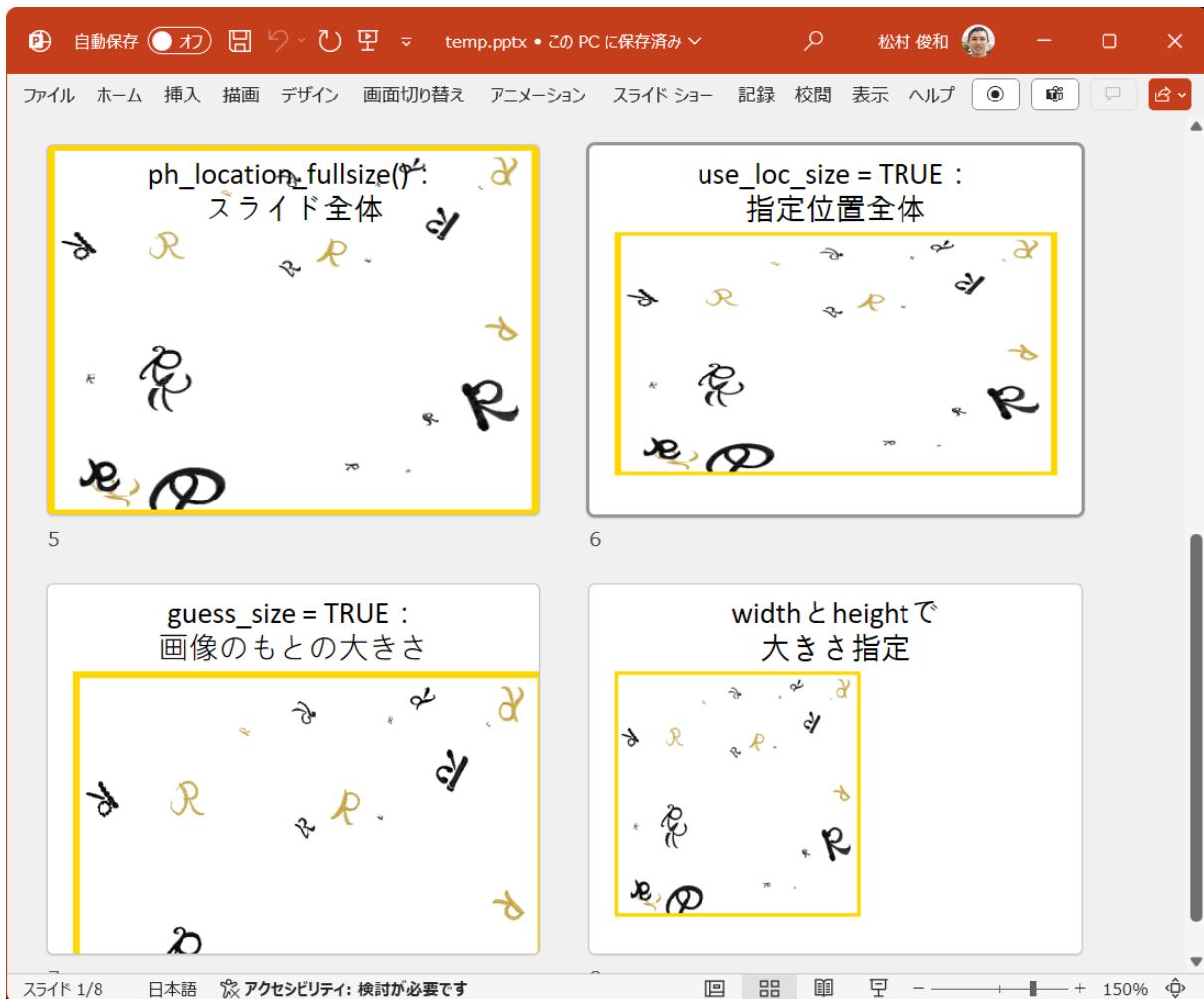


図 10.7: 図を追加したスライド

画像の大きさ指定の方法が分かりにくいので、大きさの異なる画像を使うときには不便です。そこで画像のサイズに合わせて画像を配置する関数を定義します。レイアウトは既定値でもある Title and Content とし、タイトルと画像のパスを指定します。Content の部分の幅と高さの大きさを超えないようにしつつ、縦横比を保ったまま画像を配置します。ただし、Content の部分はスライド全体の横幅よりも小さいので、スライド全体の横幅も使えるようにします。また、水平・垂直方法の中央揃えができるようにします。

関数の定義が長いので、詳細な説明は省略しますが、それほど難しいことはしていません。興味があればコメントを参考に確認する程度で結構です。

コード 10.14 (powerpoint-add-fig-fun.R) : Title and Content のレイアウトでタイトルと画像を挿入する関数

```
add_fig <- function(pp, title = "", path_img, fig_full_size = FALSE,
                     conter_horizontal = TRUE, conter_vertical = TRUE){
  # レイアウト・設置場所
  name <- "Title and Content"
  ph_label <- "Content Placeholder 2"
  # スライドのサイズ
  ss <- slide_size(pp)
```

```

# 配置場所のサイズ
cont_ph <-
  layout_properties(pp) |>
  dplyr::filter(name == {{name}} & ph_label == {{ph_label}})
if(fig_full_size){
  offx <- 0
}else{
  offx <- cont_ph$offx
}
offy <- cont_ph$offy
# 配置サイズ：全体 - offset
w_cont <- ss$width - offx * 2 # 幅, * 2:左右分
h_cont <- ss$height - offy # 高さ
# 画像のサイズ
img <- magick::image_read(path_img)
w_img <- magick::image_info(img)$width
h_img <- magick::image_info(img)$height
# 縦横比
ratio_img <- w_img / h_img # 画像
ratio_cont <- w_cont / h_cont # 配置場所
ratio <- ratio_img / ratio_cont # 画像と配置場所の比率
# 縦長・横長での補正
if(ratio > 1){
  h_cont <- h_cont / ratio # 図が横長
}else{
  w_cont <- w_cont * ratio # 図が縦長
}
# 補正
if(conter_horizontal){ # 水平方向
  offx <- (ss$width - w_cont) / 2
}
if(conter_vertical){ # 垂直方向
  offy <- (offy + ss$height - h_cont) / 2
}
# スライドの追加
pp <- add_slide(pp, layout = "Title and Content")
# 画像の追加
pp <- ph_with(pp,
              value = external_img(path_img),
              location = ph_location(left = offx, top = offy,
                                     width = w_cont, height = h_cont))
# タイトルの追加
pp <- ph_with(pp, value = title,
              location = ph_location_type(type = "title"))

```

```

    return(pp)
}

```

タイトルと画像を追加する関数を使います。横長と縦長の画像を使って、中央揃えの有無および左右の余白の有無の組み合わせでスライドを追加します。入力する引数の一覧をデータフレームに入れ、そのデータフレームを引数として使います。そのため、あらかじめ purrr パッケージの reduce() を拡張する preduce() を定義します。

コード 10.15 (powerpoint-preduce-fun.R) : purrr::reduce() をデータフレームに適用する糖衣関数

```

preduce <- function(.l, .f, ..., .init, .dir = c("forward", "backward")){
  .dir <- match.arg(.dir)
  purrr::reduce(
    purrr::transpose(.l),
    \((x, y){ rlang::exec(.f, x, !!!y, ...) },
    .init = .init, .dir = .dir)
}

```

次のコードを実行すると、設定に従って画像が追加されます(図 10.8)。

コード 10.16 (powerpoint-add-fig.R) : Title and Content のレイアウトでのタイトルと画像の挿入

```

# pp <- read_pptx()
imgs <- c("image_03_wide.jpg", "r_07_long.png")
urls <- paste0("https://matutosi.github.io/r-auto/data/", imgs)
path_imgs <- fs::path_temp(imgs)
curl::multi_download(urls, path_imgs) # url から PDF をダウンロード
## # A tibble: 2 × 10
##   success status_code resumefrom url           destfile error type
##   <lg1>     <int>      <dbl> <chr>        <chr>    <chr> <chr>
## 1 TRUE       200        0 https://matutos... "C:\\\\Us... <NA>  imag...
## 2 TRUE       200        0 https://matutos... "C:\\\\Us... <NA>  imag...
## # i 3 more variables: modified <dttm>, time <dbl>, headers <list>
wide <- path_imgs[1]
long <- path_imgs[2]
df <-
tibble::tribble(
  ~title          , ~path, ~conter_horiz , ~conter_vert , ~fig_full,
  "横長(全体)"    , wide , FALSE      , FALSE      , TRUE   ,
  "横長(全体, 中央)" , wide , TRUE      , TRUE      , TRUE   ,
  "横長(余白)"    , wide , FALSE      , FALSE      , FALSE  ,
  "横長(余白, 中央)" , wide , TRUE      , TRUE      , FALSE  ,
  "縦長(全体)"    , long , FALSE      , FALSE      , TRUE   ,

```

```

"縦長(全体, 中央)" , long , TRUE      , TRUE      , TRUE      ,
"縦長(余白)"       , long , FALSE     , FALSE     , FALSE     ,
"縦長(余白, 中央)" , long , TRUE      , TRUE      , FALSE
)
pp <- preduce(df, add_fig, .init = pp)
print(pp, target = path)
# shell.exec(path)

```



図 10.8: タイトルと図を追加したスライド

実際にスライドに画像を追加していくときは、タイトルの文字列と画像のパスのみを指定して、中央揃えと余白の設定は同じ設定を使うでしょう。そのときは、次のようにしてください。

コード 10.17 (powerpoint-add-fig-sample.R) : タイトルと画像のスライドの追加(擬似コード)

```

titles <- c("1枚目" , "2枚目" , "3枚目")
images <- c("01.png", "02.png", "03.png")
pp <- read_pptx()
pp <- reduce2(titles, images,           # タイトルと画像はスライドで異なる
              add_fig, .init = pp,
              fig_full_size = TRUE,    # 以下は全スライドで同じ
              center_horizontal = TRUE,
              center_vertical = TRUE)
print(pp, target = path)
# shell.exec(path)

```

《応用》 タイトルと画像のパスをエクセルや CSV 形式で保存しておけば、それをもとに自動的にパワーポイントを作り出すことができます。エクセルや CSV 形式からのデータの読み込みは、9.2 節を参照してください。

10.9 ggplot のグラフの追加

ggplot2(2.7 項を参照)で作成したグラフを追加することもできます。`ph_with()`で、`location = ph_location_fullsize()`とすればスライドの大きさ全体で、`location = ph_location_label("Content Placeholder 2")`とすれば指定位置の大きさでグラフを追加できます。いずれにしても、自動的にピッタリの大きさになるのが利点です。

さらに、rvg パッケージの `dml()` を用いれば、編集可能なグラフとして追加できます。パワーポイントでは、グラフのグループ化を解除すれば編集できます。

次のコードを実行すると、スライド全体の大きさと指定位置に合わせた大きさのグラフおよび編集可能なグラフを追加できます(図 10.9)。

コード 10.18 (powerpoint-ggplot.R) : ggplot のグラフを追加

```
# install.packages("rvg")
gg_iris <- # ggplot オブジェクト
iris |>
  ggplot2::ggplot(ggplot2::aes(Sepal.Length, Petal.Length, color = Species)) +
  ggplot2::geom_point(size = 3) +
  ggplot2::theme_minimal()
# スライドの最大サイズで追加
pp <- add_slide(pp)
pp <- ph_with(pp, "最大サイズでの ggplot の追加",
              location = ph_location_type(type = "title"))
pp <- ph_with(pp, gg_iris, location = ph_location_fullsize())
# 指定位置の大きさで追加
pp <- add_slide(pp)
pp <- ph_with(pp, "Placeholder 2 への追加",
              location = ph_location_type(type = "title"))
pp <- ph_with(pp, gg_iris,
              location = ph_location_label("Content Placeholder 2"))
# 編集可能な図として追加
editable_gg <- rvg::dml(ggobj = gg_iris)
pp <- add_slide(pp)
pp <- ph_with(pp, "編集可能な図として追加",
              location = ph_location_type(type = "title"))
pp <- ph_with(pp, editable_gg,
              location = ph_location_label("Content Placeholder 2"))
# パワーポイントの保存
print(pp, target = path)
# shell.exec(path)
```

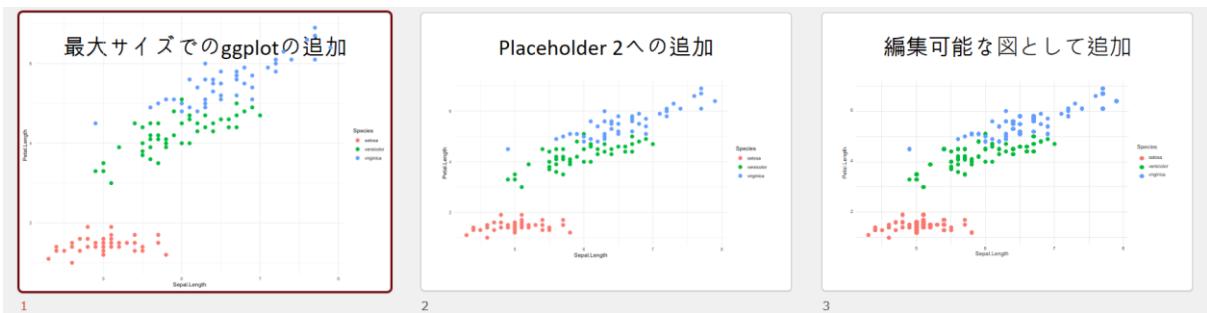


図 10.9: ggplot のグラフを追加したスライド

10.10 概要の表示

パワーポイントの概要を表示するには、`pptx_summary()`を使います。

コード 10.19 (powerpoint-summary.R) : パワーポイントの概要表示

```
library(officer)
pp |>
  pptx_summary() |>
  tibble::tibble()
## # A tibble: 330 × 9
##   text    id  content_type slide_id row_id cell_id col_span row_span
##   <chr>  <chr> <chr>        <int>  <int>  <int>     <int>     <int>
## 1 R によ… 2   paragraph      1     NA     NA     NA     NA
## 2 手作… 3   paragraph      1     NA     NA     NA     NA
## 3 手作… 3   paragraph      1     NA     NA     NA     NA
## 4 手作… 3   paragraph      1     NA     NA     NA     NA
## 5 手作… 3   paragraph      1     NA     NA     NA     NA
## 6 手作… 3   paragraph      1     NA     NA     NA     NA
## 7 自動… 4   paragraph      1     NA     NA     NA     NA
## 8 自動… 4   paragraph      1     NA     NA     NA     NA
## 9 自動… 4   paragraph      1     NA     NA     NA     NA
## 10 自動… 4  paragraph      1     NA     NA     NA     NA
## # ℥ 320 more rows
## # ℥ 1 more variable: media_file <chr>
```

`content_type` には、"paragraph", "image", "table cell"があり、それぞれ文字列、画像、表のデータです。 文字列のデータを抽出し、関係のある列のみを選択するには次のようにします。

コード 10.20 (powerpoint-summary-text.R) : 文字列のデータ

```
pptx_summary(pp) |>
  dplyr::filter(content_type == "paragraph") |>
  tibble::tibble()
## # A tibble: 211 × 9
##   text    id  content_type slide_id row_id cell_id col_span row_span
```

```

##   <chr> <chr> <chr>      <int> <int> <int> <int> <int>
## 1 R によ... 2     paragraph      1     NA     NA     NA     NA
## 2 手作... 3     paragraph      1     NA     NA     NA     NA
## 3 手作... 3     paragraph      1     NA     NA     NA     NA
## 4 手作... 3     paragraph      1     NA     NA     NA     NA
## 5 手作... 3     paragraph      1     NA     NA     NA     NA
## 6 手作... 3     paragraph      1     NA     NA     NA     NA
## 7 自動... 4     paragraph      1     NA     NA     NA     NA
## 8 自動... 4     paragraph      1     NA     NA     NA     NA
## 9 自動... 4     paragraph      1     NA     NA     NA     NA
## 10 自動... 4    paragraph      1     NA     NA     NA     NA
## # i 201 more rows
## # i 1 more variable: media_file <chr>

```

10.11 文字列の取り出し

パワーポイントから文字列を取り出すには、`pptx_summary()`を利用して、その中から、`content_type == "paragraph"`を抽出し、`text`列を選択します。

単純にすべての文字列を取り出すだけならこれで十分です。より便利にするために、パワーポイントのパスからスライドごとの文字列を取り出す関数を作成します。`tidyverse`と`dplyr`の関数は、それぞれ2.4節と2.5節を参照してください。

コード 10.21 (`extract-pp-text-fun.R`) : パワーポイントから文字列を取り出す関数

```

extract_pp_text <- function(path){
  paragraph <-
    path |>
    read_pptx() |>
    pptx_summary() |>
    dplyr::filter(content_type == "paragraph") |> # 文字列のみ
    dplyr::filter(text != "") |> # 空を除去
    dplyr::select(slide_id, text) #
  text <-
    paragraph |>
    dplyr::mutate(dammy = "text") |> # pivot_wider()で使うダミー列
    tidyverse::pivot_wider(id_cols = slide_id, # スライドごとに
                           names_from = dammy,
                           values_from = text, # 文字列を
                           values_fn = list) |> # リストに
    `\$`(_, "text") # 文字列を取り出し
  return(text)
}

```

作った関数でスライドごとに文字列を取り出せます。

コード 10.22 (extract-pp-text.R) : パワーポイントからの文字列の取り出し

```
extract_pp_text(path) |>
  head(3)
## [[1]]
## [1] "R による自動化の方法" "手作業 1"           "手作業 2"
## [4] "手作業 3"          "手作業 4"           "手作業 5"
## [7] "自動化 1"          "自動化 2"           "自動化 3"
## [10] "自動化 4"         "自動化 5"
##
## [[2]]
## [1] "みんな大好き iris データ"
##
## [[3]]
## [1] "autofit で整形した iris データ"
```

10.12表の取り出し

パワーポイントから文字列を取り出すには、`pptx_summary()`を利用して、そのなかから、`content_type == "table cell"`を抽出します。`id` がオブジェクトの番号、`row_id` が行番号、`cell_id` が列番号、`text` が表の中の文字列です。

コード 10.23 (powerpoint-summary-table.R) : 表のデータ

```
pptx_summary(pp) |>
  dplyr::filter(content_type == "table cell") |>
  dplyr::transmute(id, row_id, cell_id,
                    text = stringr::str_squish(text)) |> # 余分な空白文字を除去
  tibble::tibble()
## # A tibble: 105 × 4
##       id   row_id cell_id text
##   <chr> <int>    <int> <chr>
## 1 3     1        1 Sepal.Length
## 2 3     2        1 5.1
## 3 3     3        1 4.9
## 4 3     4        1 4.7
## 5 3     5        1 4.6
## 6 3     6        1 5.0
## 7 3     7        1 5.4
## 8 3     1        2 Sepal.Width
## 9 3     2        2 3.5
## 10 3    3        2 3.0
## # i 95 more rows
```

表番号ごとに、行と列に整理する関数を定義します。ここでは、`pivotea` パッケージ(9.5 節を参照)の `pivot()`を使っていますが、`tidyverse::pivot_wider()`を使っても同様のことができます。なお、`id` はスライド間での重複がありますので、`c("id", "slide_id")`としています。

コード 10.24 (powerpoint-extract-pp-table-fun.R) : パワーポイントから表のデータを取り出す関数

```
extract_pp_table <- function(path){  
  table <-  
    path |>  
    read_pptx() |>  
    pptx_summary() |>  
    dplyr::filter(content_type == "table cell") |>  
    pivot长:pivot(row = "row_id", col = "cell_id",  
                  value = "text", split = c("id", "slide_id"))  
  return(table)  
}
```

定義した関数で表を取り出します。

コード 10.25 (powerpoint-extract-pp-table.R) : パワーポイントからの表のデータの取り出し

```
extract_pp_table(path) |>  
  head(1) # 3つの表の内容は同じなので2つ目以降は省略  
## #`3.2`  
## # A tibble: 7 × 8  
##   id   slide_id row_id `1`      `2`      `3`      `4`      `5`  
##   <chr>    <int>  <int> <chr>    <chr>    <chr>    <chr>    <chr>  
## 1 3         2     1 Sepal.Length Sepal.Width Petal.Length Petal.Width Spec...  
## 2 3         2     2 5.1       3.5       1.4       0.2       seto...  
## 3 3         2     3 4.9       3.0       1.4       0.2       seto...  
## 4 3         2     4 4.7       3.2       1.3       0.2       seto...  
## 5 3         2     5 4.6       3.1       1.5       0.2       seto...  
## 6 3         2     6 5.0       3.6       1.4       0.2       seto...  
## 7 3         2     7 5.4       3.9       1.7       0.4       seto...
```

10.13画像ファイルの取り出し

パワーポイントから画像ファイルを取り出すには、`pptx_summary()`を利用してから、`content_type == "image"`で画像データを抽出し、`media_file`列のデータを使います。これで、画像の一覧を取り出せます。

コード 10.26 (powerpoint-summary-image.R) : 画像データの一覧の取り出し

```
pptx_summary(pp) |>  
  dplyr::filter(content_type == "image") |>  
  `\$(`_, "media_file") |>
```

```

head( )
## [1] "ppt/media//fa9fad457b9190efdef97bebfbb15c9d2ba3e25c.png"
## [2] "ppt/media//fa9fad457b9190efdef97bebfbb15c9d2ba3e25c.png"
## [3] "ppt/media//fa9fad457b9190efdef97bebfbb15c9d2ba3e25c.png"
## [4] "ppt/media//fa9fad457b9190efdef97bebfbb15c9d2ba3e25c.png"
## [5] "ppt/media//d3e14ce37547fd8e0656e9a781e2915a100c2f0c.jpg"
## [6] "ppt/media//d3e14ce37547fd8e0656e9a781e2915a100c2f0c.jpg"

```

上のコードの画像ファイルが保存されているディレクトリは、`$package_dir`で確認できます。

コード 10.27 (powerpoint-package-dir.R) : パワーポイントのファイルのディレクトリ

```

fs::path(pp$package_dir)
## C:/Users/USERNAME/AppData/Local/Temp/RtmpuQjLTs/file4b4c55f93f3a

```

`officer` には `media_extract()` という関数がありますが、パワーポイントのオブジェクトを引数にしてすぐに取り出せるわけではありません。そこで上記の情報をもとにパワーポイントから画像を取り出す関数を定義します。

`path` にパワーポイントのファイルを、`out_dir` に出力ディレクトリを指定します。パワーポイントのファイル名から、拡張子を除去した名称のディレクトリを作成し、その中に連番の画像を保存します。ディレクトリやファイルの操作の関数は第1章を参照してください。

コード 10.28 (powerpoint-extract-pp-image-fun.R) : パワーポイントから画像データを取り出す関数

```

extract_pp_image <- function(path, out_dir = NULL, overwrite = TRUE){
  pp <- officer::read_pptx(path)                                # 読み込み
  imgs <- officer::pptx_summary(pp) |>                         # 画像の一覧
  dplyr::filter(content_type == "image")                         # 概要の取得
  slide_id <- imgs$slide_id |>                                 # 画像のみ
  # スライド id
  stringr::str_pad(width = 2, side = "left", pad = "0")        # 衔合わせ
  image_files <- fs::path(pp$package_dir, imgs$media_file)    # 画像ファイル
  image_exts <- fs::path_ext(image_files)                         # 画像の拡張子
  pp_file <-
    path |>
    fs::path_file() |>                                         # ディレクトリ除去
    fs::path_ext_remove()                                     # 拡張子除去
  out_files <-
    seq_along(image_files) |>                               # 連番のファイル名
    stringr::str_pad(width = 2, side = "left", pad = "0")    # 連番
  out_files <-
    paste0(slide_id, "_", out_files) |>                      # 衔合わせ
    fs::path_ext_set(image_exts)                             # スライド番号追加
  # 拡張子の設定
  if(is.null(out_dir)){

```

```

out_dir <- fs::path_temp(pp_file) # 一時ディレクトリ
} else{
  out_dir <- fs::path(out_dir, pp_file)
}
out_files <- fs::path(out_dir, out_files)
fs::dir_create(out_dir)
fs::file_copy(image_files, out_files, overwrite = overwrite)
return(out_files)
}

```

次のコードを実行するとパワーポイントから画像を取り出して、デスクトップ上のディレクトリに保存されます(図 10.10)。

コード 10.29 (powerpoint-extract-pp-image.R) : パワーポイントからの画像データの取り出し

```

out_dir <- fs::path_home("desktop")
extract_pp_image(path, out_dir, overwrite = TRUE)

```

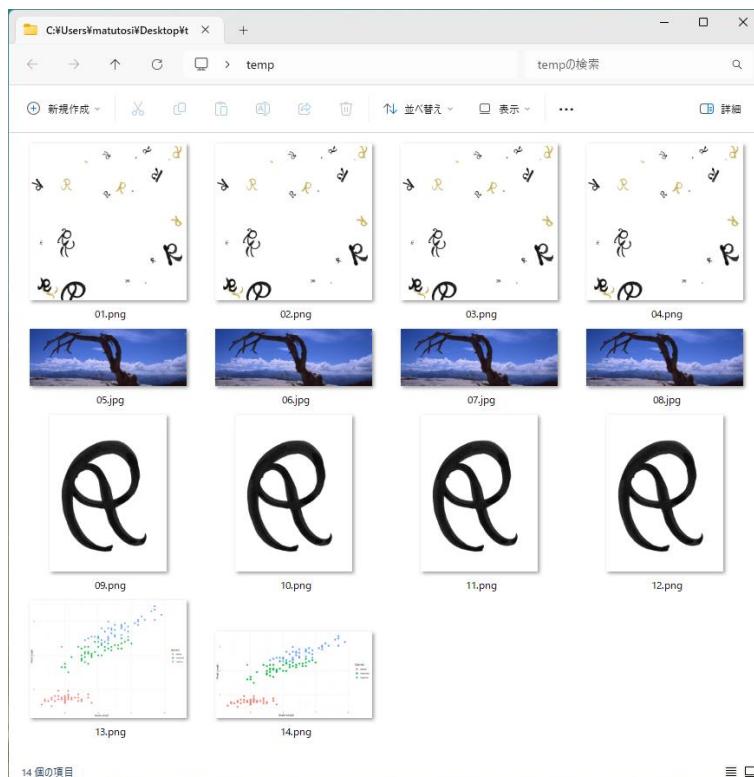


図 10.10: パワーポイントから取り出した画像

10.14 画像・PDF・動画への変換

RDCOMClient パッケージを使うと、パワーポイントを操作できます。RDCOMClient の COMCreate() で作成したオブジェクトのメソッド(関数)で、ファイルを開いたり保存したりできます。これを

を利用して、パワーポイント自体を画像ファイルに変換する関数を定義します。画像だけでなくPDFや動画への変換も可能です。

なお、RDCOMClientは8.1節を参考にしてインストールしてください。また、パワーポイントがインストールされている必要があります。

コード10.30 (powerpoint-pp2img-fun.R) : パワーポイントを画像・PDF・動画に変換する関数

```
pp2ext <- function(path, format = "png"){
  format_no <- switch(format,
    ppt = 1, rtf = 5, ppx = 11, ppsx = 28, pdf = 32,
    wmf = 15, gif = 16, jpg = 17, png = 18, bmp = 19,
    tif = 21, emf = 23,
    wmv = 37, mp4 = 39)
  path <- normalizePath(path)                                # Windows形式
  converted <-
    fs::path_ext_remove(path) |>                            # 拡張子除去
    normalizePath(mustWork = FALSE)
  ppApp <- RDCOMClient::COMCreate("PowerPoint.Application") # パワーポイント
  ppApp[["DisplayAlerts"]] <- FALSE                         # 警告を非表示
  pp <- ppApp[["Presentations"]]$Open(path)                # ファイルを開く
  pp$SaveAs(converted, FileFormat = format_no)             # 指定形式で保存
  pp$Close()
  cmd <- "taskkill /f /im powerpnt.exe"                   # 終了コマンド
  system(cmd)                                              # コマンド実行
  if(format %in% c("ppt", "rtf", "ppx", "ppsx", "pdf")){  # 拡張子の設定
    converted <- fs::path_ext_set(converted, format)
  }
  return(converted)
}
```

"png"などの画像を指定すると、パワーポイントと同じディレクトリにパワーポイントの拡張子を除いた名称のディレクトリが自動的に作成され、その中にスライド1.pngなどのファイル名で保存されます(10.11)。

コード10.31 (powerpoint-pp2img-png.R) : パワーポイントのpngへの変換

```
library(RDCOMClient) # 最初は呼び出さないとエラーになる
pp2ext(path, format = "png")
```

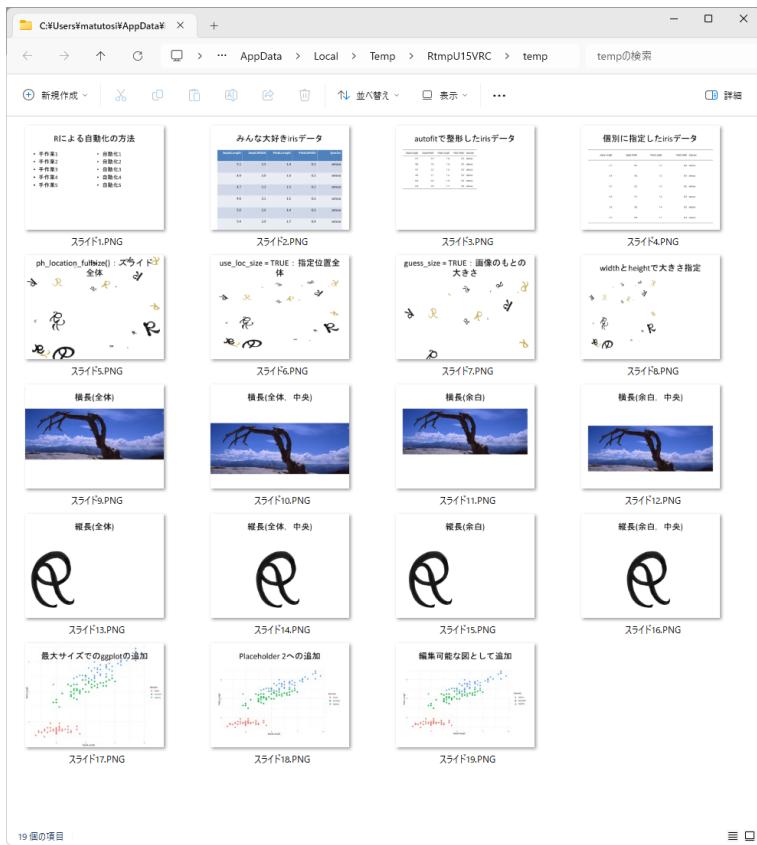


図 10.11：パワーポイントから変換した png

"pdf"を指定するとパワーポイントと同じディレクトリに PDF が保存されます(10.12).

コード 10.32 (powerpoint-pp2img-pdf.R) : パワーポイントの PDF への変換

```
pp2ext(path, format = "pdf")
```

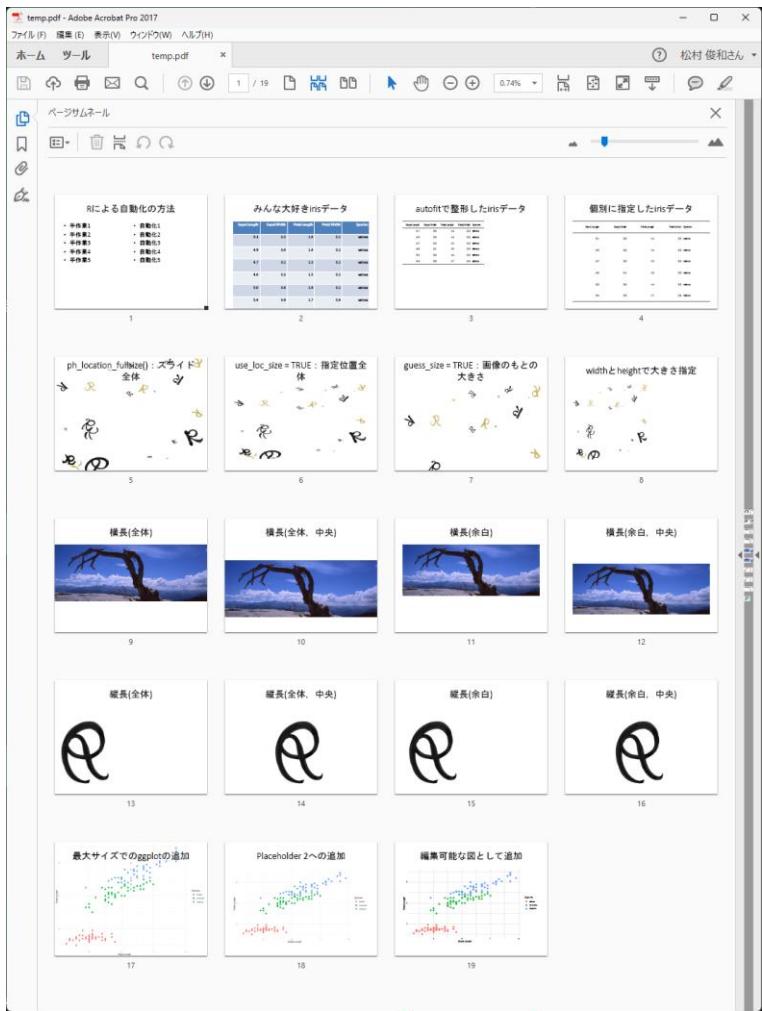


図 10.12: パワーポイントから変換した PDF

"mp4"あるいは"wmv"を指定すると、動画へ変換されます(10.13)。スライドショーのリハーサルでスライドごとの時間を保存していると、その設定が使われます。動画への変換には少し時間がかかります。また、パワーポイントのポップアップウィンドウが表示されますので、それをクリックする必要があります。

コード 10.33 (powerpoint-pp2img-mp4.R) : パワーポイントの mp4 への変換

```
pp2ext(path, format = "mp4") # 時間がかかる、ポップアップのクリックが必要_
```

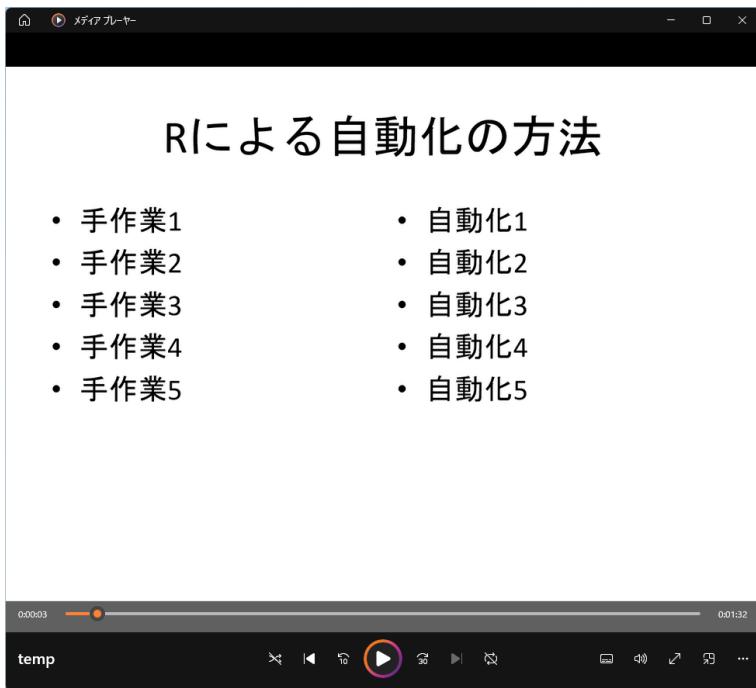


図 10.13: パワーポイントから変換した mp4

《応用》 magick パッケージ(第 11 章)の関数の `image_append()`を活用すれば、画像に変換した画像を縮小版として 1 ページに並べられます。パワーポイントで縮小版として保存することはできますが、ページ間の余白がやや広いです。関数を作成すれば、余白の設定などを好みの大きさにできます。具体的なコードは、11.19 節を参考にしてください。

【相談】 具体的な関数があった方が良ければ、入れますが、いかがでしょうか。

II 画像の操作

【相談】 画像操作は節の数が多くなってしました。統合したほうが良ければ、次の[]で囲ったところを統合しようかと思っています。 [1: 読み込み + 形式変換・書き込み] [2: 画像の描画 + 情報の表示] [3: 枠(余白)の追加 + 背景の塗りつぶし + 余白の除去] [4: 傾き追加・補正 + 画像内の文字認識]

スマートフォンの内蔵カメラやデジタルカメラの機能向上のおかげで、きれいな写真を簡単に撮れるようになりました。それにともなって、写真のファイルサイズも大きくなっています。一方、ウェブページ、ワード、印刷資料では、それほど高画質な画像は求められません。写真などの画像のファイルサイズが大きすぎると電子メールでエラーになったり、時間がかかったりします。少数の画像であれば変換ソフトを使って手作業で縮小すれば良いですが、たくさんの画像になると大変です。ワードなどにも画像の圧縮機能はありますが、手作業が必要です。

magick パッケージを使えば、ファイルの数にかかわらずファイルサイズの縮小を自動的にすることができます。その際、求めるファイルサイズ以下でなるべく画質の良い画像を作成することもできます。また、ファイルサイズの縮小だけでなく、ぼかしなどの加工、トリミング、重ね合わせ、資料画像への出典の記載、ファイル形式の変換などもできます。自動化すれば、全く同じ範囲での切り取りや同じサイズでの出力が可能ですので、仕上がりが統一できる点も利点です。さらに、これらの画像操作をもとにすれば、条件にあう画像のみの一覧をつくることができます。

傾き補正や工学的文字認識(OCR)もできます。そのため、紙媒体でしか入手できなかったデータをスキャンして画像データにすれば、手入力しなくてもテキストデータを入手できます。100%の精度ではなくても、手入力よりは手早く作業が完了するでしょう。

II.1 画像を操作するパッケージ

画像の操作に使用する magick パッケージのインストールをしてから呼び出します。svg 形式の画像を読み込むには、rsvg パッケージもインストールします。

コード II.1 (image-install.R) : magick のインストール

```
install.packages("magick")
install.packages("rsvg")
```

コード II.2 (image-library.R) : magick の呼び出し

```
library(magick)
## Linking to ImageMagick 6.9.12.98
## Enabled features: cairo, freetype, fftw, ghostscript, heic, lcms, pango, raw, rsvg, webp
## Disabled features: fontconfig, x11
# library(rsvg) # svg 形式を使うとき
```

11.2 作業用の画像ファイルの用意

本章で操作する作業用の画像ファイルを用意します。

次のコード内の URL からダウンロードできますが、読者自身の画像でも構いません。余白の除去や文字の追加では、単純なイラストの方が作業の内容がよくわかります。

コード 11.3 (image-download.R) : 画像のダウンロード

```
rs <- paste0("r_", stringr::str_pad(1:24, 2, "left", "0"), ".png")
pts <- paste0("image_", c("01", "02", "03"), ".jpg")
files <- c(pts, rs)
urls <- paste0("https://matutosi.github.io/r-auto/data/", files)
files <- fs::path_temp(files)
curl::multi_download(urls, files)
## Download status: 0 done; 27 in progress (0 b/s). Total size: 14.80...
## Download status: 9 done; 18 in progress (166.91 Kb/s). Total size:...
## (以下省略)
```

11.3 読み込み

png, jpeg, gif 形式の画像には `image_read()`, svg 形式には `image_read_svg()`, PDF 形式には `image_read_pdf()` をそれぞれ用います。svg と PDF は読み込むと png に変換されます。

ファイルは 1 つでも複数でも読み込み可能です。`image_read()` で読み込んだオブジェクトは、`magick` の関数でそのまま使えます。なお、個別の画像を取り出すには、`[]` を使います。

コード 11.4 (image-read.R) : 画像の読み込み

```
imgs <- image_read(files)
imgs
## # A tibble: 27 × 7
##   format width height colorspace matte filesize density
##   <chr>    <int>   <int> <chr>     <lgl>    <int> <chr>
## 1 JPEG      500     400 sRGB      FALSE    33247 72x72
## 2 JPEG      500     400 sRGB      FALSE    48184 600x600
## 3 JPEG      500     400 sRGB      FALSE    54705 180x180
## 4 PNG       351     296 Gray     FALSE    22722 118x118
## (以下省略)
imgs[1] # 個別の画像の取り出し
## # A tibble: 1 × 7
##   format width height colorspace matte filesize density
##   <chr>    <int>   <int> <chr>     <lgl>    <int> <chr>
## 1 JPEG      500     400 sRGB      FALSE    33247 72x72
```

11.4 画像の描画

画像オブジェクトは `plot()` で描画できます。読み込んだ画像は複数含まれますので、1つだけを描画するには `[]` を使います。

コード 11.5 (`image-plot-single.R`) : 複数画像の描画

```
plot(imgs[1])
```



図 11.1: 画像の描画

複数画像を一度に表示するには、少し工夫が必要です。ここでは描画パネルを分割するとともに、リストに変換してから `walk()` を使います。

コード 11.6 (`image-plot-multiple.R`) : 複数画像の描画

```
par(mfrow = c(3,9)) # 描画パネルの分割
par(mar = rep(0, 4))
par(oma = rep(0, 4))
imgs |>
  as.list() |> # walk()を使うためリストに変換
  purrr::walk(plot) # 繰り返し
```

œ R R R R œ R œ R

R R R R œ R œ R œ R

R R œ R œ R œ R



図 11.2: 複数画像の描画

`plot()`や他のパッケージの関数を使うときは、`as.list()`でリストに変換して操作することができます。リストに変換したものは、`image_join()`でもとの画像オブジェクトに戻すことができます。

コード 11.7 (`image-join.R`) : リストへの変換と画像オブジェクトに戻す操作

```
imgs |> as.list()
## [[1]]
## # A tibble: 1 × 7
##   format width height colorspace matte filesize density
##   <chr>   <int>  <int> <chr>     <lgl>   <int> <chr>
## 1 JPEG     500    400 sRGB      FALSE    33247 72x72
## [[2]]
## # A tibble: 1 × 7
##   format width height colorspace matte filesize density
##   <chr>   <int>  <int> <chr>     <lgl>   <int> <chr>
## 1 JPEG     500    400 sRGB      FALSE    48184 600x600
## [[3]]
## # A tibble: 1 × 7
##   format width height colorspace matte filesize density
##   <chr>   <int>  <int> <chr>     <lgl>   <int> <chr>
## 1 JPEG     500    400 sRGB      FALSE    54705 180x180
## (以下省略)
imgs |> as.list() |> image_join()
## # A tibble: 27 × 7
##   format width height colorspace matte filesize density
##   <chr>   <int>  <int> <chr>     <lgl>   <int> <chr>
## 1 PNG     500    400 sRGB      FALSE     0 72x72
## 2 PNG     500    400 sRGB      FALSE     0 600x600
## 3 PNG     500    400 sRGB      FALSE     0 180x180
## 4 PNG     351    296 Gray     FALSE     0 118x118
## 5 PNG     805   1159 Gray     FALSE     0 72x72
## 6 PNG     905   1071 Gray     FALSE     0 72x72
## (以下省略)
```

11.5 情報の表示

画像の情報を表示するには、`image_info()`を使います。引数には画像オブジェクトを指定します。なお、`magick`パッケージの多くの関数では、第1引数に画像オブジェクトを指定します。

コード 11.8 (`image-info.R`) : 画像の情報表示

```
image_info(imgs)
## # A tibble: 27 × 7
##   format width height colorspace matte filesize density
##   <chr>   <int>   <int> <chr>     <lgl>   <int> <chr>
## 1 PNG      351     296 Gray      FALSE    22722 118x118
## 2 PNG      805    1159 Gray      FALSE   123359 72x72
## 3 PNG      905    1071 Gray      FALSE   108596 72x72
## 4 PNG      751    1027 Gray      FALSE   119686 118x118
## 5 PNG      860    1111 Gray      FALSE   137950 118x118
## 6 PNG      751     997 Gray      FALSE   100472 118x118
## 7 PNG      722     967 Gray      FALSE   100523 118x118
## 8 PNG      661     884 Gray      FALSE   107154 118x118
## 9 PNG      631     982 Gray      FALSE   107334 118x118
## 10 PNG     653     942 Gray      FALSE   87457 118x118
## # i 17 more rows
image_info(imgs[1])
## # A tibble: 1 × 7
##   format width height colorspace matte filesize density
##   <chr>   <int>   <int> <chr>     <lgl>   <int> <chr>
## 1 PNG      351     296 Gray      FALSE    22722 118x118
```

`format`に画像の形式が表示されており、`imgs`には27の画像が含まれていることがわかります。さらに、画像の形式、幅、高さの情報は、それぞれ\$format, \$width, \$heightで取り出せます。

コード 11.9 (`image-info-details.R`) : 画像の高さ

```
image_info(imgs)$height
## [1] 296 1159 1071 1027 1111 997 967 884 982 942 974 907 942
## [14] 911 971 963 924 966 524 488 403 416 483 421 960 1536
## [27] 1829
```

11.6 形式変換・書き込み

`image_convert()`で画像の形式変換ができます。対応する形式はpng, jpeg, gif, pdfで、相互に変換可能です。画像形式は、`format`で指定します。また、`type = "grayscale"`で白黒画像に変換できます。

コード 11.10 (`image-convert.R`) : 画像の変換

```
image_convert(imgs, format = "jpeg")
## # A tibble: 27 × 7
```

```

##   format width height colorspace matte filesize density
##   <chr>  <int>  <int> <chr>      <lgl>    <int> <chr>
## 1 JPEG    500    400 sRGB     FALSE      0 72x72
## 2 JPEG    500    400 sRGB     FALSE      0 600x600
## 3 JPEG    500    400 sRGB     FALSE      0 180x180
## 4 JPEG    351    296 Gray    FALSE      0 118x118
## 5 JPEG    805   1159 Gray    FALSE      0 72x72
## 6 JPEG    905   1071 Gray    FALSE      0 72x72
## (以下省略)

```

変換すると、formatが全てjpegになっていることがわかります。

画像を書き込むには、`image_write()`を使います。この関数の引数にもformatがあり、`image_write()`で書き込む際にも形式を変換できます。`image_write()`の引数pathの拡張子とformatは連動していません。形式を変換するときには、必ずformatで指定してください。

コード II.11 (image-write.R) : 画像の書き込み

```

image_write(imgs[1], path = fs::path_temp("r_01.pdf"), format = "pdf")
image_write(imgs[2], path = fs::path_temp("r_02.png"))

```

`image_write()`では複数画像の書き込みはできません。複数を一気に書き込むには、`purrr::walk2()`かforループを使います。次のコードは`walk2()`を使った場合です。

コード II.12 (image-write-multi.R) : 複数画像の書き込み

```

len <- length(imgs)
path_alpha <- fs::path_temp(paste0(letters[seq(len)], ".png"))
imgs |>
  as.list() |> # walk()を使うためリストに変換
  purrr::walk2(path_alpha, image_write, format = "png")

```

複数画像を保存したいときに、`walk2()`やforループを使うのは面倒です。そこで複数画像の保存に対応する関数を作成します。ついでに、拡張子に連動して自動的にファイル形式を選択して、保存するようにします。

作成する関数の要点は次の2つです。

- `fs::path_ext()`で拡張子を取得する。
- `pwalk()`を使うために、データフレームの列名を`image_write()`の引数と合わせる。

コード II.13 (image-write-images-fun.R) : 複数画像を拡張子の形式で書き込む関数

```

images_write <- function(images, paths){
  formats <- fs::path_ext(paths) # 拡張子の形式
  tibble::tibble(image = as.list(images), path = paths, format = formats) |>

```

```

    purrr::pwalk(magick::image_write)
}

```

関数を実行すると、複数の画像を一気に保存できます。

コード 11.14 (image-write-images.R) : 複数画像を拡張子の形式で書き込む

```

exts <- rep(c("png", "jpg", "gif"), 9)
imgs_path <-
  files |>
  fs::path_file() |>      # ファイル名のみ
  fs::path_ext_set(exts) |> # 拡張子の変更
  fs::path_temp() |>       # 一時ディレクトリ
  print()
## C:/Users/USERNAME/AppData/Local/Temp/RtmpsspDvVe/r_01.png
## C:/Users/USERNAME/AppData/Local/Temp/RtmpsspDvVe/r_02.jpg
## C:/Users/USERNAME/AppData/Local/Temp/RtmpsspDvVe/r_03.gif
## C:/Users/USERNAME/AppData/Local/Temp/RtmpsspDvVe/r_04.png
## (以下省略)
images_write(imgs, imgs_path)
# imgs_path |> purrr::map(shell.exec) # 画像を表示

```

11.7 枠(余白)の追加

画像に枠(余白)をつけるには、`image_border()`を使います。引数 `color` には枠の色を、`geometry` には枠の位置と大きさをそれぞれ指定します。`geometry = "10"`は左右に、`geometry = "x10"`は上下に 10 ピクセルの枠を追加します。`geometry = "40x10"`は左右に 40 ピクセル、上下に 10 ピクセルの枠を追加します。

コード 11.15 (image-border.R) : 枠(余白)の追加

```

bordered_01 <- image_border(imgs[1], color = "gold", geometry = "40x40")
bordered_25 <- image_border(imgs[25], color = gray(0.1), geometry = "x300")
par(mfrow = c(1,2)) # 描画パネルの分割
par(mar = rep(0, 4))
par(oma = rep(0, 4))
plot(bordered_01); plot(bordered_25)

```

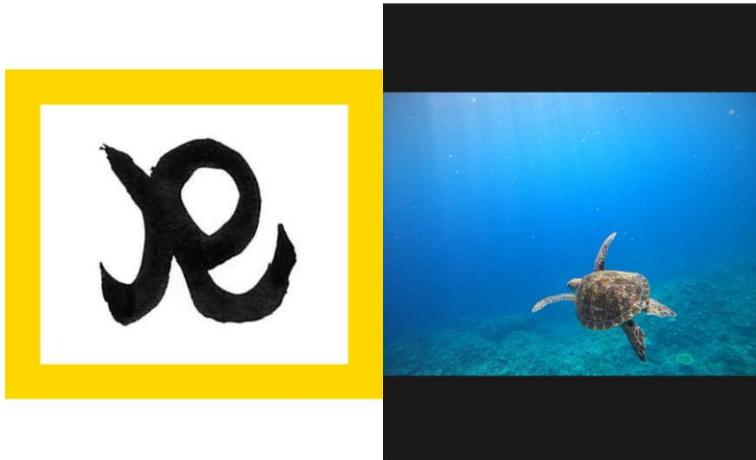


図 11.3: 枠(余白)を追加した画像

11.8 文字の追加

【相談というか連絡事項】 2024-07-02 時点では、`magick_fonts()`は GitHub での開発版では実装されていますが、CRAN のバージョンではまだ実装されていません。おそらく次のバージョニアップ時には、正式に反映されると思います。原稿の完成時点の状況に応じて、詳しい説明をどうするか考えたいと思います。

画像の上に註釈などの文字を追加するときには、`image_annotate()`を使います。引数 `text` に追加する文字列を、`size` に文字サイズを、`font` にフォントをそれぞれ指定します。起点の位置 `location`、角度 `degrees`、文字サイズ `size` なども指定できます。日本語で文字を追加するときには、`font` は必須です。使用できるフォントは、`magick_fonts()$family` で確認します。

コード 11.16 (`image-font.R`) : 使用できるフォントの例

```
# install.packages("magick", repos = "https://ropensci.r-universe.dev")
fonts <-  
  magick_fonts()$family |>  
  stringr::str_subset("Meiryo|Yu") |>  
  unique() |>  
  print()  
## [1] "Meiryo & Meiryo Italic & Meiryo UI & Meiryo UI"  
## [2] "Meiryo Bold & Meiryo Bold Italic & Meiryo UI Bold & Meiryo UI"  
## [3] "Yu Gothic Bold & Yu Gothic UI Semibold & Yu Gothic UI"  
## [4] "Yu Gothic Light & Yu Gothic UI"  
## [5] "Yu Gothic Medium & Yu Gothic UI"  
## [6] "Yu Gothic Regular & Yu Gothic UI Semilight"  
## [7] "Yu Mincho"
```

上のコードでは、筆者の環境で使用できるフォントを示しています。`character(0)`と表示されるときは、`magick_fonts()$family` でフォントの一覧を表示し、使えそうなフォントを探してください。

コード 11.17 (image-annotate-1.R) : 文字の追加

```
annotated_01 <-  
  image_annotate(bordered_01, "Rで自動化", size = 30, font = fonts[3])  
plot(annotated_01)
```



図 11.4: 文字を追加した画像

引数 `gravity` で地図に見立てた東西南北の方角で位置合わせの場所を、`location` で起点からの距離を指定できます。たとえば、`gravity = "southeast"` のときは南東つまり右下を起点とします。`location = "+50+30"` とすると、起点と文字列との距離が x 軸で 50 ピクセル、y 軸で 30 ピクセルになります。

コード 11.18 (image-annotate-2.R) : 文字の追加

```
annotated_25 <-  
  bordered_25 |>  
  image_annotate("宮古島の\nアオウミガメ",  
    gravity = "southeast", location = "+50+30",  
    size = 80, font = fonts[7], color = "white")  
plot(annotated_25)
```



図 11.5: 文字を追加した画像

《Tips》 オープンソースのパッケージやアプリの開発者に要望などを連絡すると快く対応してくれることが多いです。 `magick_fonts()` は著者が GitHub を通して連絡したところ、`magick` パッケージの開発者に迅速に対応して作成してくださったものです。`magick` 以外にも、秀丸エディタと学会タイマーの開発者に要望を送ったところ、どちらの開発者も親切に対応してくださいました。また、著者が開発した `screenshot` パッケージにも質問が来たことがあります、質問や要望を頂くのは嬉しいです。オープンソースのプログラム開発者は、自分が開発したものが役に立つてわかるのが何よりも嬉しいものです。連絡する手段には、電子メールや GitHub の Issue などがあります。

11.9 画像の結合

画像を縦あるいは横に並べて結合するには、`image_append()` を使います。引数には画像オブジェクトを指定します。既定値では横向きに結合します。

コード 11.19 (`image-append-horizontal.R`) : 横方向への結合

```
image_append(imgs[1:24]) |>  
  plot() # 横方向
```

図 11.6: 横方向に結合した画像

`stack = TRUE` と指定すると、縦向きに結合します。

コード 11.20 (image-append-vertical.R) : 縦方向への結合

```
image_append(imgs[25:27], stack = TRUE) |> # 縱方向  
  plot()
```



図 11.7: 縦方向に結合した画像

結合する画像に間隔を開けたいときは、`image_border()`と組み合わせます。次のコードでは余白の部分を分かりやすくするために、灰色(`gray()`)を使っています。

コード 11.21 (image-append-border.R) : 間隔を開けた結合

```
 imgs[1:3] |>
  image_border(color = gray(0.8), geometry = "30") |>
  image_append() |>
  plot()
```



図 11.8: 間隔を開けて結合した画像

`imgs` の画像を全て結合します。 もともとの幅、高さ、余白の大きさが異なっていますので、同じ大きさでは表示されません。

コード 11.22 (`image-append-all.R`) : 画像を全て結合

```
appended <-  
  image_append(  
    c(image_append(imgs[1:8]),  
     image_append(imgs[9:16]),  
     image_append(imgs[17:24]),  
     image_append(imgs[25:27])),  
    stack = TRUE)  
plot(appended)
```



図 11.9: 全てを結合した画像

II.10 背景の塗りつぶし

背景を塗りつぶすには、`image_fill()`を使います。`color`に塗りつぶす色を指定します。基準点である`point`の既定値は"+1+1"で画像の左上です。

コード II.23 (`image-ffill.R`) : 背景の塗りつぶし

```
filled_06 <- image_fill(imgs[6], "gold")
plot(filled_06)
```



図 II.10: 背景を塗りつぶした画像

`image_fill()`では、基準点と同じあるいは似た色が連続している部分を塗りつぶしています。どの程度の色が似ているかは`fuzz`で指定します。`fuzz`の既定値は0で基準点と同じ色のみです。100のときは全ての色ですので画像全体を1色に塗りつぶします。

コード II.24 (`image-ffill-fuzz.R`) : `fuzz` を指定した背景の塗りつぶし

```
image_fill(imgs[27], "white", fuzz = 20) |>
  plot()
```



図 II.11: `fuzz` を指定して背景を塗りつぶした画像

II.11 余白の除去

画像の余白を除去するには、`image_trim()`を使います。余白の色は白と限らず、上下と左右の端から同じあるいは似た色の部分が除去されます。どの程度の色が似ているかは `fuzz` で指定します (II.10 節を参照)。次のコードでは、余白部分をわかりやすくするために、余白を除去したあとに灰色の枠をつけます。

コード II.25 (`image-trim.R`) : 余白の除去

```
img_06_07 <- c(filled_06, imgs[7])
trimed <- image_trim(img_06_07)
c(img_06_07, trimed) |>
  image_border(gray(0.7), "10x10") |>
  image_append() |>
  plot()
```



図 II.12: 余白を除去した画像(左 2 つ: 前, 右 2 つ: 後)

II.12 サイズの変更

画像サイズを変更するには、`image_scale()`を使います。`geometry = "100"`で幅 100 ピクセルに、`geometry = "x100"`で高さ 100 ピクセルに変更できます。なお、縦横比は維持されます。`geometry = "100x200"`のように幅と高さの両方を指定すると、幅 100・高さ 200 の長方形に収まるように縮小されます。

コード II.26 (`image-scale.R`) : サイズの変更

```
r_01_w100 <- image_scale(imgs[1], geometry = "200")
r_01_h250 <- image_scale(imgs[1], geometry = "x50")
```

```
scaled <- image_append(c(imgs[1], r_01_w100, r_01_h250))
plot(scaled)
```



図 11.13: サイズを変更した画像

最終的な幅や高さではなく、0.5倍のように比率を指定したいこともあるでしょう。比率を下に幅を求めて、サイズを変更する関数を定義します。

コード 11.27 (image-scale-ratio-fun.R) : 比率を指定して画像サイズを変更する関数

```
image_scale_ratio <- function(image, ratio){
  round(magick::image_info(image)$width * ratio) |>
    purrr::map(magick::image_scale, image = image) |>
    magick::image_join()
}
```

0.5倍と0.25倍で実行してみます。

コード 11.28 (image-scale-ratio.R) : 比率指定でのサイズの変更

```
r_21_050 <- image_scale_ratio(imgs[21], 0.5)
r_21_025 <- image_scale_ratio(imgs[21], 0.25)
scaled_21 <- image_append(c(imgs[21], r_21_050, r_21_025))
plot(scaled_21)
```



図 11.14: 比率指定でサイズを変更した画像

画像の幅や高さあるいは比率ではなく、画像のファイルサイズを指定したいこともあるでしょう。画像のファイルサイズは画像の面積とほぼ比例すると考えられます。また、面積は幅や高さの2乗です。これらから、元の画像のファイルサイズ、画像の幅や高さ、求めるファイルサイズをもとにすれば画像の面積を推測可能です。実際の画像ファイルでは、ヘッダー部分や圧縮のされ方も関係するため、面積とファイルサイズとは厳密には対応していません。そこで、上記の方法で求めた比率の0.1倍から3倍までのファイルを作ります。その中から求めるファイルサイズ以下の最も高画質な画像を選びます。

これらをもとに関数を定義します。画像をファイルから読み込んだ後に画像変換した場合、画像のファイルサイズはわかりませんので、その場合は一旦保存してから読み込み直します。

コード 11.29 (image-scale-filename.R) : ファイルサイズを指定してサイズを変更する関数

```
image_scale_filesize <- function(image, filesize){  
  info <- magick::image_info(image)  
  fm <- info$format  
  fs <- info$filesize  
  wd <- info$width  
  tmp_path <- fs::file_temp(ext = fm)  
  if(fs == 0){  
    # ファイルサイズが0  
    magick::image_write(image, tmp_path)          # 一旦保存  
    image <- magick::image_read(tmp_path)         # 再読み込み  
    fs <- magick::image_info(image)$filesize  
  }  
  ratio <-  
    (filesize / fs) ^ 0.5 |>                      # 比率の平方根  
    `*` (e1 = _, e2 = seq(from = 0.1, to = 3, by = 0.1)) # 0.1-3.0で、0.1刻み  
  ratio <- ratio[ratio < 1]                         # 比率が1未満に限定  
  tmp_path <-  
    paste0(fs::path_ext_remove(tmp_path), "_",  
           file_ext(image))
```

```

    ratio, "_.", fs::path_ext(tmp_path))
image_scale_ratio(image, ratio) |>                                # 複数の比率倍の画像
  as.list() |>
  purrr::map2(tmp_path, magick::image_write)                         # 一旦保存
  image <- magick::image_read(tmp_path)                                 # 再読み込み
  fs <- magick::image_info(image)$filesize                            # ファイルサイズ
  index <-
    which(fs < filesize) |>
    max()                                                               # 指定のサイズ未満
  return(image[index])                                              # 最大値
}

```

ファイルサイズを指定してサイズを変更してみます。

コード II.30 (image-scale-filename.R) : ファイルサイズを指定してたサイズの変更

```

img_25_fs <- image_scale_filesize(imgs[25], 100000)
image_info(img_25_fs)$filesize # ファイルサイズ
## [1] 84527
scaled_25 <- image_append(c(imgs[25], img_25_fs))
plot(scaled_25)

```



図 II.15: ファイルサイズを指定してサイズを変更した画像

指定したよりも若干小さめのファイルサイズになっていることがわかります。

II.13 画像内の文字認識

`image_ocr()`を使うと、画像内の文字認識が可能です。ただし、画像認識 자체は `tesseract` パッケージがしますので、事前のインストールが必要です。文字認識に使用する言語のモデルのダウンロードも必要で、日本語モデルは `tesseract_download(lang = "jpn")` でダウンロードします。なお、言語モデルのダウンロードは初回だけです。

コード 7.18 (pdf-tesseract-install.R) : tesseract のインストール

```
install.packages("tesseract")
tesseract::tesseract_download(lang = "jpn")
```

ここでは、読み取り用のサンプルとして、祝日と都道府県の一覧が書かれた画像を使います(図 11.16)。なお、この画像は実際のスキャン画像を想定しているため、わざとノイズを加えています。

元日	北海道	石川県	岡山県
成人の日	青森県	福井県	広島県
建国記念の日	岩手県	山梨県	山口県
天皇誕生日	宮城県	長野県	徳島県
春分の日	秋田県	岐阜県	香川県
昭和の日	山形県	静岡県	愛媛県
憲法記念日	福島県	愛知県	高知県
みどりの日	茨城県	三重県	福岡県
こどもの日	栃木県	滋賀県	佐賀県
海の日	群馬県	京都府	長崎県
山の日	埼玉県	大阪府	熊本県
敬老の日	千葉県	兵庫県	大分県
秋分の日	東京都	奈良県	宮崎県
スポーツの日	神奈川県	和歌山県	鹿児島県
文化の日	新潟県	鳥取県	沖縄県
勤労感謝の日	富山県	島根県	

図 11.16: 文字認識用のサンプル画像

`image_read()`で読み込んでから、`image_ocr()`を使います。引数 `lang` には言語を指定します。

コード 11.31 (image-ocr.R) : 画像内の文字認識

```
jps <- "jps.png"
url <- paste0("https://matutosi.github.io/r-auto/data/", jps)
path <- fs::path_temp(jps)
curl::curl_download(url, path)
path |>
  image_read() |>
  image_ocr(lang = "jpn") |>
  cat()
## 元日 北海道 石川県 岡山県
## 成人の日 青森県 福井県 広島県
## 建国記念の日 岩手県 山梨県 山口県
## 天皇誕生日 宮城県 長野県 徳島県
## 春分の日 秋田県 岐阜県 香川県
## 昭和の日 山形県 静岡県 愛媛県
## 喜法記念日。 福島県 愛知県 高知県
## みどりの日 茨城県 三重県 福岡県
## こどやの日 栃木県 滋賀県 佐賀県
## 海の日 群馬県 京都府 長崎県
## 山の日 埼玉県 大阪府 熊本県
## 散老の日 千葉県 兵庫県 大分県
## 秋分の日 東京者 奈良県 宮崎県
## スポーツの日 神奈川県。 和歌山県 鹿児島県
## 文化の日 新潟県 鳥取県 沖縄県
## 勤労感謝の日。 富山県 島根県
```

認識精度はあまり高くありません。確認作業は必要ですが、大量の資料の場合は手入力するよりは手間が省けるはずです。

上のコードでは傾きのない画像を使いましたが、スキャンしたものやカメラで撮影した画像を使うときは、読み込んだ文字が傾いていることがあります。そのときは、`image_rotate()`や`image_deskew()`で傾きを補正すると認識精度が高くなります。

画像を白と黒の2色に変換、つまり2値化することで認識精度が高くなることがあります。画像の2値化は、`image.binarization`パッケージで可能です。

コード 11.32 (image-binarization-package.R) : `image.binarization` のインストール

```
install.packages("image.binarization")
library(image.binarization)
```

2値化の前には、`image_convert()`でグレースケールに変換してから、`image_binarization()`を使います。引数 `type` には2値化に使うアルゴリズムを指定します。多くのアルゴリズムがありますので、いくつか試して画像に適したものを使ってください。

実行すると、グレースケールと2値化によって文字が鮮明になります(図 II.17)。

コード II.33 (image-binarization.R) : 画像の2値化

```
img_bin <-
  path |>
  image_read() |>
  image_convert(colorspace = "Gray") |>
  image.binarization::image_binarization(type = "wolf")
plot(img_bin)
```

元日	北海道	石川県	岡山県
成人の日	青森県	福井県	広島県
建国記念の日	岩手県	山梨県	山口県
天皇誕生日	宮城県	長野県	徳島県
春分の日	秋田県	岐阜県	香川県
昭和の日	山形県	静岡県	愛媛県
憲法記念日	福島県	愛知県	高知県
みどりの日	茨城県	三重県	福岡県
こどもの日	栃木県	滋賀県	佐賀県
海の日	群馬県	京都府	長崎県
山の日	埼玉県	大阪府	熊本県
敬老の日	千葉県	兵庫県	大分県
秋分の日	東京都	奈良県	宮崎県
スポーツの日	神奈川県	和歌山県	鹿児島県
文化の日	新潟県	鳥取県	沖縄県
勤労感謝の日	富山県	島根県	

図 II.17: 2値化した画像

2値化したものを画像認識させると、誤認識するところもありますが、若干は認識精度が高くなる傾向にあります。

コード II.34 (image-binarization-ocr.R) : 2値化した画像の文字認識

```
img_bin |>
  image_ocr(lang = "jpn") |>
  cat()
## Image too small to scale!! (1x36 vs min width of 3)
## Line cannot be recognized!!
## (中略)
## 元日 北海道 。 石川県 岡山県
## 成人の日 青森県 福井県 広島県
## 建国記念の日 岩手県 山梨県 山口県
## 天皇誕生日 。 宮城県 長野県 ” 徳島上
## 春分の日 秋田県 岐阜県 。 香川県
## 昭和の日 山形県 静岡県 愛媛県
## 法記今日 福島県 。 胡知県 高知上
## みどりの日 茨城県 三重県 福岡上
## こどもやの日 栃木県 滋賀県 佐賀県
## 海の日 格馬県 京都府 長崎県
## 山の日 埼玉県 _ 大阪府 熊本県
## 敬老の日 千葉県 兵庫県 大分県
## 秋分の日 導 奈良県 。 宮崎県
## スポーツの日 神奈川県 和歌山県 。 鹿児島県
## 文化の日 新潟県 鳥取県 沖縄上
## 勤労感謝の日 富山県 。 島根県
```

II.14 傾き追加・補正

画像を傾けるには `image_rotate()` を使います。引数 `degree` に傾ける角度を時計回りで指定します。次のコードでは傾きがわかりやすいように、枠を付けた画像を使います。

コード II.35 (image-rotate.R) : 傾きの追加

```
rotated <-
  image_border(imgs[10], color = gray(0.1), geometry = "10x10") |>
  image_rotate(5)
plot(rotated)
```



図 11.18: 傾きを追加した画像

画像の傾きを補正するには `image_deskew()` を使います。次の画像では、元の傾きとは逆向きに少し傾いてしまっているように、全てが正しく補正できるわけではありません。それでも、画像の文字認識をする際には、試してみる価値はあります。

コード 11.36 (`image-deskew.R`) : 傾きの補正

```
deskewed <- image_deskew(rotated)
plot(deskewed)
```



図 11.19: 傾きを補正した画像

11.15部分の切り取り

画像から特定の部分を切り取るには、`image_crop()` を使います。引数 `geometry` には切り取りサイズと位置を文字列で指定します。`geometry` は、"幅 × 高さ + 横位置 + 縦位置" で指定します。ただし、幅と高さは切り取り後のサイズ、縦・横位置は起点です。つまり、幅 600 高さ 500 の画像を、横 400 縦 450 を起点として切り取るには "600x500+400+450" とします。

コード II.37 (image-crop.R) : 部分の切り取り

```
cropped <- image_crop(imgs[25], "600x500+400+450")
plot(cropped)
```



図 II.20: 部分を切り取った画像

好みの問題はありますか、geometry の指定方法が独特です。 画像の左上と右下の xy 座標の位置で取りサイズ・位置を指定する関数を作成します。

コード II.38 (image-ltrb2geo-fun.R) : 左上・右下の位置を geometry に変換する関数

```
ltrb2geo <- function(left_top, right_bottom){
  left <- left_top[1]
  top <- left_top[2]
  right <- right_bottom[1]
  bottom <- right_bottom[2]
  geometry <-
    paste0(right - left, "x", bottom - top,
    "+", left, "+", top)
  return(geometry)
}
```

実際に左上と右下で切り取ると、当然ながら同じ画像を得ることができます。

コード II.39 (image-ltrb2geo.R) : 左上・右下の位置での切り取り

```
geometry <- ltrb2geo(c(400, 450), c(1000, 950))
geometry
## [1] "600x500+400+450"
cropped_ltrb <- image_crop(imgs[25], geometry = geometry)
plot(cropped_ltrb)
```

画像の一部を切り取る前ときに、切り取りたい部分の正確な座標は分かっていないことが普通です。そこで、画像をクリックして切り取る位置を取得する関数を作成します。

複数の画像に対して同じ範囲で切り取るときに便利です。手作業は1回だけ必要ですが、切り取り範囲を1つの画像で決めるだけで済みます。また、正確に同じ位置になるのが、自動化の利点です。

コード 11.40 (image-click-locate-image-fun.R) : 画像をクリックして位置を取得する関数

```
click_locate_image <- function(img, n = 2){  
  par(mar = rep(0.1, 4))          # 余白を狭く  
  par(oma = rep(0.1, 4))  
  plot(img)                      # 描画  
  pos <- locator(n = n)          # 位置取得の回数  
  pos <- purrr::map(pos, round) # 丸め  
  w <- magick::image_info(img)$width  
  h <- magick::image_info(img)$height  
  pos$y <- h - pos$y            # 上下位置の反転  
  pos <-  
    seq(n) |>  
    purrr::map(\(i){ c(pos$x[i], pos$y[i]) } )  
  return(pos)  
}
```

《Tips》 `map()` の中で `()` としているのは、使い捨ての関数を作るためのもので、`function()` と同じ役割をします。

画像の位置をマウスでクリックして位置を得るには、`locator()` を使います。引数 `n` には箇所数を指定します。ここでは左上と右下ですので、`n = 2` です。その後は、`locator()` で取得した位置の x 座標と y 座標を左上および右下の位置に変換して返します。

`click_locate_image()` を実行すると画像が表示されますので、位置情報を取得したい部分をクリックします。切り取りたい部分の左上と右下の位置をクリックすると、位置を取得できます。

コード 11.41 (image-click-locate-image.R) : 画像をクリックした位置の取得

```
pos <-  
  imgs[25] |>  
  image_scale("1200") |>  
  click_locate_image(n = 2)  
pos  
[[1]]  
[1] 395 450
```

```
[[2]]
```

```
[1] 736 724
```

Rではクリックした位置は表示されませんが、RStudioではクリックした位置が小さな吹き出しのようなアイコンで表示されます(図 11.21)。

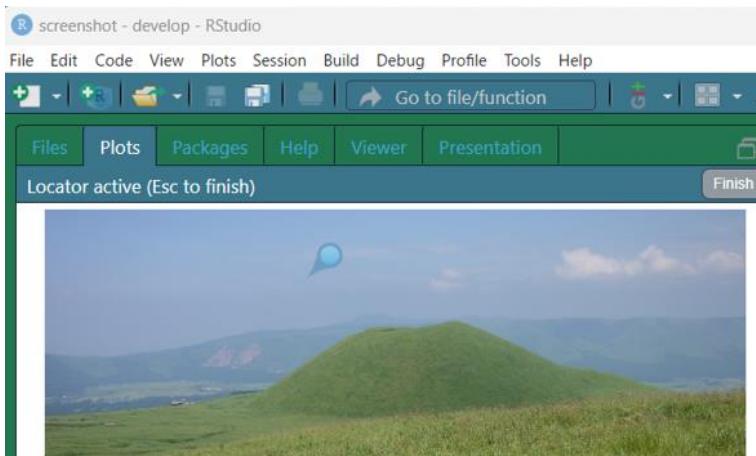


図 11.21: クリックした位置の取得(RStudioでの表示)

複数の画像を同じ位置で切り取るには、以下のように `as.list()` でリストに変換してから `map` を使います。

コード 11.42 (`image-click-locate-image-multi.R`) : 複数画像の同一位置での切り取り

```
geometry <- ltrb2geo(pos[[1]], pos[[2]])
cropped_multi <-
  imgs[25:27] |>
  image_scale("1200") |>
  as.list() |>
  purrr::map(image_crop, geometry) |>
  image_join()
cropped_multi
## # A tibble: 3 × 7
##   format width height colorspace matte filesize density
##   <chr>   <int>  <int> <chr>     <lgl>    <int> <chr>
## 1 JPEG     341    274 sRGB      FALSE      0 72x72
## 2 JPEG     341    274 sRGB      FALSE      0 180x180
## 3 JPEG     341    274 sRGB      FALSE      0 600x600
cropped_multi |>
  image_border("white", "10") |>
  image_append() |>
  plot()
```

`width` と `height` が 3つとも同一であることがわかります。



図 11.22: 同一位置で切り取った複数画像

11.16 連続した指定範囲の切り取りと保存

画像操作の自動化手法を紹介します。指定した画像を読み取り、マウスで指定した範囲を切り取って、ファイルとして書き込む方法です。操作の特性上、マウスでの範囲指定だけは手作業が入りますが、かなりの手間を削減できます。

切り取りたい位置の左上と右下をマウスで指定するために、コード 11.40 で定義した `click_locate_image()` を使います。

コード 11.43 (`click-crop-image-fun.R`) : 指定範囲を切り取る関数

```
click_crop_image <- function(path){
  img <- magick::image_read(path)                      # 読み取り
  pos <- click_locate_image(img)                        # 切り取り位置
  geometry <- ltrb2geo(pos[[1]], pos[[2]])            # 位置の変換
  img_cropped <- magick::image_crop(img, geometry)    # 切り取り
  path_cropped <- fs::path(fs::path_dir(path), paste0("cropped_", fs::path_file(path = path))) # 保存ファイル
  magick::image_write(img_cropped, path_cropped)        # 保存
  return(list(path_cropped, geometry))
}
```

作業したい画像ファイルの一覧を `fs` パッケージの `dir_ls()` で取得するとともに、`purrr` パッケージの `map()` や `for` ループを使えば、連続的にファイルが表示されますので、切り取りたい位置をクリックしていくだけで次々と画像の切り取りができます。画像ごとにファイルを開いて、切り取った画像を複写・貼り付けして、ファイル名を指定して保存するという手間が省けます。

コード 11.44 (`click-crop-image.R`) : 指定範囲の連続切り取り

```
path_imgs <- fs::path_temp(files)
  # map
path_imgs[1:3] |>
  purrr::map(click_crop_image)
  # for
for(path in path_imgs[1:3]){

```

```
    click_crop_image(path)
}
```

11.17 アニメーション化

他の画像形式と異なり, gif ではアニメーション表示が可能です。いわゆるアニメーション gif です。

`image_animate()`を使うと、複数の画像を結合してアニメーション GIF が作れます。引数 `fps` は 1 秒あたりのフレーム数(frames per second)で、`fps = 10` とすると 1 秒間に 10 の画像が表示されます。つまり画像 1 つの表示時間は $1/10$ 秒です。

画像を徐々に変化させるには、`image_morph()`を使って複数の画像を混ぜ合わせて、中間的な画像を生成します。引数 `frames` には中間画像の数を指定します。2 つの画像を入力し、`frames = 10` とすると、元の画像に加えて 10 の中間画像が作られますので、入力画像とあわせて 12 の画像を含むオブジェクトを出力します。

コード 11.45 (`image-morph-animate.R`) : アニメーション化

```
animated <- fs::path_temp("animated.gif")
mor <-
  c(imgs[22], imgs[9], imgs[12], imgs[22]) |>
  image_trim() |>          # 余白の除去
  image_resize("x200") |>   # サイズ合わせ
  image_morph(10) |>        # 中間画像の生成
  image_animate(fps = 5) |> # アニメーション化
  image_write(animated)
# shell.exec(animated)
```

`shell.exec()` で画像を表示すると徐々に画像が変化していくのがわかります(図 11.23)。



図 11.23: アニメーション画像の変化の様子

11.18 その他の画像の加工

magick では、ぼかし、ノイズの追加、木炭画、ネガなど多くの画像加工も可能です(表 11.1)。第 1 引数 `image` には画像オブジェクトをとりますが、それ以外の引数は関数によって異なります。

直感的に分かりにくい引数として、`radius`, `sigma`, `fuzz` があります。`radius` は範囲の大きさで、数値を大きくすると加工が強くなります。`sigma` は強さで、数値を大きくと加工が強くなります。ちらも加工の強さに影響を与えますので、実際に使うときにはいくつかの数値を試してください。`fuzz` は指定色との非類似度で、0 の場合は指定色のみ、数値が大きいと類似度の低い色までが指定されますになります。たとえば、透明化加工の `image_transparent()` で、`fuzz = 0` のときは指定した色のみを透明化しますが、`fuzz = 10` のようにすると指定した色と少し異なる色も透明化します。

表 11.1: 画像加工の主な関数

加工	関数	<code>image</code> 以外の引数
ぼかし	<code>image_blur()</code>	<code>radius</code> (範囲), <code>sigma</code> (強さ)
ノイズ追加	<code>image_noise()</code>	<code>noisetype</code> (ノイズの種類)
木炭画	<code>image_charcoal()</code>	<code>radius</code> , <code>sigma</code>
エッジ	<code>image_edge()</code>	<code>radius</code>
油絵風	<code>image_oilpaint()</code>	<code>radius</code>
ネガ	<code>image_negate()</code>	
透明化	<code>image_transparent()</code>	<code>color</code> , <code>fuzz</code> (非類似度)
上下反転	<code>image_flip()</code>	
左右反転	<code>image_flop()</code>	

【相談】 PDF を画像に変換して透明化する部分は、画像のところに入れましたが、PDF のところの方が良いでしょうか。

PDF を画像として読み取り、その画像の背景を `image_transparent()` で透明化してから保存する関数を定義します。

コード 11.46 (`image-etc-transparent-fun.R`) : PDF ファイルの背景を透明化する関数

```
gg_point <- function(path, size, color, fill){ # 散布図の描画・保存
  tibble(x = runif(1000), y = runif(1000)) |>
    ggplot(aes(x, y)) +
    geom_point(shape = 21, size = size, color = color, fill = fill) +
    theme_bw()
  ggsave(path, width = 5, height = 5)
```

```

}

pdf_transparent <- function(path){ # PDF 背景の透明化
  path |>
    image_read_pdf() |>
    image_transparent("white") |> # 白を透明化
    image_write(path, format = "pdf")
}

```

ggplot2で散布図を作成し、重ね合わせます。

コード II.47 (image-etc-transparent.R) : PDF ファイルの背景の透明化

```

path <- fs::path_temp(c("gg_1.pdf", "gg_2.pdf"))      # 個別の散布図の PDF
path_out <- fs::path_temp(c("fill.pdf", "trans.pdf")) # 重ね合わせした PDF
tibble(path = path,
       color = c("black", "red"),
       size = c(1, 5),
       fill = c("black", "white")) |>
  purrr::pwalk(gg_point)
  # 透明化・重ね合わせ
qpdf::pdf_overlay_stamp(path[1], path[2], out = path_out[1]) # 透明化前
purrr::walk(path, pdf_transparent)                                # 透明化
qpdf::pdf_overlay_stamp(path[1], path[2], out = path_out[2]) # 透明化後
out <- pdftools::pdf_combine(c(path, path_out))           # 全 PDF を結合
# shell.exec(out)

```

図 II.24 は、左から黒色の散布図、赤色の散布図、透明化前に重ね合わせたもの、透明化後に重ね合わせたものです。透明化前は上書きされたため赤色の点しか表示されていませんが、透明化後は赤色と黒色の両方の点が表示されています。

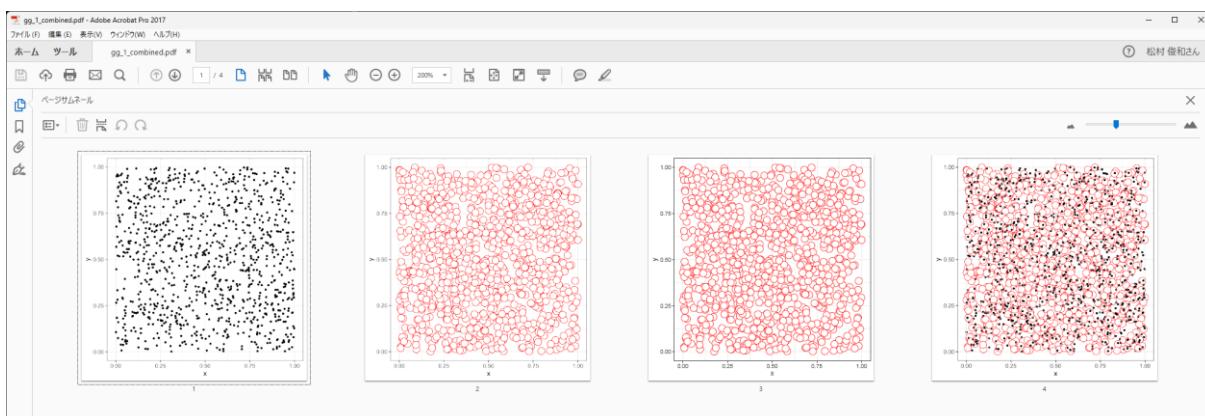


図 II.24: PDF 背景の透明化(左から、黒色のみ、赤色のみ、透明化前、透明化後)

11.19 画像の一覧整理

特定のディレクトリにある画像のうち、特定の条件に該当するファイルだけの一覧が欲しいことがあります。枠と文字列を追加する関数を組み合わせて使うと、大量の画像の中から条件に該当する画像の一覧整理が自動化できます。ここでは、これまでの内容をもとにしてそのための関数をつくります。

- `trim()`で余白の削除
- `image_scale()`で横幅の統一
- `image_border()`と `image_annotate()`で余白にファイル名を書き込み
- `image_border()`と `image_crop()`で高さの調整の
- `image_append()`で画像の結合

コード 11.48 (`image-annotate-fnames-fun.R`) : ディレクトリ内の画像にファイル名を書き込んで結合する関数

```
image_annotate_fnames <- function(dir,
  regexp = "\\.(png|jpg)$", ncol = NULL,
  scale = "200", border = "x30", size = 25, color = "white"){
  fnames <- fs::dir_ls(dir, regexp = regexp)
  annotated <-
    magick::image_read(fnames) |> # 読み込み
    magick::image_trim() |> # 余白の削除
    magick::image_scale(geometry = scale) |> # サイズ変更
    magick::image_border(color = color, geometry = border) |> # 余白の追加
    as.list() |>
    purrr::map2(fs::path_file(fnames),
      magick::image_annotate, size = size) |> # 註釈の追加
      magick::image_join() # 結合
    same_sized <- same_height(annotated) # サイズの統一
  n <- length(fnames)
  if(is.null(ncol)){
    ncol <- ceiling(n^0.5)
  }
  rows <- row_index(n, ncol) #
  appended <-
    rows |>
    purrr::map(\(x){ image_append(same_sized[x]) }) |>
    magick::image_join() |>
    magick::image_append(stack = TRUE)
  return(appended)
}
# 画像数と列数をもとに、行の画像の配列を設定
row_index <- function(n, ncol){
```

```

nrow <- ceiling(n %/% ncol)                                # 行数
n_ends <- seq(nrow) * ncol                               # 各行の終了 index
n_sarts <- dplyr::lag(default = 1, n_ends + 1) # 各行の開始 index
n_ends[nrow] <- n                                       # 最終行の最後を n に修正
rows <- purrr::map2(n_sarts, n_ends, \{x, y\}{ seq(from = x, to = y) })
return(rows)
}

# 画像の高さを揃える
same_height <- function(imgs){
  height <- magick::image_info(imgs)$height # 高さ
  width <- magick::image_info(imgs)$width    # 幅
  max_h <- ceiling(max(height) * 0.1) * 10 # 高さの最大値
  border <- max(max_h - height)           # 追加分
  bordered <- magick::image_border(          # 上下に余白を追加
    imgs, color = "white", geometry = paste0("x", border))
  geometry <- paste0(width, "x", max_h, "+0+", border)
  same_sized <-                                         # 上の余白を削除
    purrr::map2(as.list(bordered), geometry, magick::image_crop) |>
    magick::image_join()
  return(same_sized)
}

```

`regexp` には正規表現(3.4 節を参照)が使えます。そのためやや複雑な条件でもファイル名を指定可能です。

コード 11.49 (`image-annotate-fnames.R`) : ファイル名を書き込んで結合する

```

dir <- fs::path_temp()
regexp <- "r_\\d+\\.\\.(png|jpg)$"
img_all <- image_annotate_fnames(dir = dir, regexp = regexp, ncol = 8)
plot(img_all)

```



図 11.25: ファイル名を書き込んで結合した画像の一覧

11.20スクリーンショットの保存

スクリーンショットを保存するには、`screenshot`パッケージを使います。

コード 11.50 (`image-screenshot-install.R`) : `screenshot` のインストールと呼び出し

```
install.packages("screenshot")
library(screenshot)
```

Windows では、`install_screenshot()`でスクリーンショットのためのアプリをインストールします。引数 `bin_dir` を省略したときは `screenshot` のディレクトリにインストールされます。このときは、`screenshot()`で引数 `bin_dir` を省略できるので、おすすめです。もし、他の場所にインストールしたければ、任意のディレクトリを指定してください。

コード 11.51 (`image-screenshot-install-screenshot.R`) : スクリーンショットのためのアプリのインストール

```
screenshot::install_screenshot()
## screenshot.exe is installed in
## C:/Users/USERNAME/AppData/Local/R/win-library/4.3/screenshot
```

スクリーンショットを保存するには、`screenshot()`を使います。`install_screenshot()` 引数 `file` には保存するパスを指定します。省略すると一時ファイル(`fs::file_temp()`)に `sc_` で始まるファイル名で保存されます。スクリーンショットは PNG 形式です。引数 `bin_dir` には、スクリーンショットのアプリをインストールしたディレクトリを指定します。ただし、引数を省略して `install_screenshot()` を実行したときには、省略可能です。

コード 11.52 (`image-screenshot-screenshot.R`) : スクリーンショットの保存

```
ss <- screenshot::screenshot()
fs::path_file(ss) # ファイル名のみ
## [1] "sc_158839211323.png"
magick::image_read(ss)
## # A tibble: 1 × 7
##   format width height colorspace matte filesize density
##   <chr>    <int>   <int> <chr>     <lgl>    <int> <chr>
## 1 PNG      1920    1080 sRGB      TRUE     185806 47x47
# shell.exec(ss) # 関連付けアプリで起動
```

返り値はスクリーンショット画像のパスですので、`image_read()`で読み込んだり、`shell.exec()`で関連付けのアプリで起動したりできます。

11.2 | クリップボード画像の自動保存

クリップボードに保存された画像をファイルとして保存するには、ちょっと手間がかかります。たとえば、スクリーンショットで四角形等の範囲で切り抜いたときは、画像がクリップボードに保存されます。その後、画像編集ソフトを開き、クリップボードから画像を貼り付けて、ファイル名を指定して保存します。個別の作業は簡単ですが、作業マニュアルの作成のように枚数が多いときには結構手間がかかります。このようなときには、`screenshot` の `save_clipboard_image()` を使うとファイルの保存を自動化できるので、楽に作業が完了します。



図 11.26: 四角形の範囲でのスクリーンショット

`save_clipboard_image()` は、引数 `path` で指定した場所にクリップボード画像が `png` 形式で保存されます。`path` を省略すると、一時ディレクトリに画像が保存されます。

コード 11.53 (`image-screenshot.R`) : スクリーンショットの保存

```
clipboard_img <- save_clipboard_image()  
# shell.exec(clipboard_img)
```

【相談】 バッチファイルでの実行・瞬間起動ですが、少し詳しすぎるでしょうか。「x.x 節を参照」でよければ、多少省略します。

ただし、範囲指定でスクリーンショットをとったあと、毎回 `save_clipboard_image()` を実行するのは面倒です。また、保存するファイルは連番であれば、整理が楽です。ここでは、クリップボード画像をデスクトップに連番のファイル名で保存するプログラムを作成します。このプログラムをバッチファイル(R CMD BATCH)から実行するように設定しておき、`[Win] + [R]` で起動します(5.1 節を参照)。そうすると、次の作業を繰り返すだけで部分的なスクリーンショットを連続的に保存できます。

- ・ `[PrtSc]` でスクリーンショットを起動(図 11.26)
- ・ マウスで保存範囲を選択(図 11.27)
- ・ `[Win] + [R]` で `sci` を実行して、クリップボードの画像を保存(図 11.28)

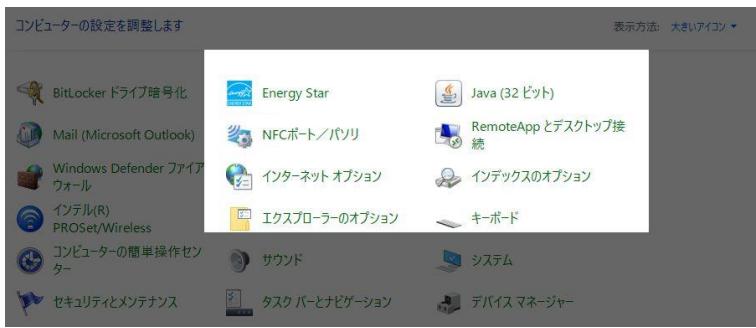


図 II.27: マウスでの保存範囲の選択

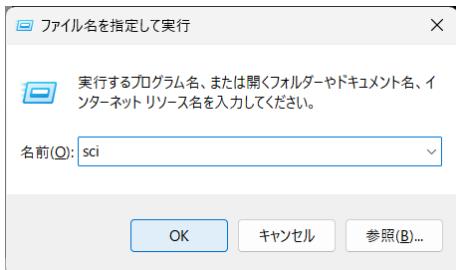


図 II.28: Win + R でのプログラムの実行

次のプログラムを `c:/Users/USERNAME/RR/` に `save_clipboard_image.R` で保存します。また、`c:/Users/USERNAME/RR/` と R の実行ファイルがあるディレクトリにパスを通しておきます。なお、ディレクトリは読者の環境に合わせて設定してください。

コード II.54 (img-save-screenshot-code.R) : クリップボード画像の自動保存

```
wd <- fs::path(fs::path_home(), "desktop") # 保存先ディレクトリ
setwd(wd) # 保存ファイルの指定
no <- stringr::str_pad(1:99, width = 2, "left", "0") # 2桁の連番
path <- paste0("clip_image_", no, ".png")
pngs <- fs::dir_ls(regexp = "\\.png") # 既存ファイル
path <- setdiff(path, pngs)[1] # 既存ファイル以外での1つ目

screenshot::save_clipboard_image(path) # 画像の保存
write.table(path, "clipboard", # ファイル名をクリップボードに保存
            quote = FALSE, row.names = FALSE, col.names = FALSE)
```

`save_clipboard_image.R` を実行するバッチファイルを、`save_clipboard_image.R` と同じディレクトリに `save_clipboard_image.bat` として保存します。

```
R CMD BATCH --silent --vanilla ^
save_clipboard_image.R save_clipboard_image.log
```

`save_clipboard_image.bat` へのショートカットをパスの通ったディレクトリに `sci(Save Clipboard Image)` として保存します。

ここまで の作業で準備は完了です。 [Win] + [R]で `sci` を実行すれば、 `sci`(ショートカット)から `save_clipboard_image.bat` が起動し、 R が `save_clipboard_image.R` を実行して、 デスクトップにクリップボードの画像が保存されます。

12 メールの操作

お礼、調査の依頼、会費の請求などでほぼ同じ電子メールを一斉に送ることがあります。全員に対して全く同じ文面であれば、BCCで送るでしょう。ただし、同時に大量の宛先に送ると、メールサーバが迷惑メールだと判断することがあるため、宛先を10名程度に分割して送るのが安全です。また、メーリングリストを使ってお知らせすることもあります。

一斉メールでの依頼には「また後でやっておこう」と後回しにしがちで、そのまま忘れてしまうこともしばしばです。一方、個人名が入ったメールには、きっちり対応する人がほとんどです。個人的な経験からいっても、一斉メールよりも個別のメールのほうが、遙かに効果が高いです。

しかし、個別のメールを大量に作成するのは時間がかかります。雛形を作成するのはすぐにできますが、そこから氏名とメールアドレスを間違えないように複写・貼り付けしないといけません。名前の間違えたメールが届くと逆効果です。数名ならともかく、何十人にも送るときには、それだけで疲弊するでしょう。

こんなときは、メール操作の自動化の出番です。氏名とメールアドレスの一覧があれば、数人に送るのも100人に送るのもほとんど同じ手間で作業できます。また、氏名だけでなく添付ファイルが異なる場合でも手間をかけずにメールを送信できます。

本章では、GmailとOutlookの操作を説明します。他の電子メールサービスでも、APIとRのパッケージがあれば、同様のことをできる可能性が高いです。

12.1 Gmailの操作の準備

Gmailの操作には、`gmailr`パッケージを使いますが、それ以外にも準備が必要です。Google Cloud Platform(GCP)への登録や認証です。準備には若干手間がかかりますが、多量のメールをRから操作できる利点は非常に大きいです。

12.1.1 `gmailr`パッケージ

Gmailを操作するには、`gmailr`パッケージを利用します。

コード12.1 (`mail-package.R`) : `gmailr`のインストール

```
install.packages("gmailr")
```

コード12.2 (`mail-library.R`) : `gmailr`の呼び出し

```
library(gmailr) # 呼び出すと説明等が表示されます
```

12.1.2 Google Cloud Platform への登録とプロジェクトの設定

Gmail を R から操作するには、Google Cloud Platform (GCP)への登録と設定など次の 4 つの作業が必要です。

- ・ GCP への登録
- ・ Gmail API の有効化
- ・ OAuth 同意画面の設定
- ・ 認証情報の設定

12.1.2.1 GCP への登録

GCP に登録するために、以下の URL にアクセスします。

<https://console.cloud.google.com/>

- ・ 利用規約に同意します。
- ・ 無料トライアルの提案があればそのまま利用しましょう。
- ・ アカウント情報として、メールアドレスを確認します。
- ・ クレジットカードや住所を登録します。無料トライアル期間であっても、支払い情報を登録する必要があります。無料トライアルが終了後も、有料サービスを利用しない限り請求はありません。

12.1.2.2 Gmail API の有効化

Gmail API を有効にするため、以下の内容を実行します。

- ・ 「API とサービス」 - 「ライブラリ」を選択します。
- ・ API ライブラリで「gmail api」で検索します(図 12.1)。
- ・ Gmail API が出てきたら、アイコンをクリックして有効にします(図 12.2)。

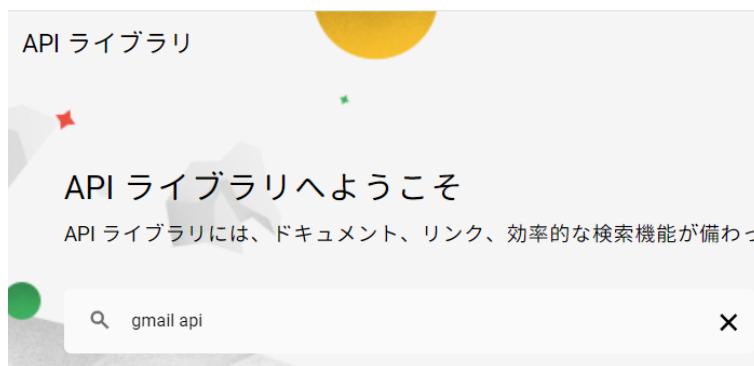


図 12.1: gmail API の検索



図 12.2: Gmail API の有効化

12.1.2.3 OAuth 同意画面の設定

次に、OAuth 同意画面を設定します。

- 「API と管理サービス」 - 「OAuth 同意画面」を選択します(図 12.3).
- User Type で「外部」を選択して、「作成」します(図 12.4).
- 「アプリ登録の編集」でアプリ名などを登録します(図 12.5). アプリ名は任意のもので構いません。ただし、「gmail」だとエラーになりますのでご注意ください。「保存して次へ」を選択します。
- その後、いくつか画面が出ますので、そのまま「保存して次へ」を選択していきます。
- OAuth 同意画面の「公開ステータス」の「アプリを公開」を選択します(図 12.6).

Cloud の概要 >

プロダクトとソリュ... >

固定されたプロダクト

API	API とサービス	有効な API とサービス
IAM	IAM と管理	ライブラリ
マーケットプレ...	マーケットプレ...	認証情報
		OAuth 同意画面

図 12.3: OAuth 同意画面を選択

API とサービス OAuth 同意画面

有効な API とサービス

ライブラリ

認証情報

OAuth 同意画面

ページの使用に関する契約

User Type

内部 [? ヘルプ](#)

組織内のユーザーのみが使用できます。確認を受けるためにアプリを送信する必要はありません。[ユーザーの種類の詳細](#)

外部 [? ヘルプ](#)

Google アカウントを持つすべてのテストユーザーが使用できます。アプリはテストモードで起動し、アプリを使用できるのは、テストユーザーのリストに追加されたユーザーに限られます。アプリを本番環境に移す準備ができたら、アプリの確認が必要となる場合があります。[ユーザーの種類の詳細](#)

作成

図 12.4: User Type の選択

アプリ情報

この情報は同意画面に表示されるため、デベロッパーのユーザー情報とデベロッパーへの問い合わせ方法をエンドユーザーが把握できます。

アプリ名*

Gmail-auto

同意を求めるアプリの名前

ユーザー サポートメール*

@gmail.com

同意に関して問い合わせる際に使用します。 [詳細](#)

アプリのロゴ

これがお客様のロゴです。このロゴは、ユーザーがアプリを認識できるよう、OAuth 同意画面に表示されます。

ロゴをアップロードした後に、アプリを送信して確認を受ける必要があります。ただし、アプリが内部でのみ使用するように構成されている場合や、アプリの公開ステータスが「テスト中」の場合は、送信は不要です。 [詳細](#)

アップロードするロゴ ファイル

[参照](#)

ユーザーがアプリを認識できるように、同意画面に 1 MB 以下の画像をアップロードします。使用できる画像形式は、JPG、PNG、BMP です。最適な結果を得るには、ロゴを 120 x 120 ピクセルの正方形にすることをおすすめします。

アプリのドメイン

デベロッパーとユーザーを保護するために、Google では、OAuth を使用するアプリのみに認可ドメインの使用を許可しています。同意画面では、次の情報がユーザーに表示されます。

アプリケーションのホームページ

ホームページへのリンクをユーザーに提供します

[アプリケーション プライバシー ポリシー] リンク

一般公開のプライバシー ポリシーへのリンクをユーザーに提供します

[アプリケーション利用規約] リンク

一般公開の利用規約へのリンクをユーザーに提供します

承認済みドメイン [?](#)

同意画面または OAuth クライアントの構成でドメインが使用されている場合は、ここで事前登録する必要があります。アプリの検証が必要な場合は、[Google Search Console](#) にアクセスして、ドメインが承認済みであるかどうかを確認してください。承認済みドメインの上限の[詳細](#)をご覧ください。

図 12.5: アプリ登録の編集

The screenshot shows the 'API & Services' section of the Google Cloud Platform console. On the left, a sidebar lists '有効な API とサービス', 'ライブラリ', '認証情報', 'OAuth 同意画面' (which is selected and highlighted in blue), and 'ページの使用に関する契約'. The main area displays the 'OAuth Consent screen' configuration for the project 'Gmail--auto'. It includes fields for '公開ステータス' (set to 'Test') and a button labeled 'アプリを公開'.

図 12.6: アプリの公開

12.1.2.4 認証情報の設定

最後に、認証情報を設定します。

- 「API とサービス」 - 「認証情報」を選択します(画面は図 12.3 と同様).
- 「+ 認証情報を作成」 - 「OAuth クライアント ID」を選択します(図 12.7).
- 「アプリケーションの種類」で「デスクトップアプリ」を選択し、名前を入力します。名前は任意のもので構いません(図 12.8).
- 「OAuth クライアントを作成しました」と表示される画面で JSON をダウンロードします(図 12.9).
- ダウンロードした JSON ファイルは、他人がアクセスできない適切な場所に保存してください。

Google のページには「5 分から数時間」必要と書かれていますが、すぐに有効になることがあります。

The screenshot shows the '認証情報' (Authentication) section of the Google Cloud Platform API & Services page. Under 'API キー', it says 'API キーを使用してプロジェクトを識別し、割り当てとアクセスを確認します' and has a link to '詳細'. Under 'OAuth クライアント ID', it says 'ユーザーのデータにアクセスできるようにユーザーの同意をリクエストします' and has a link to '詳細'. Below these, there's a 'サービス アカウント' section with a note about using a service account for inter-server communication. At the bottom, there's a 'ウィザードで選択' (Select via Wizard) section with a note about selecting authentication types. The right side of the screen shows two tables: one for '制限' (Limits) and another for 'クライアント ID' (Client ID) management.

図 12.7: OAuth クライアント ID の作成(1/2)

← OAuth クライアント ID の作成

クライアント ID は、Google の OAuth サーバーで個々のアプリを識別するために使用します。アプリが複数のプラットフォームで実行される場合、それぞれに独自のクライアント ID が必要になります。詳しくは、[OAuth 2.0 の設定](#) をご覧ください。OAuth クライアントの種類の[詳細](#)

アプリケーションの種類* –
デスクトップアプリ ▾

名前* _____

デスクトップ クライアント:1

OAuth 2.0 クライアントの名前。この名前はコンソールでクライアントを識別するためにのみ使用され、エンドユーザーには表示されません。

注: 設定が有効になるまで 5 分から数時間かかることがあります

図 12.8: OAuth クライアント ID の作成(2/2)

OAuth クライアントを作成しました

クライアント ID とシークレットには、常に API とサービスの認証情報からアクセスできます

クライアント ID	[REDACTED]	maa.apps.googleusercontent.com
クライアントシークレット	[REDACTED]	
作成日	2024年3月5日 21:32:17 GMT+9	
ステータス	 有効	
 JSON をダウンロード		

図 12.9: JSON のダウンロード

12.1.3 gmailr での認証

`gm_auth_configure()`で保存した JSON ファイルの認証情報を読み込みます。 `gm_oauth_client()` で読み込んだ情報を確認できます。

コード 12.3 (mail-qm-auth-conf.R) : 認証情報の読み込み

```
gm_auth_configure(path = "oauth-client.json")  
qm oauth client()
```

```

## <gargle_oauth_client>
## name: gmail-automater-xxxxxx_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
## id:
## xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.apps.googleusercontent.com
## secret: <REDACTED>
## type: installed
## redirect_uris: http://localhost

```

認証情報を読み込んだら、自分のメールアドレスを用いて gm_auth()で認証します。

コード 12.4 (mail-gm-auth.R) : メールアドレスでの認証

```

gm_auth("YOURNAME@gmail.com")
# Waiting for authentication in browser...
# Press Esc/Ctrl + C to abort
# Authentication complete.

```

- gm_auth()を実行するとブラウザが開きますので、アカウトを選択します。
- その後、警告画面が出ますので、「詳細」あるいは「続ける」を選択します。なお、ブラウザによって警告画面は多少異なる可能性があります。下の方にある小さめの表示の「プロジェクト名(安全ではないページ)に移動)」を選択します(図 12.10)。
- アクセスの許可の画面(図 12.11)で、「Gmail のすべての…」にチェックを入れて、「続行」を選択します。



このアプリは Google で確認されていません

アプリが、Google アカウントのプライベートな情報へのアクセスを求めていました。デベロッパー ([REDACTED]@gmail.com) と Google によって確認されるまで、このアプリを使用しないでください。

デベロッパーの場合は、この画面が表示されないようにするには審査リクエストを送信してください。 詳細



図 12.10: 警告画面



図 12.11: アクセスの許可

認証が成功すると、ブラウザで以下のように表示されますので、Rに戻ってください。

```
# 認証後にブラウザに表示されるメッセージ
Authentication complete. Please close this page and return to R.
```

gm_profile()で認証できたアカウントを確認します。

コード 12.5 (mail-gm-profile.R) : アカウントの確認

```
gm_profile()
# Logged in as:
#   * email: YOURNAME@gmail.com
#   * num_messages: 123456
#   * num_threads: 78901
```

Rを起動するたびに認証するのは面倒です。Rを再起動して作業するときのために、gm_token_write()でトークンを保存します。トークンは、他人がアクセスできない適切な場所に保存してください。

コード 12.6 (mail-gm-token-write.R) : トークンの保存

```
gm_token_write(path = "gmailr-token.rds")
fs::dir_ls(regexp = "rds") # 保存できたか確認
# gmailr-token.rds
```

R を再起動したときは、`gm_token_read()`を使います。`gm_profile()`で認証情報が表示されれば成功です。

コード 12.7 (`mail-gm-token-read.R`) : 再起動時のトークンの呼び出し

```
library(gmailr)
token <- gm_token_read("gmailr-token.rds")
gm_auth(token = token)
gm_profile()
# Logged in as:
#   * email: YOURNAME@gmail.com
#   * num_messages: 123456
#   * num_threads: 78901
```

12.2 Gmail の一覧取得

Gmail からスレッドの一覧を取得するには、`gm_threads()`を使います。引数 `search` で文字列を指定すると検索結果が表示されます。また、`num_results` で表示数を指定できます。

```
gm_threads(search = "検索文字列", num_results = 3)
## 画面によって表示され方が異なります
##           thread_id    snippet
## 1 xxxxxxxxxxxxxxxx9 本文 1 の冒頭部分。
## 2 xxxxxxxxxxxxxxxx38 本文 2 の冒頭部分。
## 3 xxxxxxxxxxxxxxxx90 本文 3 の冒頭部分。
```

個別のメールの内容を表示させるには、`gm_message()`を使います。表示するメールを特定するために、メールかスレッドの `id` を指定します。`id` は `gm_id()` で得たものを使います。

コード 12.8 (`mail-gm-messages.R`) : Gmail の内容表示

```
messages <- gm_messages(search = "検索文字列", num_results = 3)
messages
##           message_id      thread_id
## 1 xxxxxxxxxxxxxxxx9d xxxxxxxxxxxxxxxx68
## 2 xxxxxxxxxxxxxxxxc5 xxxxxxxxxxxxxxxx68
## 3 xxxxxxxxxxxxxxxx68 xxxxxxxxxxxxxxxx68
gm_id(messages)[[1]] |>
  gm_message()
# Id: xxxxxxxxxxxxxxxx9d
# To: YOURNAME@gmail.com
# From: hoge@hoge.com
# Date: Wed, 17 Jan 2024 14:12:06 +0900
# Subject: メールの件名
# メールの本文。メールの本文。メールの本文。メールの本文。
# メールの本文。メールの本文。メールの本文。メールの本文。
# Attachments: '_main.docx'
```

```
gm_id(threads)[[1]] |>
  gm_message()
# Id: xxxxxxxxxxxxxxxx9d
# To: hogehoge@gmail.com
# (以下省略)
```

`id` を特定するのが面倒なので、検索文字列と表示数を指定することでメッセージを表示する関数を定義します。

コード 12.9 (`mail-gm-messages-show-fun.R`) : メール内を検索して表示する関数

```
gm_show <- function(search = "", num_results = 3){
  msgs <-
    gmailr::gm_messages(search = search, num_results = num_results) |>
    gmailr::gm_id() |>
    purrr::map(gmailr::gm_message)
  return(msgs)
}
```

定義した関数を使ってみます。読者のメールに含まれそうな文字列で検索してください。

コード 12.10 (`mail-gm-messages-show.R`) : メールの内容を検索して表示

```
gm_show(search = "検索文字列", num_results = 3)
# [[1]]
# Id: xxxxxxxxxxxxxxxx9d
# To: YOURNAME@gmail.com
# From: hogehoge@gmail.com
# Date: Wed, 17 Jan 2024 14:12:06 +0900
# Subject: メールの件名
# メールの本文。メールの本文。メールの本文。メールの本文。
# メールの本文。メールの本文。メールの本文。メールの本文。
# (中略)
# [[2]]
# Id: xxxxxxxxxxxxxxxxc5
# (以下省略)
```

12.3 Gmail の新規メール作成とファイルの添付

メールを作成するには `gm_mime()`を使います。`gm_mime()`で作成したメールには宛名や本文などの情報がありません。`gm_to()`で宛名、`gm_cc()`でCC、`gm_bcc()`でBCC、`gm_subject()`で件名、`gm_text_body()`で本文、`gm_attach_file()`で添付ファイルを指定します。なお、`gm_attach_file()`では複数ファイルは添付できません。複数のときは、`gm_attach_file()`を複数回実行してください。

コード 12.11 (mail-gm-email.R) : メールの作成

```
gmail <-  
  gm_mime() |>  
  gm_to("hogehoge@gmail.com") |>  
  gm_subject("テストメール 1") |>  
  gm_text_body("テストメールの本文。") |>  
  gm_attach_file("photo.jpg") # 作業ディレクトリに photo.jpg がある場合
```

また、次のようにしてもメールを作成できます。どちらを使っても作られるメールの内容は同じです。

コード 12.12 (mail-gm-mine.R) : メールの作成の別 の方法

```
gmail <-  
  gm_mime(to = "hogehoge@gmail.com",  
          subject = "テストメール 1",  
          body = "テストメールの本文。",  
          attach_file = "photo.jpg")
```

12.4 Gmail でのメールの送信

コード 12.11 や 12.12 で作成したメールは、R のオブジェクトとして存在するだけで、Gmail に情報は送られていません。gm_send_message() でメールを送信できます。

コード 12.13 (mail-gm-send-mail.R) : メールの送信

```
gm_send_message(gmail)
```

12.5 Gmail の下書きへの保存と下書きメールの送信

Gmail に下書きとして保存するには、gm_create_draft() を使います。

コード 12.14 (mail-gm-messages-draft.R) : 下書きへの保存

```
draft <- gm_create_draft(gmail)
```

上のコードを実行すると、Gmail に下書きが保存されます(図 12.12.)。

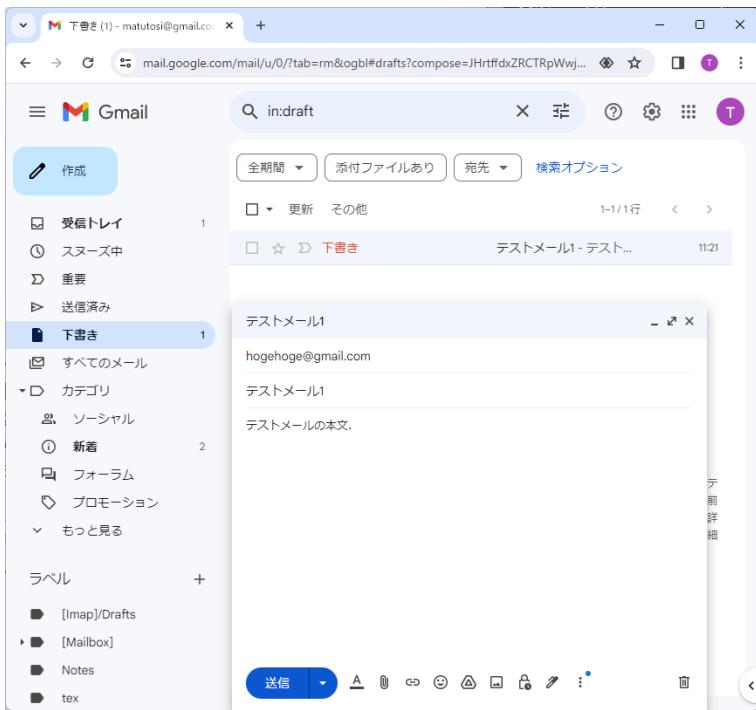


図 12.12: Gmail に保存した下書きメール

下書きフォルダ内のメールを送信するには, `gm_send_draft()`を使います.

コード 12.15 (`mail-gm-messages-draft-send.R`) : 下書きメールの送信

```
gm_send_draft(draft)
```

12.6 Gmail の一斉作成と送信

これまでの内容をもとに, 一斉メールの自動作成・送信の関数を作ります.

メールのデータはエクセルに保存しており, 送信フラグ(`send`), 宛先(`to`), 件目(`subject`), 本文(`body`), CC, BCC, 添付ファイル名(`attachment`)が入力されたエクセルのファイルがあるとします. 送信フラグは, "1"がメール作成後すぐに送信する, "0"が下書きに保存するとします. このエクセルのデータをもとにメールを作成して送信します. なお, 複数データでの繰り返しを実行するため, `purrr` パッケージ(2.8 節を参照)を多用します.

関数の全体像は次のとおりです.

- `auto_gmails()`
 - `readxl::read_xlsx()`でエクセルを読み込み
 - `gen_gmails()`で全メールの作成
 - `gen_gmail()`で個別メールの作成
 - `attach_files_gmail()`でファイルの添付
 - メールの送信 or 下書きへの保存

まずは、全体を取り仕切る `auto_gmails()` です。ここでは、メールのデータが入ったエクセルのパスを受け取り、`readxl::read_xlsx()` でメールのデータを読み込みます。読み込んだデータをもとに、`gen_gmails()` でメールを作成します。また、`send` の列の内容をもとにメールを下書きに保存するか送信します。

コード 12.16 (`mail-auto-gmails-fun.R`) : メールを自動作成・送信する関数

```
auto_gmails <- function(path){  
  df <-  
    path |>  
    readxl::read_xlsx(col_types = "text") |>  
    tidyverse::replace_na(list(cc = "", bcc = "", attachment = ""))  
  gmails <- gen_gmails(df)  
  if("send" %in% colnames(df)){  
    draft <- gmails[df$send == "0"] |> # 下書きに保存  
    purrr::map(gmailr::gm_create_draft)  
    sent <- gmails[df$send == "1"] |> # 送信  
    purrr::map(gmailr::gm_send_message)  
  }  
  return(list(sent = sent, draft = draft))  
}
```

次は、メールを作成する `gen_gmails()` です。`gen_gmails()` では、データフレームである `df` を受け取ります。`df` からメール作成に必要な列を選択して、`purrr::pmap` を使ってメールを一斉に作成します。

コード 12.17 (`mail-gen-gmails-fun.R`) : 複数メールの作成

```
gen_gmails <- function(df){  
  cols <- c("to", "cc", "bcc", "subject", "body")  
  gmails <-  
    df |>  
    dplyr::select(dplyr::any_of(cols)) |> # 必要な列を選択  
    purrr::pmap(gen_gmail)  
  if("attachment" %in% colnames(df)){  
    gmails <- purrr::map2(gmails, df$attachment, attach_files_gmail)  
  }  
  return(gmails)  
}
```

実際に個別のメールを作成するのは、`gen_gmail()` です。

コード 12.18 (`mail-gen-gmail-fun.R`) : 個別メールの作成

```

gen_gmail <- function(to, from, subject, body, cc, bcc){
  gmail <-
    gmailr::gm_mime(to = to, cc = cc, bcc = bcc,
                    subject = subject, body = body)
  return(gmail)
}

```

ファイルを添付する `attach_files_gmail()` では、複数のファイルを添付できるようにしています。ここでは、ファイル名はカンマ区切りとしており、`stringr::str_split_1()` で分割します。`gm_attach_file()` でファイルを添付しますが、複数ファイルに対応するため `purrr::reduce()` で処理します(2.8 節を参照)。

コード 12.19 (mail-attach-files-fun.R) : ファイルを添付する(複数対応)

```

attach_files_gmail <- function(gmail, files){
  if(is.na(files) | files == ""){ # 添付ファイルなし
    return(gmail)
  }
  gmail <-
    stringr::str_split_1(files, ",") |> # カンマで分割
    purrr::reduce(gmailr::gm_attach_file, .init = gmail)
  return(gmail)
}

```

`auto_gmails()` を実行すると、`send` が "1" のメールはそのまま送信します。`send` が "0" のメールは下書きフォルダに保存します。関数ができたので、実際にテストメールを送ってみましょう。

テストデータは、`tibble` で作成し(2.3 節を参照)、`openxlsx` でエクセルとして保存します(9.4 節を参照)。また、列幅、オートフィルタ、ウィンドウ枠を設定します(9.6 - 9.8 節を参照)。

コード 12.20 (mail-auto-gmails-test-data.R) : テストデータの作成

```

path <- fs::path_temp("email.xlsx")
tibble::tibble(
  send = c("1", "0"),
  to = "matutosi@gmail.com",
  cc = "",
  bcc = "",
  subject = paste0("テストメール", 1:2),
  body = paste0("本文", 1:2),
  attachment = paste0(fs::path(fs::path_package("magick")),
                     "images", c("man.gif", "building.jpg")),
  collapse = ","))
|>
openxlsx::write.xlsx(path)

```

```

wb <- openxlsx::loadWorkbook(path)
openxlsx::setColWidths(wb, 1, cols = 1:7, widths = "auto") # 列幅
openxlsx::addFilter(wb, 1, cols = 1:7, rows = 1) # オートフィルタ
openxlsx::freezePane(wb, 1, firstRow = TRUE) # ウィンドウ枠
openxlsx::saveWorkbook(wb, path, overwrite = TRUE)

```

なお、保存したエクセルファイルは図 12.13 のとおりです。

sel	to	c	t	subject	body	attachment
1	matutosi@gmail.com			テストメール1	本文1	C:/Users/matutosi/AppData/Local/R/win-library/4.3/magick/images/man.gif,C:/Users/
0	matutosi@gmail.com			テストメール2	本文2	C:/Users/matutosi/AppData/Local/R/win-library/4.3/magick/images/man.gif,C:/Users/

図 12.13: 送信用のテストデータ

コード 12.21 (mail-auto-gmails-generate.R) : メールの一斉送信

```

gmails <- auto_gmails(path = path)
gmails

```

一斉に作成したメールをブラウザで確認してから一斉送信したいこともあるでしょう。その場合は send を全て"0"にしてメールを作成後に下書きに保存してから、ブラウザで確認してください。確認後に下書きフォルダのメールを一斉送信するには、purrr::walk()を使います。

コード 12.22 (mail-auto-drafts-send.R) : 下書きメールの一斉送信

```

gmails$draft |>
  purrr::walk(gm_send_draft)
# 下書きを個別に送信するとき
# gmails$draft[[1]] |>
#   gm_send_draft()

```

12.7 Gmail での下書きメールの削除

作成した大量のメールに間違いが見つかったときには、全部削除してから作成し直すでしょう。下書きメールを全て削除するには、gm_delete_draft()を使います。gm_drafts()で得た下書きの一覧をもとに、gm_id()でメールの id を取得して、purrr::map()と gm_delete_draft()を使います。

コード 12.23 (mail-auto-drafts-delete.R) : 下書きメールの一斉削除

```
gm_drafts() |>  
  gm_id() |>  
  purrr::map(gm_delete_draft)
```

12.8 Outlook の操作の準備

12.8.1 Microsoft365R パッケージ

Outlook を R から使うには、Microsoft365R パッケージを使います。なお、Microsoft365R は Outlook だけでなく、OneDrive、SharePoint、Teams などの Microsoft のアプリも操作可能です。

コード 12.24 (mail-ms365r-package.R) : Microsoft365R のインストール

```
install.packages("Microsoft365R")
```

コード 12.25 (mail-ms365r-library.R) : Microsoft365R の呼び出し

```
library(Microsoft365R)
```

12.8.2 Outlook での認証

Outlook を使うには、認証が必要です。個人用アカウントの認証には `get_personal_outlook()` を、職場または学校アカウントの認証には `get_business_outlook()` を使用します。次のコードを実行すると、ブラウザで認証するための画面が表示されますので、アカウントとパスワードを入力します。返り値であるオブジェクトは、メールの操作で使いますので変数に保存します。

ここでは変数名を `outlook` としていますが、任意の変数名で構いません。ただし、メールを操作する関数は変数 `outlook` のメソッドになっています。そのため、別の変数名を使ったときには、以下のコードを適宜読み替えてください。

コード 12.26 (mail-ms365r-get.R) : Outlook での認証

```
# outlook <- get_personal_outlook() # 個人用アカウント  
outlook <- get_business_outlook() # 職場または学校アカウント
```

認証が成功すると以下のメッセージが表示されますので、ブラウザを閉じて R に戻ってください。

```
# 認証後にブラウザに表示されるメッセージ  
Authenticated with Azure Active Directory.  
Please close this page and return to R.
```

12.9 Outlook のメールの一覧取得

認証の返り値を保存したオブジェクトである `outlook` には、メール操作のためのが Method(メソッド、関数)が含まれています。

コード 12.27 (`mail-ms365r-outlook.R`) : outlook の内容

```
outlook
## <Outlook client for '甲南 花子'>
##   email address: yourname@outlook.co.jp
## ---
##   Methods:
##     create_email, create_folder, delete, delete_folder,
##     do_operation, get_deleted_items, get_drafts,
##     get_folder, get_inbox, get_list_pager,
##     get_sent_items, list_emails, list_folders,
##     sync_fields, update
```

`outlook$list_emails()` は受信トレイのメールの一覧を、`outlook$get_drafts()$list_emails()` は下書きのメールの一覧を、`outlook$get_sent_items()$list_emails()` は送信済みメールの一覧を取得できます。

コード 12.28 (`mail-ms365r-list-emails.R`) : メールの一覧取得

```
outlook$list_emails(n = 2) # メールを 2 つ表示
## Access token has expired or is no longer valid; refreshing
## [[1]]
## <Outlook email>
##   directory id:xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
##   from: 送信者 <yourname@outlook.co.jp>
##   sent: 2024-03-02 12:00:00 JST
##   to:宛先の氏名 <hogehoge@gmail.com>
##   subject: メールの件名
## ---
## メールの本文.....
##
## [[2]]
## <Outlook email>
##   directory id:xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
##   from: 送信者 <yourname@outlook.co.jp>
##   sent: 2024-03-02 14:00:00 JST
##   to:宛先の氏名 <fugafuga@gmail.com>
##   subject: メールの件名
## ---
## メールの本文.....
```

12.10 Outlook の新規メール作成と下書きの保存

メールを作成するには、`outlook$create_email()`を使います。`body`にはメールの本文を、`subject`には件名を入力します。`content_type`は、"text"か"html"を指定します。`to`には宛先を指定します。必要に応じて`cc`や`bcc`も指定可能です。宛先と異なるメールアドレスを返信先として指定する場合は、`reply_to`を使います。`send_now`では作成後すぐにメールを送信するか指定します。規定値は`FALSE`で、作成したメールは下書きフォルダに保存されます。`TRUE`とすると作成後すぐに送信されます。

コード 12.29 (mail-ms365r-create-email.R) : メールの作成と下書きの保存

```
body <- "メールの本文....."
body
## [1] "メールの本文....."
em <-
  outlook$create_email(
    body = body,
    content_type = "text", # "html"も可能
    subject = "メールの件名",
    to = "hogehoge@gmail.com",
    cc = NULL,
    bcc = NULL,
    reply_to = NULL,
    send_now = FALSE # FALSE: 下書きに保存, TRUE: 送信する
)
em # 作成したメールの内容
## <Outlook email>
##   directory id: xxxxxxxxxxxxxxxxxxxxxxxx
##   from:
##   sent:
##   to: "hogehoge@gmail.com"
##   subject: メールの件名
## ---
## メールの本文.....
```

上のコードを実行後すると、Outlook に下書きが保存されます(図 12.14.)

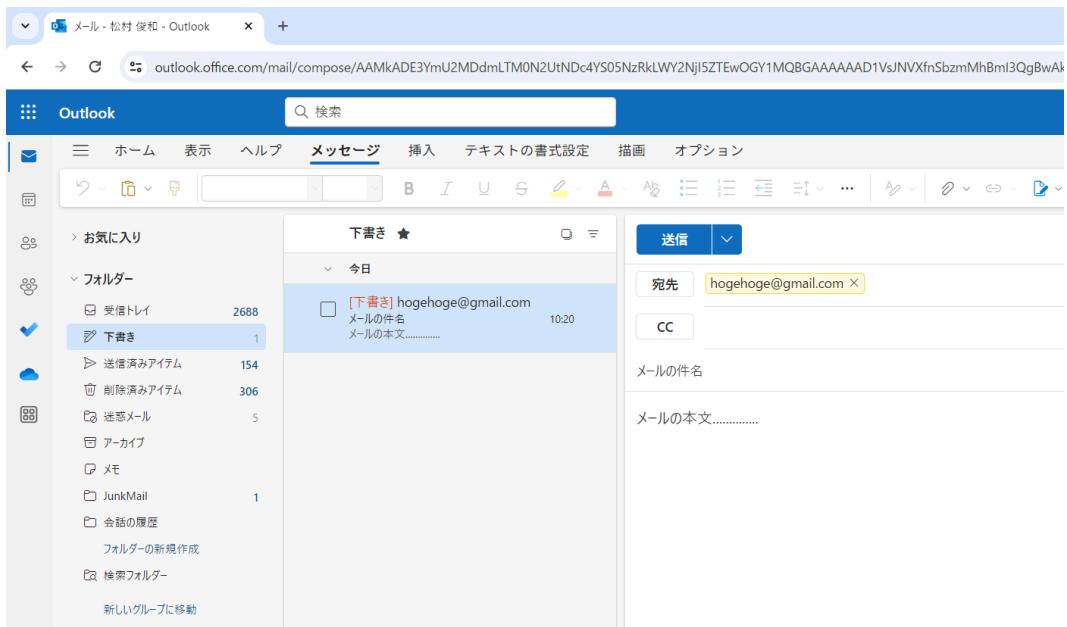


図 12.14: Outlook に保存した下書きメール

12.11 Outlook のファイルの添付

メールにファイルを添付するには `add_attachment()` を使用します。作業フォルダにあるファイルならファイル名だけで構いませんが、それ以外のときは絶対パスか相対パスで指定します。

`add_attachment()` で添付できるファイルは 1 つだけです。複数のファイルを添付するには複数回実行してください。`list_attachments()` で添付したファイルを確認できます。

コード 12.30 (mail-ms365r-add-attachment.R) : ファイルの添付

```
em$add_attachment("添付ファイル名.xlsx")
em$list_attachments()
## [[1]]
## <Outlook email attachment '添付ファイル名.xlsx'>
##   directory id:xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
##   attachment type: file
##   attachment size: 128
```

下書きに保存したメールは、`outlook$get_drafts()$list_emails()` で確認します。先ほど作成したメールがあることがわかりますが、それ以外にも下書きにメールがある場合は、[[2]] 以降も表示されます。

コード 12.31 (mail-ms365r-get-drafts.R) : 下書きフォルダのメール一覧取得

```
drafts <- outlook$get_drafts()$list_emails()
drafts
## [[1]]
## <Outlook email>
##   directory id: Axxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```

##   from:
##   sent:
##   to: hogehoge@gmail.com
##   subject: メールの件名
## ---
## メールの本文.....
## 
## [[2]]
## <Outlook email>
## # (省略)

```

12.1 Outlook のメールの送信

`outlook$create_email()`で作成して、下書きフォルダに保存したメールを送信するには `em$send()`を使います。 `outlook$get_drafts()$list_emails()`で得たメールの一覧の `drafts` からもメールを送れます。ただし、`drafts` はメールの一覧ですので、`[[1]]`のように送るメールを指定します。

コード 12.32 (`mail-ms365r-send-email.R`) : メールの送信

```

em$send() # メールの送信
# drafts[[1]]$send() # これでも送信可能

```

12.13 Outlook のメールの一斉作成と送信

これまでの内容をもとに、一斉メールの自動的な作成・送信の関数を作ります。

メールのデータはエクセルに保存しており、送信フラグ(`send`)、宛先(`to`)、件目(`subject`)、本文(`body`)、CC、BCC、添付ファイル名(`attachment`)が入力されたエクセルのファイルがあるとします。送信フラグは、"1"がすぐに送信する、"0"が下書きに保存するとします。それ宛先以降の項目は全て文字列とします。これらのエクセルのデータをもとにメールを作成して送信します。なお、複数データの繰り返しを実行するため、`purrr` パッケージ(2.8 節を参照)を多用します。

関数の全体像は次のとおりです。

- `auto_emails()`
 - `readxl::read_xlsx()`でエクセルを読み込み
 - `gen_emails()`でメールの作成
 - `attach_files_gmail()`でファイルの添付
 - メールの送信

まずは、全体を取り仕切る `auto_emails()`です。ここでは、メールのデータが入ったエクセルのパスと `outlook` を受け取ります。エクセルのパスをもとに、`readxl::read_xlsx()`でメールのデータを読み込みます。読み込んだデータをもとに、`gen_emails()`でメールを作成します。また、`send` の列があればその内容をもとにメールを送信します。

コード 12.33 (mail-ms365r-auto-emails-fun.R) : メールを自動送信する関数

```
auto_emails <- function(path, outlook){  
  df <- readxl::read_xlsx(path)  
  emails <- gen_emails(df, outlook)  
  if("send" %in% colnames(df)){  
    emails[df$send == 1] |>  
    purrr::walk(\(x){ x$send() })  
  }  
  return(emails)  
}
```

次に、メールを作成する `gen_emails()` です。 `gen_emails()` では、データフレームの `df` と `outlook` オブジェクトを受け取ります。`df` から `create_email()` に必要な列を選択して、`purrr::pmap` を使ってメールを一斉に作成します。その後、`df` に `attachment` の列があれば、`attach_files()` でファイルを添付します。

コード 12.34 (mail-ms365r-gen-emails-fun.R) : メールの作成(複数対応)

```
gen_emails <- function(df, outlook){  
  cols <- c("to", "subject", "body", "cc", "bcc")  
  emails <-  
    df |>  
    dplyr::select(dplyr::any_of(cols)) |> # 必要な列を選択  
    purrr::pmap(outlook$create_email)      # メールを作成  
  if("attachment" %in% colnames(df)){  
    purrr::walk2(emails, df$attachment, attach_files)  
  }  
  return(emails)  
}
```

ファイルを添付する `attach_files()` では、複数のファイルを添付できるようにしています。ここでは、ファイル名はカンマ区切りとしており、`stringr::str_split_1()` で分割します。`add_attachment()` でファイルを添付しますが、複数ファイルのときに備えて `purrr::walk()` で処理します。

コード 12.35 (mail-ms365r-attach-files-fun.R) : ファイルを添付する(複数対応)

```
attach_files <- function(email, files){  
  if(is.na(files) | files == ""){      # 添付ファイルなし  
    return(invisible(email))  
  }
```

```

stringr::str_split_1(files, ",") |> # ,で分割
purrr::walk(email$add_attachment) # ファイルを添付
return(invisible(email))
}

```

`auto_emails()`を実行すると、`send`が"1"のメールはそのまま送信されます。`send`が"0"のメールは下書きフォルダに保存されます。

事前に、エクセルのファイルにはコード 12.20 で作成したものと同じ内容(図 12.13)を入れておきます。

コード 12.36 (`mail-ms365r-auto-emails-generate.R`) : メールの一斉送信

```

# outlook <- get_personal_outlook() # 個人用アカウント
outlook <- get_business_outlook()      # 職場または学校アカウント
emails <- auto_emails(path = "email.xlsx", outlook)
emails

```

一斉に作成したメールをブラウザで確認してから一斉送信したいこともあるでしょう。その場合は`send`を全て"0"にして、メールを作成してから、ブラウザで確認してください。内容を確認後に下書きフォルダのメールを一斉送信するには、`send()`を使います。

コード 12.37 (`mail-ms365r-auto-drafts-send.R`) : 下書きメールの一斉送信

```

outlook$get_drafts()$list_emails() |>
  purrr::map( \(x){ x$send() })

```

12.14 Outlook の下書きメールの削除

作成した大量のメールで間違いが見つかったときには、全部削除してから作成し直します。下書きメールを全て削除するには、`delete()`を使います。`confirm = FALSE`を指定しないと、メールごとに確認画面がでてきます。

コード 12.38 (`mail-ms365r-auto-drafts-delete.R`) : 下書きメールの一斉削除

```

outlook$get_drafts()$list_emails() |>
  purrr::map( \(x){ x$delete(confirm = FALSE) })

```

13 翻訳作業の操作

ネットでの高機能な翻訳サービスはいくつかあります。そのうち、DeepLは高機能な翻訳ツールで、かなり自然な文章を提案してくれます。無料版はブラウザで誰でも簡単に使えます。また、日本発のサービスでは、国立研究開発法人情報通信研究機構が開発したみんなの自動翻訳@TexTraがあります。TexTraでは100以上の言語を扱えるのが特徴です。また、汎用モデルに加えてブログや新聞などの翻訳から特許や科学論文など分野ごとのモデルがあります。

両サービスともPDFやワードのファイルを翻訳することは可能ですが、ブラウザでのアップロードは結構手間のかかる作業です。どちらもAPIがありますので、PDFやワードから文章を抽出(8.8節、7.8節を参照)すれば、翻訳作業を自動化できます。

本章では、DeepLとTexTraへの登録からAPIキーの取得方法、それぞれを扱うパッケージでの作業について紹介します。

13.1 DeepLの操作の準備

DeepLをRから使うには、パッケージのインストールとDeepLへの登録をして、APIキーを取得する必要があります。

13.1.1 deeplrパッケージ

次のコードでdeeplrのインストールと呼び出しをします。

コード 13.1 (translate-install.R) : deeplrのインストールと呼び出し

```
install.packages("deeplr")
library(deeplr)
```

13.1.2 DeepLへの登録

DeepLを使うには、事前に以下のURLで利用登録してください。

<https://www.deepl.com/ja/pro#developer>

プランには、無料のものと有料のものがあります。無料プランは月あたり50万字を利用できます。有料プランでは、利用制限はありませんが利用量に応じた料金がかかります。クレジットカード情報を登録する必要がありますが、手動でアップグレードしない限り料金は発生しません。

登録後、ログインしてから右上の自分のイニシャルのアイコンから「アカウント」を選択してください(図13.1)。



図 13.1: アカウントの選択

API キーのタブを選択するとキーが表示されますので、API キーをコピーします(図 13.2).

名前	APIキー	作成日	ステータス
	ce94*****9fx		有効

図 13.2: API キーの表示とコピー

翻訳する前に、API キーを変数に代入しておきます。ここでは、API キーを明示的に入力していますが、他人が閲覧可能なコードには書くのはおすすめしません。次項のように、他人からアクセスできない場所に保管しておくほうがよいでしょう。

コード 13.2 (translate-set-key.R) : キーの設定

```
deepl_key <- "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXX:XX"
```

13.1.3 API キーの保存

コードに直接 API キーを書いていると、何かの拍子に情報が漏れる可能性があります。他人がアクセスできない場所に保存しておき、R の起動時に情報を読み込むことで安全性が高くなります。Windows であれば c:\Users\USERNAME\Documents に.Renvironment というファイルを作成して、情報を保存すれば R を起動時に環境変数として読み込まれます。

まずは、次のコードでディレクトリを開きます。

コード 13.3 (translate-api-key-dir.R) : ユーザのドキュメントのディレクトリを開く

```
fs::path(Sys.getenv("HOME")) |> # c:\Users\USERNAME\Documents  
shell.exec() # ディレクトリを開く
```

パーミッションの関係で R からはファイルを直接作成できない可能性がありますので、手作業でファイルを新規作成します。テキストエディタ等で `.Renvironment` を開いて、API キーを以下のように保存します。

```
DEEPL_API_KEY=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx:xx
```

保存後は R の起動時にこのファイルが読み込まれます。環境変数は `Sys.getenv()` で取得できます。`Sys.getenv("DEEPL_API_KEY")` で設定した API キーを取得できます。なお、引数を指定しないと、全ての環境変数が表示されます。

コード 13.4 (`translate-api-sys-getenv.R`) : 環境変数の読み込み

```
deepl_key <- Sys.getenv("DEEPL_API_KEY")
deepl_key
## [1] "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx:xx"
```

13.1.4 利用可能な言語の一覧

利用可能な言語の一覧は、`available_languages2()` あるいは `available_languages()` で取得できます。無料版のときは関数名の末尾に "2" が付くもの、有料版のときはないものを使います。以下の他の関数も同様です。ここでは全て無料版の例を示します。

コード 13.5 (`translate-available-languages.R`) : 利用可能な言語の一覧

```
langs <- available_languages2(deepl_key)
head(langs, 14)
## # A tibble: 29 × 2
##   language name
##   <chr>     <chr>
## 1 BG        Bulgarian
## 2 CS        Czech
## 3 DA        Danish
## 4 DE        German
## 5 EL        Greek
## 6 EN        English
## 7 ES        Spanish
## 8 ET        Estonian
## 9 FI        Finnish
## 10 FR       French
## 11 HU       Hungarian
## 12 ID       Indonesian
## 13 IT       Italian
## 14 JA       Japanese
```

13.2 DeepL での翻訳の自動化

DeepL で翻訳するには、`translate2()`あるいは`translate()`を使います。引数`target_lang`には翻訳後の言語を指定します。`source_lang`を指定しないと、自動的に判別されます。

コード 13.6 (`translate-translate.R`) : 翻訳の例

```
text <- "This is an example of a translation by DeepL."
translate2(text = text, target_lang = "JA",
           source_lang = "EN", auth_key = deepl_key)
## [1] "これはDeepLによる翻訳の例である。"
text <- "I am a cat. I have no name. It is fine today." # 夏目漱石の「吾輩は猫である」
translate2(text = text, target_lang = "JA",
           source_lang = "EN", auth_key = deepl_key)
## 私は猫だ。名前はない。今日も元気だ。
```

上の例では、1文だけの翻訳ですが、本来は長い文章を翻訳するでしょう。ここでは、テキストファイルの英語を日本語に翻訳するとします。

ここでは次のコードで`stringr`内の文を繋げて、テキストファイルとして保存します。PDF とワードの文書からテキストを取り出すときには、`pdftools`による文字列の抽出(7.8)と`officer`による文書の相互変換(8.8)をそれぞれ参考にしてください。

コード 13.7 (`translate-writelines.R`) : 翻訳用の文書の保存

```
path <- fs::path_temp("sample.txt")
head(sentences) # stringr のデータ
paste0(sentences[1:30], collapse = " ") |>
  writeLines(path) # テキストファイルで保存
# shell.exec(path)
```

PDF やワードから取り出した大量の文書を、自動的に翻訳するときには注意が必要です。翻訳するテキストが30キロバイトを超える場合は、エラーが発生します。`split_text()`を利用するとき、テキストを指定したサイズで分割できます。引数`max_size_bytes`の既定値は29000バイト(29キロバイト)です。次のコードでは読み込んだ文書を複数に分割するために、`max_size_bytes = 500`と指定します。

コード 13.8 (`translate-readlines.R`) : 翻訳用の文書の読み込みと分割

```
en <-
  path |>
  readLines() |>
  split_text(max_size_bytes = 500)
en
```

```

## # A tibble: 3 × 3
##   text_id segment_id segment_text
##       <int>      <int> <chr>
## 1         1          1 The birch canoe slid on the smooth planks...
## 2         1          2 The source of the huge river is the clear ...
## 3         1          3 Two blue fish swam in the tank. Her purse ...

```

文章全体が3つに分割され、segment_text列に本文があります。

コード 13.9 (translate-translated.R) : deeplrによる翻訳(for版)

```

text <- en$segment_text
translated <- list()
len <- length(text)
for(i in 1:len){
  translated[i] <-
    translate2(text[i], target_lang = "JA", auth_key = deepl_key)
}
translated <- unlist(translated)
translated
## [1] "白樺のカヌーは滑らかな板の上を滑った。紺色の背景にシートを..."
## [2] "大河の源は清流。ボールをまっすぐ蹴り、フォロースルー。女性..."
## [3] "水槽には2匹の青い魚が泳いでいた。彼女の財布は無駄なゴミで..."

```

《Tips》 上のコードでは purrr::map()ではなく、あえて for ループを使用しています。DeepL の上限や通信状況などによって実行中にエラーが発生したときに、map()では返り値が得られないためです。for ループであれば、途中までの結果は変数 translated に残っているため、結果が全く得られないという事態は避けられます。

《Tips》 purrr::map()を使う場合は、purrr::possibly()で事前に関数をエラー対応の関数にします。引数 otherwise には、エラー発生時の返り値を設定します。以下のコードでは、エラー発生時には"!翻訳エラー"という文字列が返り値として得られます。

コード 13.10 (translate-translated-safely.R) : deeplrによる翻訳(map版)

```

text <- en$segment_text
translate2 Possibly <- purrr::possibly(translate2, otherwise = "!翻訳エラー")
translated <-
  text |>
    purrr::map_chr(translate2_Possibly, target_lang = "JA", auth_key = deepl_key)
translated
## (2番目がエラーのときの結果の例)
## [1] "白樺のカヌーは滑らかな板の上を滑った。紺色の背景にシートを..."
## [2] "!翻訳エラー"
## [3] "水槽には2匹の青い魚が泳いでいた。彼女の財布は無駄なゴミで..."

```

翻訳内容を確認するときに、文ごとに原文と翻訳結果を対照できると便利です。そこで文を分割するとともにデータフレームとして整形してからエクセルに出力します。

まずは文を分割する関数を定義します。次のコードでは正規表現を用いて、"."(ドット)あるいは"。(句点)を置換して、".EOS"(EOSはEnd Of Sentenceの意味)あるいは"。EOS"に置換しています。その後、"EOS"を区切り文字として分割します。なお、stringrパッケージの関数と正規表現は第3章を参照してください。

コード 13.11 (translate-split-sentence-fun.R) : 文に分割する関数

```
split_sentence <- function(x){  
  x <-  
    x |>  
    stringr::str_replace_all("([。。] *)", "\\1EOS") |> # 区切り文字の挿入  
    stringr::str_split("EOS") |> # 区切り文字で分割  
    unlist()  
  return(x[x != ""]) # ""(空文字列)を除去  
}
```

文に分割する関数を使って、データフレームに変換します。

コード 13.12 (translate-split-text.R) : 文への分割

```
result <-  
  tibble::tibble(en = split_sentence(text),  
                 jp = split_sentence(translated)) |>  
  print()  
## # A tibble: 30 × 2  
##   en                               jp  
##   <chr>                            <chr>  
## 1 "The birch canoe slid on the smoo..."  "白樺のカヌーは滑らかな板の上..."  
## 2 "Glue the sheet to the dark blue ..."  "紺色の背景にシートを糊付けする."  
## 3 "It's easy to tell the depth of a..."  "井戸の深さを知るのは簡単だ."  
## 4 "These days a chicken leg is a ra..."  "最近、鶏のモモ肉は珍しい料理だ."  
## 5 "Rice is often served in round bo..."  "ご飯は丸い茶碗に盛られること..."  
## 6 "The juice of lemons makes fine p..."  "レモンの果汁はいいパンチを作る."  
## 7 "The box was thrown beside the pa..."  "箱は駐車中のトラックの横に投げら"  
## 8 "The hogs were fed chopped corn a..."  "豚には刻んだトウモロコシと生..."  
## 9 "Four hours of steady work faced ..."  "4時間の地道な作業が待っていた."  
## 10 "A large size in stockings is har..."  "ストッキングの大きいサイズはな..."  
## # i 20 more rows
```

原文と翻訳結果の対応が見やすくなりました。最後に、エクセル形式として保存します。一旦ファイルに書き込みますが、その後列幅を自動で設定してから再度書き込みます。なお、openxlsxパッケージの関数は、第9章を参照してください。

コード 13.13 (translate-write-xlsx.R) : 翻訳結果をエクセルに書き込み

```
path <- fs::path_temp("sample.xlsx")
openxlsx::write.xlsx(result, path) # エクセルに書き込み
wb <- openxlsx::loadWorkbook(path) # 読み込み
openxlsx::setColWidths(wb, 1, cols = 1:2, width = "auto") # 列幅の変更
openxlsx::saveWorkbook(wb, path, overwrite = TRUE) # 書き込み
# shell.exec(path)
```

エクセルのファイルを開くと原文と翻訳結果を見比べられます(図 13.3)。

A	B
1 en	jp
2 The birch canoe slid on the smooth planks.	白樺のカヌーは滑らかな板の上を滑った。
3 Glue the sheet to the dark blue background.	紺色の背景にシートを糊付けする。
4 It's easy to tell the depth of a well.	井戸の深さを知るのは簡単だ。
5 These days a chicken leg is a rare dish.	最近、鶏のモモ肉は珍しい料理だ。
6 Rice is often served in round bowls.	ご飯は丸い茶碗に盛られることが多い。
7 The juice of lemons makes fine punch.	レモンの果汁はいいパンチを作る。
8 The box was thrown beside the parked truck.	箱は駐車中のトラックの横に投げられた。
9 The hogs were fed chopped corn and garbage.	豚には刻んだトウモロコシと生ゴミが与えられた。
10 Four hours of steady work faced us.	4時間の地道な作業が待っていた。
11 A large size in stockings is hard to sell.	ストッキングの大きいサイズはなかなか売れない。
12 The boy was there when the sun rose.	太陽が昇った時、その少年はそこにいた。
13 A rod is used to catch pink salmon.	竿でピンクサーモンを釣る。
14 The source of the huge river is the clear spring.	大河の源は清流。
15 Kick the ball straight and follow through.	ボールをまっすぐ蹴り、フォロースルー。
16 Help the woman get back to her feet.	女性が立ち上がるのを助ける。
17 A pot of tea helps to pass the evening.	一杯のお茶が夕暮れのひとときを癒してくれる。
18 Smoky fires lack flame and heat.	煙のような火は炎と熱に欠ける。
19 The soft cushion broke the man's fall.	柔らかいクッションが男の転倒を防いだ。
20 The salt breeze came across from the sea.	海から塩の風が渡ってきた。
21 The girl at the booth sold fifty bonds.	ブースにいた少女は50枚の債券を売った。
22 The small pup gnawed a hole in the sock.	小さな子犬が靴下に穴を開けた。
23 The fish twisted and turned on the bent hook.	魚は曲がった針に絡みついで回った。
24 Press the pants and sew a button on the vest.	ズボンを押さえ、ベストのボタンを縫う。
25 The swan dive was far short of perfect.	白鳥の飛び込みは完璧にはほど遠かった。
26 The beauty of the view stunned the young boy.	その景色の美しさに少年は感動した。
27 Two blue fish swam in the tank.	水槽には2匹の青い魚が泳いでいた。
28 Her purse was full of useless trash.	彼女の財布は無駄なゴミでいっぱいだった。
29 The colt reared and threw the tall rider.	仔馬が反り返り、背の高い騎手を投げた。
30 It snowed, rained, and hailed the same morning.	同じ朝、雪が降り、雨が降り、雹が降った。
31 Read verse out loud for pleasure.	喜びのために詩を声に出して読む。

図 13.3: エクセルでの翻訳結果の表示

13.3 DeepL を使った文章の改善

文章を一旦別の言語に翻訳してから元の言語に翻訳し直すことで、正しい表現になることがあります。 `pimp2()`あるいは `pimp()`を使うとこれができます。

コード 13.14 (translate-pimp2.R) : 中間言語を使った文章の改善

```
text <- "In former times I lived in Kobe"
pimp2(text = text, source_lang = "EN", help_lang = "JA", auth_key = deepl_key)
## [1] "I used to live in Kobe."
text <- "私の大きい兄弟は、仕事を教師です。" # 変な日本語
pimp2(text = text, source_lang = "JA", help_lang = "EN", auth_key = deepl_key)
## [1] "私の兄は教師をしている。"
```

外国語の練習として、自分が書いた文章を `pimp2()` で修正することで、外国語の上達につながるかもしれません。大量の文書を自動的に処理するには、コード 13.9 を参考にしてください。

13.4 DeepL の利用量の確認

利用済みの文字数と上限を確認するには、`usage2()` あるいは `usage()` を使います。
`$character_count` が利用済み数、`$character_limit` が上限です。大量の文章を翻訳する前に
は、利用可能量を確認しておくと良いでしょう。

コード 13.15 (`translate-usage.R`) : 利用量の確認

```
usage2(deepl_key)
## $character_count
## [1] 2024
## $character_limit
## [1] 500000
```

13.5 TexTra の操作の準備

`Textra` を R から利用するには、パッケージをインストールするとともに、`TexTra` へ登録して認証情報取得する必要があります。

13.5.1 textrar パッケージ

次のコードで `textrar` をインストールと呼び出しします。

コード 13.16 (`translate-textrar-install.R`) : `textrar` のインストールと呼び出し

```
install.packages("textrar")
library(textrar)
```

13.5.2 TexTra への登録と認証情報の取得

`TexTra` の API を利用するには、登録が必要です。非商用での利用のみですが、無料で利用できます。

登録は以下の URL から行ってください(図 13.4)。

<https://mt-auto-minhon-mlt.ucri.jgn-x.jp/>



図 13.4: TexTra への登録

登録したら、ログインして API キー等を取得します。サイトの上にある雲形のアイコンを選択し、「Web API 一覧」の画面で「OAuth2.0 トークン取得」をクリックすると、認証のための情報が表示されます(図 13.5)。



図 13.5: TexTra の認証情報の取得

API キーと API secret が必要ですので、これらをコピーして他人がアクセスできないところに保存してください。R の起動時に読み込まれる R の環境変数として保存する方法は、以下のように保存すると良いでしょう。詳細は、13.1.3 節を参照してください。

```
TEXTRA_KEY="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"      # API キー
TEXTRA_SECRET="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" # API secret
```

コード 13.17 (textra-auth.R) : TexTra の認証情報の取得

```
# 環境変数に保存したとき
# textra_key <- Sys.getenv("TEXTRA_KEY")
# textra_secret <- Sys.getenv("TEXTRA_SECRET")
textra_key <- "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"      # API キー
textra_secret <- "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"      # API secret
name <- "LOGIN_ID"                                # ログイン ID
```

```
params <- gen_params(key = textra_key, # 認証情報
                      secret = textra_secret, name = name)
```

13.6 TexTraでの自動翻訳

`textrar`で翻訳するには、`textra()`を使います。引数 `text` には翻訳するテキストを、`params` には `gen_params()`で取得した認証情報を、`model` には翻訳モデルを、`from` には原文の言語を、`to` には翻訳後の言語をそれぞれ指定します。`model` の既定値は"transLM"です。通常は既定値か汎用モデルの"generalNT"を使えばよいでしょう。利用可能な主なモデルは表 13.1 を参考にしてください。

表 13.1: TexTraで利用できる主なモデル

モデル	指定方法
新エンジン	"transLM"
汎用	"generalNT"
特許	"patentNT"
金融	"fsaNT"
法令契約	"lawsNT"
ニュース	"news"
日常会話	"seikatsu"
サイエンス	"science"

`from` と `to` には多くの言語が使えます。利用可能な主な言語は表 13.2 を参考にしてください。また、利用できる言語コードの全体は以下の URL で閲覧可能です。

https://mt-auto-minhon-mlt.ucr.i.jgn-x.jp/content/help/detail.html?q_pid=FAQ_ETC

表 13.2: TexTraで利用できる主な言語

言語	指定方法
日本語	"ja"
英語	"en"
韓国語	"ko"
中国語(繁体字)	"zh-TW"
中国語(簡体字)	"zh-CN"
ドイツ語	"de"
スペイン語	"es"

言語	指定方法
フランス語	"fr"
イタリア語	"it"

モデルによってどのように翻訳結果が異なるのか試してみます。

コード 13.18 (translate-textra-models.R) : モデルによる翻訳の違い

```
sample <- "I am a cat. I have no name. It is fine today."
textra(sample, params = params) # 新エンジン(既定値)
## [1] "私は猫です。名前はありません。今日はいい天気です。"
textra(sample, params = params, model = "patentNT") # 特許
## [1] "私は猫であり、名前はない。今日は良い。"
textra(sample, params = params, model = "seikatsu") # 日常会話
## [1] "私は猫を飼っています。名前は書いてありません。今日はお天気もいいです。"
```

モデルによって、翻訳結果が異なります。目的に合わせたモデルを使ってください。

DeepLで翻訳したのと同じ文章(コード 13.9 を参照)を TexTra でも翻訳します。また、コード 13.12 と同様に、文ごとに翻訳結果を比較します。ただし、TexTra では 2 つの文をつなげて 1 つの文として翻訳結果を返す場合があります。そのため、事前に text を split_sentence() (コード 13.11)で文に分割してから、翻訳します。

なお、TexTra の API 利用では、1 回あたり 4 キロバイトに文字列長が制限されています。長い文章の場合は deeplr パッケージの split_text() で max_size_bytes = 4000 と指定するか、次のコードのように split_sentence() で文を区切るかしてください。

コード 13.19 (translate-textrar.R) : TexTra による翻訳

```
models <- c("transLM", "patentNT", "seikatsu")
text <- split_sentence(text)
len <- length(text)
translated_models <- list()
for(model in models){
  for(i in 1:len){
    translated_models[[model]][i] <- textra(text[i], params, model = model)
  }
}
result2 <- tibble::as_tibble(translated_models)
result <- dplyr::bind_cols(result, result2) |>
  print()
## # A tibble: 30 × 5
##       en          jp      transLM      patentNT      seikatsu
##       <chr>        <chr>     <chr>        <chr>        <chr>
## 1 <chr>        <chr>     <chr>        <chr>        <chr>
```

## 1 "The birch"	白樺の...	樺のカヌーは滑...	カバノキカヌー...	白樺のカヌーは...
## 2 "Glue the "	紺色の...	シートを濃い青...	シートを暗青色...	シートを濃いブ...
## 3 "It's easy"	井戸の...	井戸の深さは簡...	井戸の深さを知...	井戸の深さはわ...
## 4 "These day"	最近、 ...	最近では、鶏も...	今日、鶏の脚は...	最近は、鶏のも...
## 5 "Rice is "	ご飯は...	お米は、丸いお...	米はしばしば丸...	ご飯は、丸い器...
## 6 "The juice"	レモン...	レモンの果汁は...	レモンの汁は、 ...	レモンの果汁で...
## (以下省略)				

以下は、コード 13.13 とほぼ同じです。エクセルに書き込むとより見やすくなります(図 13.6)。

```
path <- fs::path_temp("sample.xlsx")
openxlsx::write.xlsx(result, path) # エクセルに一旦書き込み
wb <- openxlsx::loadWorkbook(path) # 読み込み
openxlsx::setColWidths(wb, 1, cols = 1:5, width = 40) # 列幅の変更
openxlsx::saveWorkbook(wb, path, overwrite = TRUE) # 書き込み
# shell.exec(path)
```

	A	B	C	D	E
1	en	jp	translM	patentNT	seikatsu
2	The birch canoe slid on the smooth plank.	白樺のカヌーは滑らかな板の上を滑った。	カバノキカヌーは滑らかな厚板上で滑った。	白樺のカヌーは滑らかな板の上を滑った。	
3	Glue the sheet to the dark blue background.	紺色の背景にシートを糊付けする。	シートを濃い青の背景に接着します。	シートを暗青色の背景に接着する。	シートを濃いブルーの背景に合わせます。
4	It's easy to tell the depth of a well.	井戸の深さを知るのは簡単だ。	井戸の深さは簡単にわかる。	井戸の深さを知ることは容易である。	井戸の深さはわかりやすいですね。
5	These days a chicken leg is a rare dish.	最近、鶏のモモ肉は珍しい料理だ。	鶏のもも肉は最近珍しい料理です。	今日、鶏の脚は希少な料理である。	最近は、鶏のもも肉が珍しくなりました。
6	Rice is often served in round bowls.	ご飯は丸い茶碗に盛られることが多い。	ご飯は丸いお椀に入っていることが多い。	米はしばしば丸いボウルで給仕される。	ご飯は丸い器に入っています。
7	The juice of lemons makes fine punch.	レモンの果汁はいいパンチを作る。	レモン汁は、上品なパンチを作る。	レモンの汁は、細かいパンチを作る。	レモンの果汁でいい香りがします。
8	The box was thrown beside the parked truck.	箱は駐車中のトラックの横に投げられその箱は駐車していたトラックの横にこの箱を駐車したトラックの脇に投げ置いてあったトラックの横に段ボール箱。			
9	The hogs were fed chopped corn and garlic.	豚には刻んだトウモロコシと生ゴミが豚には細かく切ったトウモロコシとゴミタブには、切り刻んだトウモロコシと生ゴミ犬は、刻んだトウモロコシと生ゴミ			
10	Four hours of steady work faced us.	4時間の地道な作業が待っていた。	4時間の安定した作業に直面した。	4時間も立派な労働が待っていました。	
11	A large size in stockings is hard to sell.	ストッキングの大きいサイズはなかなかストッキングの大きいサイズは売れない。	販売が大型サイズのストッキングは販売が大型サイズのストッキングは売れない。	女性を助けあげてください。	
12	The boy was there when the sun rose.	太陽が昇った時、その少年はそこにいた。	太陽が昇ったとき、少年はそこにいた。	太陽が昇っているとき、男の子はそこ	
13	A rod is used to catch pink salmon.	竿でピンクサーモンを釣る。	ロッドは、マスを釣るために使いますロッドを使用して桃色のサケを捕獲するロッドは、ピンク色のサーモンのキャ		
14	The source of the huge river is the clear sea.	大河の源は清流。	その巨大な川の源は清らかな泉である巨大な川の源は、明らかな春である。	川幅が広くて清らかなところが泉質で	
15	Kick the ball straight and follow through.	ボールをまっすぐ蹴り、フォロースルボールをまっすぐ蹴って、続けて打つボールをまっすぐにキックし、フォローボールをまっすぐ蹴って、あとは追			
16	Help the woman get back to her feet.	女性が立ち上がるのを助ける。	女性が立ち直れるよう手助けしてあげるのを助ける。	女性が足を戻るのを助ける。	
17	A pot of tea helps to pass the evening.	一杯のお茶を夕暮れのひとときを癒し紅茶を1杯飲むと夜が過ごしやすくな茶のポットは夕方を過ぎるのに役立つティーポットで一晩寝かせておくべき			
18	Smoky fires lack flame and heat.	煙のような火は炎と熱を欠く。	煙のようないは炎と熱を欠く。	煙のようないは炎と熱を欠く。	
19	The soft cushion broke the man's fall.	柔らかいクッションが男の転倒を防ぎ柔らかいクッションが男の落下を防ぎ柔らかいクッションが男性の転倒を防			
20	The salt breeze came across from the sea.	海から塩の風が渡ってきた。	海からは塩気のある風が吹き込んでき海から塩の微風がやってきた。	海から塩の風が吹いてきました。	
21	The girl at the booth sold fifty bonds.	ブースにいた少女は50枚の債券を売った。	ブースにいた少女は50枚の債券を買ってくれた。	少女の手は50枚の債券を売った。	
22	The small pup gnawed a hole in the sock.	小さな子犬が靴下に穴を開けた。	その小さな子犬は靴下に穴を開けた。	靴下に小さな穴が開いていました。	
23	The fish twisted and turned on the bent hook.	魚は曲がった針に絡みついて回った。	その魚は曲がった釣針にひっかかった魚は曲がったフックの上にねじれて巨魚がひっくり返ってしまった。	針が曲	
24	Press the pants and sew a button on the zipper.	ズボンを押さえ、ベストのボタンをズボンを押さえ、ベストのボタンを			
25	The swan dive was far short of perfect.	白鳥の飛び込みは完璧にはほど遠か	スワンダイブは完全には程遠いものでスワンダイブは完全ではありませんでした。		
26	The beauty of the view stunned the young.	その景色の美しさに少年は唖然としたその景色の美しさに、少年はうっとりビュの美しさは、若い少年を気絶させその眺めの美しさが、その若者をたた			
27	Two blue fish swam in the tank.	水槽には2匹の青い魚が泳いでいた。	二匹の青い魚が水槽の中を泳いでいた2匹の青い魚がタンク内で泳いた。	水槽の中を泳いでいる青い魚が2匹で	
28	Her purse was full of useless trash.	彼女の財布は無駄なゴミでいっぱいいた彼女の財布は、役に立たないゴミで彼女のハンドバッグは無用のゴミで彼女の財布は、使い物にならないゴミ			
29	The colt reared and threw the tall rider.	仔馬が反り返り、背の高い騎手を投げた。	その仔馬は身を起こし、背の高い騎手を投げた。	騎乗員が背が高くて背の高いライダー	
30	It snowed, rained, and hailed the same morning.	雪が降り、雨が降り、雹が降る朝は雪が降り、雨が降り、そして同じ朝、雪が降り、雨が降った。	雪が降って、また朝にはひょうが降		
31	Read verse out loud for pleasure.	喜びのために詩を声に出して読む。	朗読を楽しむ。	喜びのために大きな声で読みなさい。喜びで声を出して読むんですよ。	

図 13.6: エクセルファイルでの翻訳結果の比較

14 AI の操作

AI による自然言語生成や画像生成は仕事にも大きな影響を及ぼしています。自然言語の生成では、アイデア出し、コードの修正や提案、文章の翻訳や校正ができます。また、画像ファイルをもとにその説明を求めることが可能です。ChatGPT や Gemini など多くの AI はブラウザ経由で利用可能で、対話的ときはこの方法が適しています。

多くの AI では API が整備されており、R からの作業を自動化できます。多くの文書や画像を扱う時には手間を減らすことができます。プログラミング中の質問は、R や RStudio を使いながら対話することができるため、アプリ内で操作が完結できるのは便利です。また、API では回答の創造性や多様性を決めるパラメータを細かく指定することができますので、この点もブラウザ経由での利用よりも優れています。

本章では、自然言語でのやり取りができる ChatGPT と Gemini および画像生成 AI の DALL-E について、API を使った利用方法を説明します。なお、説明する内容は 2024 年 5 月時点のものです。

14.1 ChatGPT の操作の準備

ChatGPT を R からインストールするには、OpenAI に利用登録してから API キーを取得するとともに、パッケージをインストールする必要があります。

14.1.1 OpenAI の利用登録

【対応】 本項が長いとのことでしたので、思い切って説明をバッサリと削除しました。もし、削除し過ぎでしたらご連絡ください。

ChatGPT の API を利用するには、OpenAI の利用登録が必要です。以下の URL にアクセスして、「Get started」をクリックしてください

<https://openai.com/product>

既にアカウントがあるときや Google 等のアカウントを使うときは、各アカウントでログインしてください。ログインしたら、API キーの取得(14.1.2 項)に移動してください。

はじめて使う場合はメールアドレス、パスワード、氏名、生年月日などの必要な項目を設定してください。なお、途中で電子メールでの認証や画像での認証をする場合がありますので、適宜作業してください。

図 14.1 のような画面が出てくれば、利用登録とログインは成功です。

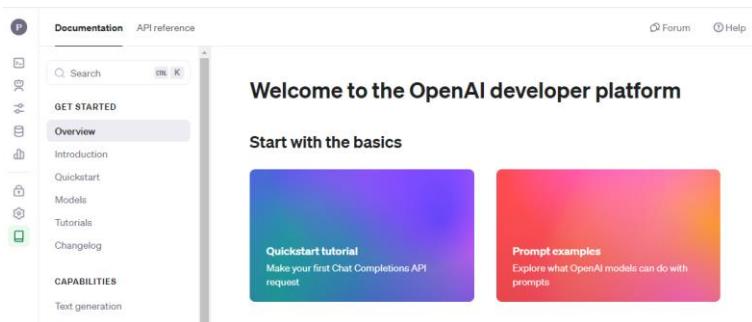


図 14.1: ログイン後の画面

14.1.2 API キーの取得

【相談】 この項も長い気がするので、不要そうな図には「不要(?)」とつけています。削除しても問題ないものを教えてもらえるとありがたいです。

過去に API を利用したことがある場合は、API key(図 14.2)を選んでから API キーを取得してください。

はじめて API キーを取得するには、事前に支払い方法を設定します。

支払い方法は、Settings(図 14.2)から、Billing - Add payment details と進みます(図 14.3)。

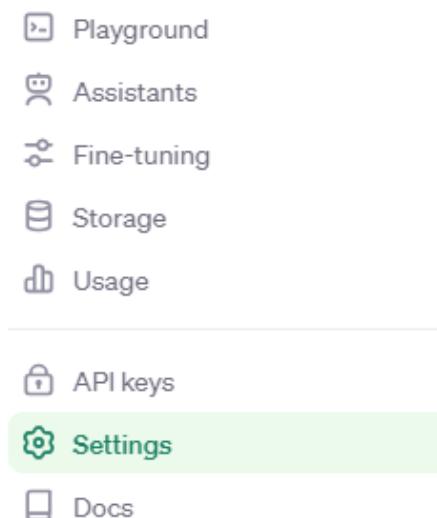


図 14.2: メニュー表示

Settings

Billing

Organization

Personal

General

Members

Billing

Limits

Overview Payment methods Billing history Preferences

Credit remaining ⓘ \$0.00

Add payment details View usage

図 14.3: Billing - Add payment details を選択

その後、個人利用のときは Individual を選択します(図 14.4)。

What best describes you? ×

<input type="radio"/> Individual	>
I'm an individual	
<input type="radio"/> Company	>
I'm working on behalf of a company	

図 14.4: 利用方法の選択(不要?)

支払い方法を選んで、カード番号等を入力します(図 14.5)。

Add payment details ×

Add your credit card details below. This card will be saved to your account and can be removed at any time.

Card information		
<input type="text"/> カード番号	<input type="text"/> 月 / 年	<input type="text"/> セキュリティコード
Name on card		
<input type="text"/>		
Billing address		
Country		
<input type="text"/> Address line 1		
<input type="text"/> Address line 2		
<input type="text"/> City	<input type="text"/> Postal code	
<input type="text"/> State, county, province, or region		

Cancel Continue

図 14.5: カード番号等の入力(不要?)

電話番号による認証をします。Start verification を選択します(図 14.6)。

Project API keys

Verify your phone number to create an API key Start verification

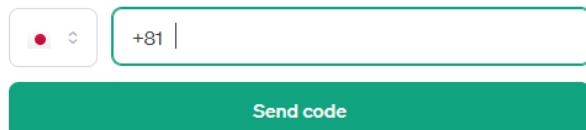
As an owner of this project, you can view and manage all API keys in this project.
Do not share your API key with others, or expose it in the browser or other client-side code. In order to protect the security of your account, OpenAI may also automatically disable any API key that has leaked publicly.
View usage per API key on the [Usage page](#).

Create Create an API key to access the OpenAI API.

図 14.6: Start verification を選択

次に、携帯電話の番号を入力します(図 14.7)。+81 の次に、携帯電話の番号から最初の 0 を削除したものを入力します。「090-xxxx」のときは、「90-xxxx」とします。Send code をクリックすると、携帯電話に認証コードが送られます。

Verify your phone number



A screenshot of a mobile application interface for phone number verification. At the top left is a small icon with a red dot and a dropdown arrow. Next to it is a text input field containing '+81'. Below the input field is a green button labeled 'Send code'.

図 14.7: 電話番号の入力(次とどちらか一方で OK?)

携帯電話等に携帯電話に認証コードが送られるので入力します(図 14.8).

Enter code

[Go back](#)

Please enter the code we just sent you.



A screenshot of a mobile application interface for entering a verification code. It shows a text input field with the code '000 000' entered. Below the input field is a green button labeled 'Resend code'.

図 14.8: 認証コードの入力(前とどちらか一方で OK?)

API キーの生成画面になるので、必要であれば Name を入力して、Create secret key をクリックします(図 14.9).

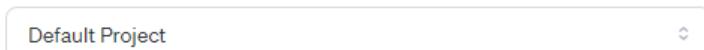
Create new secret key

Name Optional



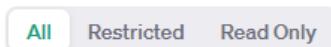
A screenshot of a mobile application interface for creating a secret key. The 'Name' field is labeled 'My Test Key'.

Project



A screenshot of a mobile application interface for creating a secret key. The 'Project' dropdown is set to 'Default Project'.

Permissions



A screenshot of a mobile application interface for creating a secret key. The 'Permissions' tab bar includes 'All' (selected), 'Restricted', and 'Read Only'.

[Cancel](#)

[Create secret key](#)

図 14.9: API キーの生成

生成された API キーをコピーして保存します(図 14.10). API キーは、ChatGPT を使用する際に使用します。14.1.4 項で、環境変数として設定しますが、他人には知られないようにしてください。

Save your key

Please save this secret key somewhere safe and accessible. For security reasons, **you won't be able to view it again** through your OpenAI account. If you lose this secret key, you'll need to generate a new one.



図 14.10: API キーのコピー

14.1.3 chatgpt パッケージ

ChatGPT を利用するには、chatgpt パッケージを使います。インストールしてから呼び出します。

コード 14.1 (ai-chatgpt-install.R) : chatgpt のインストールと呼び出し

```
install.packages("chatgpt")
library(chatgpt)
```

14.1.4 API キーの設定

chatgpt パッケージで ChatGPT を使うには、取得した API キーを環境変数として設定します。コードに直接書き込む場合は次のようにしますが、R の起動時に自動的に読み込む方法は、13.1.3 項を参考にしてください。

コード 14.2 (ai-chatgpt-api-key.R) : API キーの設定

```
Sys.setenv(OPENAI_API_KEY = "xx-xxxxxxxxxxxxxxxxxxxx")
```

14.2 ChatGPT でのチャット

ChatGPT に質問するには ask_chatgpt() を使います。引数には、質問を文字列で指定します。なお、API キーは上記で設定したものが自動的に読み込まれます。

コード 14.3 (ai-chatgpt-ask-chatgpt.R) : ChatGPT への質問

```
# (ChatGPT からの回答は読みやすく調整、以下同様)
ask_chatgpt("R の特徴を教えてください。")
```

```

## *** ChatGPT input:
## R の特徴を教えてください。
## "R は統計解析やデータ可視化のためのプログラミング言語で、"
## "オープンソースソフトウェアです。"
## "データを扱うための豊富なライブラリやパッケージが提供されており、"
## "グラフィカルな表現も容易です。"
## "R は統計学やデータサイエンスの分野で広く利用されています。"

```

`ask_chatgpt()`で続けて質問した場合、以前の質問・回答などの履歴が引き継がれます。たとえば、英語のことを教えてもらうために、ChatGPTに"あなたは優秀な英語の教師です。"と設定を伝えることができます。これ以降は、ChatGPTは英語の教師として振る舞ってくれます。一方、セッションをリセットして新しい設定で質問したいときには、`reset_chat_session()`を使います。

コード 14.4 (ai-chatgpt-ask-reset-chat-session.R) : セッションのリセット

```

ask_chatgpt("あなたは優秀な英語の教師です。")
## *** ChatGPT input:
## あなたは優秀な英語の教師です。
## "ありがとうございます！お手伝いが必要でしたらいつでもお知らせください。"
ask_chatgpt("あなたは何の教師ですか。")
## *** ChatGPT input:
## あなたは何の教師ですか。
## "私はあらゆる種類の英語教育や言語学習に関するお手伝いをするよう"
## "プログラムされています。"
## "勉強や練習に関する質問、文法や表現についての説明、学習の助言など、"
## "どんなことでもお気軽にお聞きください。"
## (以降も、これまでの履歴が引き継がれる)
reset_chat_session() # セッションをリセット
ask_chatgpt("あなたは何の教師ですか。")
## *** ChatGPT input:
## あなたは何の教師ですか。
## "私はあなたがコンピュータや技術関連の質問にお答えするための"
## "アシスタントです。"
## "それ以外の教師としての役割は担いませんが、"
## "何か質問があれば気軽にお聞きください。"

```

14.3 ChatGPT でのコードの改善

`chatgpt`には R のコードを改善するための関数が多く用意されています(表 14.1)。それぞれの関数は入力した内容の前に追加する文字列以外は同じ内容です。たとえば、`explain_code()`では "Explain the following R code:" ("次の R のコードの説明をしてください")を追加しています。他の関数で追加している文字列を知りたいときは、関数の中身を見てください。

表 14.1: R のコード改善のための関数

関数	回答
comment_code()	コードのコメント
complete_code()	コードの完成形
create_variable_name()	返り値の変数名
explain_code()	コードの説明
find_issues_in_code()	問題点・バグ
optimize_code()	最適化したコード
refactor_code()	リファクタリングしたコード
document_code()	roxygen2 形式のドキュメント
create_unit_tests()	testthat 形式のテスト

explain_code()で実行すると、コードの説明をしてくれます。

コード 14.5 (ai-chatgpt-explain-code.R) : コードの説明

```
code <- '
df <- readr::read_csv("sample.csv") |>
  filter(df, yr == 2024) |>
  arrange(name)
'

explain_code(code)
## *** ChatGPT input:
## Explain the following R code:
## df <- readr::read_csv('sample.csv') |>
##   filter(df, yr == 2024) |>
##   arrange(name)
##
## "This R code reads a CSV file named 'sample.csv' into a data frame ..."
## "Then, it uses the pipe operator `|>`, which is denoted by `|>`, t..."
## "`filter(df, yr == 2024)`: This filters the rows of the `df` data f..."
## "`arrange(name)`: This arranges the filtered data frame in ascendi..."
## "the final result will be a subset of the original data frame `df` ..."
## "sorted by the `name` column."
```

何もしてないと、回答が英語になりました。Sys.setenv()で環境変数を設定すれば回答言語を変更できます。もちろん「回答は日本語でお願いします。」と prompt に書くと日本語で回答してくれます。

コード 14.6 (ai-chatgpt-explain-code-jp.R) : 日本語でのコードの説明

```

Sys.setenv(OPENAI_RETURN_LANGUAGE = "Japanese")
code <- '
df <- readr::read_csv("sample.csv") |>
  filter(df, yr == 2024) |>
  arrange(name)
'

explain_code(code)
## *** ChatGPT input:
## Explain the following R code:
## df <- readr::read_csv('sample.csv') |>
##   filter(df, yr == 2024) |>
##   arrange(name)
##
## "このコードは、\"sample.csv\"というファイルからデータフレームを読み込みます。"
## "次に、読み込んだデータフレームを 2024 年(yr == 2024)にフィルタリングし、"
## "その後に名前(name)で昇順に並び替えます。"
## "データフレームの読み込みからフィルタリング、"
## "並び替えまでの処理がパイプ演算子(|>)を使って連鎖的に行われています。"

```

14.4 ChatGPT のパラメータの調整

ChatGPT には回答を設定できるパラメータがあります。パラメータを設定する環境変数は表 14.2 のとおりで、先頭の OPENAI_ を除去した部分を小文字にしたものが ChatGPT での本来のパラメータです。OPENAI_MODEL は ChatGPT Plus に契約していれば gpt-4 も使えます。OPENAI_MAX_TOKENS は使用する最大トークン数で、大きくすると一度に多くの回答を得られます。ただし、API では消費トークン数に応じた料金がかかります。

ChatGPT の API では、1 回の質問・回答を合わせて 4,097 トークンが上限です。英語では 1 単語が 1 トークン、日本語ではひらがな 1 文字が 1 トークンで漢字は 2-3 トークンとされることが多いようです。そのため、日本語ではあまり長い文章は扱うことはできません。

OPENAI_TEMPERATURE を 0 にすると回答が固定され、回答の多様性がなくなります。

OPENAI_TOP_P は ChatGPT の回答候補のうちどの程度上位のものを回答するのか決定します。つまり、0 に近いほど上位のものだけが選択されるようになります。ただし、OPENAI_TEMPERATURE と OPENAI_TOP_P は、似た機能を持つので、回答の多様性を制御したいときは、どちらか一方だけ変更すれば構いません。両方を設定しても、結局はどちらかの閾値で回答は決まるようです。また、OPENAI_TEMPERATURE を大きくしそすぎたり、OPENAI_TOP_P を小さくしそすぎたりすると、回答が意味の通らないものになります。

OPENAI_FREQUENCY_PENALTY と OPENAI_PRESENCE_PENALTY は似た役割で、大きくするとこれまでに出現したトークンが選ばれにくくなり、回答の幅が広がります。違いは、出現頻度(frequency)によるか、出現の有無(presence)によるかです。両方とも小さくしそすぎると、文章の意味が通らないものになります。

表 14.2: ChatGPT の回答を制御する環境変数

環境変数	内容	既定値 (範囲)
OPENAI_MODEL	モデル	gpt-3.5-turbo
OPENAI_MAX_TOKENS	最大トークン	256
OPENAI_TEMPERATURE	回答のばらつき	1 (0~2)
OPENAI_TOP_P	上位候補の選ばれやすさ	1 (0~1)
OPENAI_FREQUENCY_PENALTY	頻出トークンの選ばれにくさ	0 (-2~2)
OPENAI_PRESENCE_PENALTY	既出トークンの選ばれにくさ	0 (-2~2)

OPENAI_TEMPERATURE を 0 と 1.5 と設定して試してみます。

コード 14.7 (ai-chatgpt-parameter-temperature.R) : パラメータによる回答の制御

```
# 以前の内容を引き継がないように、毎回セッションをリセット
library(chatgpt)
prompt <- "「昔々あるところに」に続けて自由な物語を考えてください。"
Sys.setenv(OPENAI_TEMPERATURE = 0)
reset_chat_session()
ask_chatgpt(prompt)
## "昔々あるところに、美しい森が広がっていました。"
## "その森には色とりどりの花や、さえずる小鳥たちが住んでいました。"
## "ある日、森の奥深くに住む妖精たちが、"
## "森の平和を守るために大切な宝石を盗まれてしまいました。"
## "妖精たちは困り果て、森の中をさまよいながら、"
## "誰か助けを求めるにしました。"
## "そのとき、勇敢な若者が森に迷い込んできました。"
## "若者は妖精たちの悩みを聞き、"
## "宝石を取り戻すために立ち上がる決意しました。"
## "若者は森の中を冒険し、"
## (以下省略)
reset_chat_session()
ask_chatgpt(prompt)
## "昔々あるところに、美しい森が広がっていました。"
## "その森には色とりどりの花や樹木が生い茂り、"
## "小さな動物たちがのんびりと過ごしていました。"
## "ある日、森の奥深くに住む小さな妖精たちが、"
## "大切な宝石を失くしてしまいました。"
## "宝石は妖精たちの力の源であり、"
## "失くしてしまったことで森の魔法が弱まってしまいました。"
## "妖精たちは困り果てていましたが、勇敢な少女が現れました。"
## "彼女は森の宝石を見つけ出し、"
## "妖精たちに返してあげることを決意しました。"
```

```

## "少女は森の中を冒険し、さまざまな試続きを読む。"
## (以下省略)
Sys.setenv(OPENAI_TEMPERATURE = 1.5)
reset_chat_session()
ask_chatgpt(prompt)
## "昔々あるところに広大な森がありました。"
## "森は季節を問わず、緑豊かで生命に満ちあふれていました。"
## "その森には美しい湖が広がり、水は透明で清冽でした。"
## "ある日、森の中で小さなウサギが誕生しました。"
## "周りのすべての生き物から愛されるほどのかわいらしいウサギでした。"
## "ウサギはさすらう音楽家のキツネと偶然出会いました。"
## "キツネはウサギの歌声を聞いて感動し、"
## "二人は旅を共にすることになりました。"
## "長い冒険の中でウサギとキツネは、さまざまな生き物と出会い、"
## "様々な困難を乗り越していきました。"
## (以下省略)
reset_chat_session()
ask_chatgpt(prompt)
## "昔々あるところに、美しい庭園と温泉が自慢の西方の国がありました。"
## "ある日、その国に大変お金持ちで権力を持つ商人がやってきました。"
## "彼は新しい企業を立ち上げることを決意し、"
## "庭園や温泉を自社が運営するために、王様から買収しようと企てていたのです。"
## "王様はその商人の企みを知り、困り果てていました。"
## "そこで、若くて有能な庭師の少女が「商人に負けたくないなら、"
## "庭園や温泉の存在価値をさらに高めましょう」と提案しました。"
## "王様はその提案に賛同し、少女の計画通り庭園に美しい古代物。"
## "少女は庭園を四季折々に美しくする計画を練り、"
## (以下省略)

```

OPENAI_TEMPERATURE が 0 の場合は 1 回目も 2 回目は似た内容です。若干の違いはありますが、話の流れがほぼ同じです。1.5 ではセッションごとに内容が大きく異なります。

通常の ChatGPT ではパラメータは設定できませんが、API を利用するとパラメータの設定が可能です。アイデア出しのように幅広い出力が欲しいときには、OPENAI_TEMPERATURE をやや大きめに設定すると良いでしょう。ただし、上のコードで試した OPENAI_TEMPERATURE = 1.5 では途中から意味不明の内容になることがあります。コード 14.7 の最後の回答の下から 2 行目の行末にも意味不明の文字列("古代"の部分)があります。また、回答がエラーになることがありますので、大きすぎる数値はおすすめしません。

Copilot (Bing Chat)で、会話のスタイルとして「創造的に」「バランスよく」「厳密に」から選べますが、これは同様のパラメータを設定していると考えられます。手作業での変更はできますが、やや面倒です。パラメータをいくつか設定してそれぞれに対する出力を得るコードを一度書いておけば、R から自動的に回答を取得できます。また、新たな質問をするときにも使いまわしができます。

14.5 DALL-E での画像生成

DALL-E という AI で画像生成ができます。OpenAI が提供していますので、API キーは ChatGPT と同じものが使えます。

14.5.1 openai パッケージ

画像生成には、openai パッケージを使いますので、インストールと呼び出しをします。

コード 14.8 (ai-chatgpt-openai.R) : openai のインストールと呼び出し

```
install.packages("openai")
library(openai)
```

14.5.2 画像生成

画像生成には、create_image()を使います。引数 prompt には求める画像の説明を、n には生成する画像の数(既定値は 1)を、size は "1024x1024"(既定値), "256x256", "512x512" のいずれかを、response_format には回答形式を "url"(既定値)か "b64_json" で指定します。

生成した画像の URL は create_image() の返り値の \$data\$url にあります。画像の有効期限は 60 分ですので、早めにダウンロードしてください。

コード 14.9 (ai-chatgpt-create-image-jp.R) : DALL-E による画像生成(日本語での指示)

```
prompt <- "背景は白色で光り輝く「R」という文字を生成してください。"
n <- 5
response <- create_image(prompt, n = n, size = "256x256")
response$data$url
## "https://oaidalleapiprodscus.blob.core.windows.net/private/..." # 長いので省略
## "https://oaidalleapiprodscus.blob.core.windows.net/private/..."
pngs <- fs::path(fs::path_home(), "desktop", paste0(1:n, ".png"))
purrr::map2(response$data$url, pngs, curl::curl_download)
```

日本語でも画像を生成してくれますが、どうもうまくいかないときがあります。図 14.11 の右から 4 つ目は、R ではない文字が生成されました。



図 14.11: DALL-E で生成した画像(日本語での指示)

必ずではありませんが、日本語よりも英語の方が適切な画像を生成してくれる傾向があるようです。英語が不得意なときは、prompt を ask_chatgpt() で翻訳すると良いでしょう。

コード 14.10 (ai-chatgpt-create-image.R) : DALL-E による画像生成(英語での指示)

```
ask_chatgpt(paste0("英語に翻訳してください。\\n", prompt))
## "Please generate a glowing letter \"R\" on a white background."
prompt <- "Please generate a glowing letter \"R\" on a white background."
response <- create_image(prompt, n = n, size = "256x256")
pngs <- fs::path(fs::path_home(), "desktop", paste0((n+1):(n*2), ".png"))
purrr::map2(response$data$url, pngs, curl::curl_download)
```

英語のときは、図 14.12 のような画像が生成されました。さらに求めるイメージに近い画像が欲しいときは、説明を具体的にすると良いでしょう。



図 14.12: DALL-E で生成した画像(英語での指示)

14.6 Gemini の操作の準備

Gemini を利用するには、Google AI Studio にログインしてから API キーを取得する必要があります。まずは、以下の URL にアクセスしてください。

<https://aistudio.google.com/>

14.6.1 API キーの取得(AI Studio の利用経験がないとき)

Google AI Studio を利用したことがない場合は、まずは以下の URL にアクセスしてください。

<https://aistudio.google.com/>

URL で右上の「ログインから」 Google のアカウントでログインして、「Google AI Studio にログイン」をクリックします(図 14.13)。



図 14.13: Google AI Studio へのログイン

「Get API Key」をクリックします(図 14.14)。

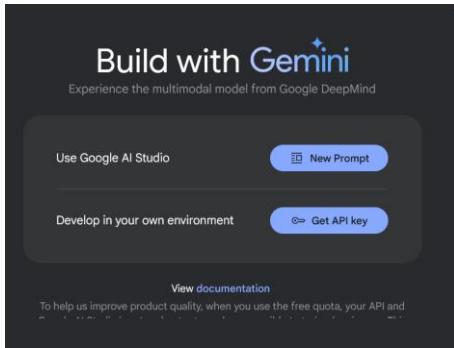


図 14.14: 「Get API Key」をクリック

「Create API Key」をクリックすると、API キーの一部が表示されるのでその部分をクリックします(図 14.15)。

ポップアップ画面で出てきた API Key をコピーします(図 14.16)。

Create API key			
Your API keys are listed below. You can also view and manage your project and API keys in Google Cloud.			
API key	Google Cloud project name	Created	Action
...gCM8	Generative Language Client	Mar 17, 2024	[Copy]

図 14.15: 「Create API Key」をクリック

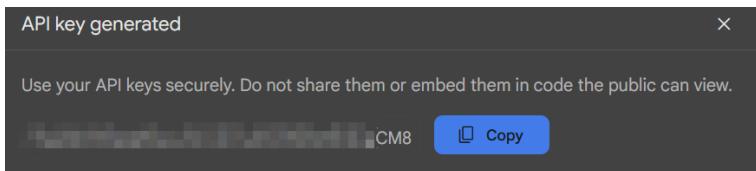


図 14.16: API Key のコピー

14.6.2 API キーの取得(AI Studio の利用経験があるとき)

既に Google AI Studio を利用中でログイン済みの場合は、左側のメニューから「Get API Key」をクリックします(図 14.17)。

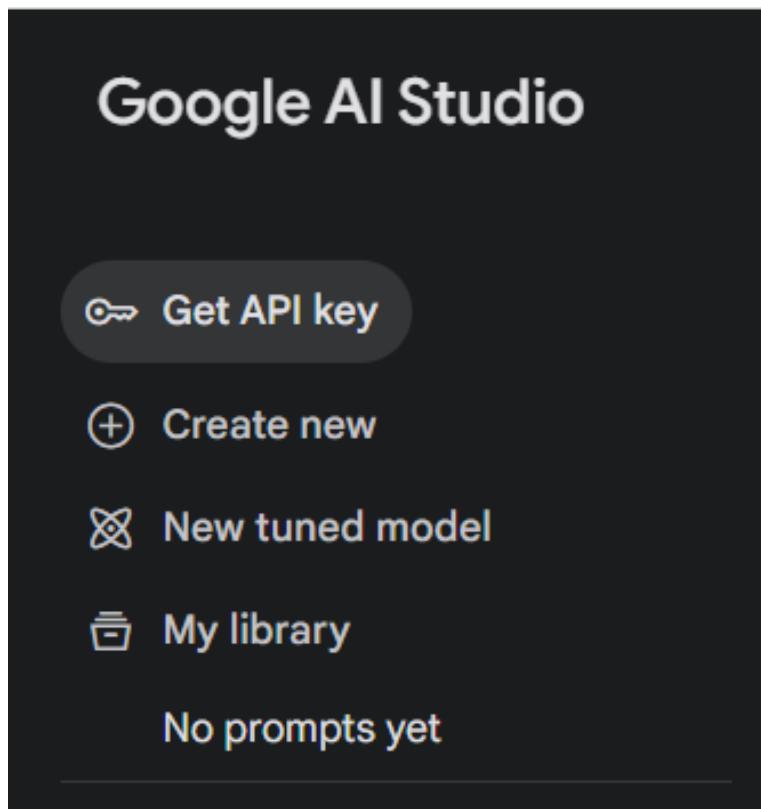


図 14.17: 「Get API Key」をクリック

「Create API Key」をクリックすると、API Key の一部が表示されるのでその部分をクリックします(図 14.15)。

ポップアップ画面で出てきた API Key をコピーします(図 14.16)。

14.6.3 gemini.R パッケージ

Gemini を R から利用するには、gemini.R パッケージを使います。CRAN には登録されていませんので、GitHub からインストールします。インストールしてから呼び出します。

コード 14.11 (ai-gemini-r-install.R) : gemini.R のインストールと呼び出し

```
remotes::install_github("jhk0530/gemini.R")
library(gemini.R)
```

14.6.4 API キーの設定

Gemini を使うには、`setAPI()`で API キーを設定します。以下ではコードに直接 API キーを書き込んでいますが、環境変数 "GEMINI_API_KEY" にキーを設定しても構いません。`setAPI()`は、引数 `gemini_api_key` を環境変数 "GEMINI_API_KEY" に設定しているだけだからです。なお、R の起動時に自動的に環境変数として設定する方法は、13.1.3 項を参考にしてください。

コード 14.12 (ai-gemini-api-key.R) : API キーの設定

```
gemini_api_key <- "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"  
setAPI(gemini_api_key)
```

14.7 Gemini でのチャット

Gemini ではトークン数の上限は 16,384 トークンですので、比較的長い文章のやりとりが可能です。英語では 1 単語が 1 トークン、日本語では 1 文字が 1 トークンと扱われることが多いようです。ここでは Gemini で文章を要約するために、その文章を準備します。『学問の自由』(寺田寅彦, 1985)を青空文庫からダウンロードして、テキストを取り出します。rvest パッケージによるスクレイピングは 15.4 節を、文字列の置換は 3.5 節を参考にしてください。

PDF やワードの文章の要約や文書の内容について質問するときには、文書の内容をテキストファイルに変換してから文章を読み込みます。具体的な方法は、pdfTools による文字列の抽出(7.8)や officer による文書の相互変換(8.8)を参考にしてください。

また、必要に応じて stringr で文字列を整形してください。

コード 14.13 (ai-gemini-rvest.R) : 作業用文章の準備

```
# 寺田寅彦 学問の自由  
url <- "https://www.aozora.gr.jp/cards/000042/files/43535_24583.html"  
text <-  
  url |>  
  rvest::read_html() |>  
  rvest::html_elements(".main_text") |>  
  rvest::html_text() |>  
  stringr::str_remove_all("\s+") # 空白文字を削除  
stringr::str_sub(text, 1, 34)      # 1-34 文字目の表示  
## [1] "学問の研究は絶対自由でありたい。これはあらゆる学者の「希望」である。"
```

Gemini で質問するには、gemini()あるいは gemini_chat()を使います。gemini()は単独の質問で使います。つまり前のセッションとは関係ない質問として扱われます。もちろん、後ろのやり取りにも影響しません。

コード 14.14 (ai-gemini-gemini.R) : Gemini への質問

```
# (Gemini からの回答を読みやすく調整済み、以下同様)  
prompt <- paste0("次の文章を 200 文字程度に要約してください。 \n", text)  
gemini(prompt)  
## "学問の研究における自由は理想とされるが、現実には制限がある."  
## "社会科学では特に顕著で、自然科学でも研究者の環境や資金源に左右される."  
## (中略)  
## "しかし、科学的な視点から問題を捉えることで、"  
## "新たな洞察が得られる可能性がある。"
```

`gemini_chat()`は質問・回答などの履歴を引き継ぐことができます。`gemini_chat()`の返り値の`$outputs`はチャットの回答内容で、`$history`は履歴です。`gemini_chat()`の引数`history`に`$history`を指定することで履歴を引き継ぐことができます。

コード 14.15 (ai-gemini-gemini-chat.R) : Geminiへの連続的な質問

```
chat <- gemini_chat(prompt)
chat$outputs
## "学問の自由は理想だが、現実には制約がある。"
## "自然科学でも、個人研究所を持つ者以外は、"
## "資金や設備面で支援を受けざるを得ず、支援者からの影響を受ける。"
## (中略)
## "しかし、科学的な視点から事実を認識することは、議論の参考になる。"
prompt <- "要約をさらに短くして、1文にまとめてください。"
chat <- gemini_chat(prompt, chat$history)
chat$outputs
## "学問の自由は理想だが、現実には資金や環境、"
## "内面的な要因などにより制約される。"
prompt <- "文章には、自由な研究には何が重要だと書かれていますか。"
chat <- gemini_chat(prompt, chat$history)
chat$outputs
## "文章では、自由な研究には以下が重要だと書かれています。"
## "* **資金と設備:** 個人研究所を持つなど、資金と設備が十分にあること。"
## "* **時間:** 教育や事務などの業務に追われず、"
## "研究に専念できる時間があること。"
## (以下省略)
```

14.8 Geminiでの画像の説明

Geminiでは、画像ファイルの説明を得ることができます。ここではグラフの説明を求めます。準備として `ggplot2` パッケージ(2.7節を参照)を使って簡単なグラフを作成します(図 14.18)。

コード 14.16 (ai-gemini-image.R) : 作業用のグラフの生成

```
library(ggplot2)
gg <- fs::path_temp("gg.png")
ggplot(mpg, aes(cty, hwy)) + geom_point() + theme_bw()
ggsave(gg, width = 5, height = 5)
```

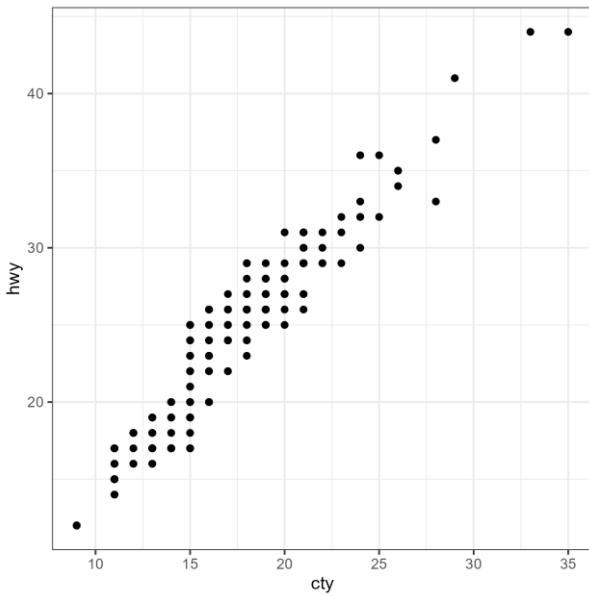


図 14.18: `gemini_image()`で使用する画像

図の説明を得るには、`gemini_image()`を使います。第1引数 `prompt` には質問内容を、第2引数 `image` には画像のパスをそれぞれ指定します。

コード 14.17 (`ai-gemini-gemini-image.R`) : Gemini でのグラフの説明

```
prompt <- "グラフの説明をしてください。"
gemini_image(prompt, gg)
## "これは散布図で、縦軸が高速道路の燃費（hwy）、"
## "横軸が街中の燃費（cty）を表しています。"
## "燃費の良い車は、cty と hwy の値が高くなります。"
## "燃費の悪い車は、cty と hwy の値が低くなります。"
## "この散布図から、cty と hwy の間に正の相関関係があることがわかります。"
## "つまり、cty の燃費が良い車は、hwy の燃費も良くなる傾向にあります。"
```

複数の画像の説明を求める場合は、次のコードのように `for` ループを使うと作業の自動化ができます。ファイルサイズが大きい写真は、時間切れでエラーになる可能性があります。そのときは、`magick` パッケージであらかじめリサイズして保存しておくとよいでしょう(第 11.12 節を参照)。

ここでは Gemini から 3 枚の写真(図 14.19)についての説明を取得します。1 つ目はアオウミガメ、2 つ目はヨセミテ国立公園、3 つ目はキク科のタムラソウという植物の写真です。

【報告・お願い】 `image` のところでも使っていますが、ウミガメの写真は提供を受けて、使用的承諾を得ています。それ以外は自前の写真です。書籍の最後に以下の記載をお願いします。「写真提供(P○○のアオウミガメ)：伊良部島海遊びガイドシャーカン」



図 14.19: `gemini_image()`で使用する画像

コード 14.18 (ai-gemini-gemini-images.R) : Gemini での図の説明

```
prompt <- "写真の全体の説明をしてください。また、生物がいる場合は、その生物の説明も  
お願いします。"  
url <- "https://matutosi.github.io/r-auto/data/"  
jpgs <- paste0(url, "image_0", 1:3, ".jpg")  
comments <- list()  
for(jpg in jpgs){  
  comments[[jpg]] <- gemini_image(prompt, jpg)  
}  
comments  
## $`https://matutosi.github.io/r-auto/data/image_01.jpg`  
## "これはウミガメが泳いでいる様子をとらえた写真です."  
## "ウミガメは、海亀科に属する爬虫類の一種で、"  
## "世界中の暖かい海に生息しています."  
## "ウミガメは、長い年月をかけて進化してきた生物であり、"  
## "その生態には多くの謎が残されています."  
##  
## $`https://matutosi.github.io/r-auto/data/image_02.jpg`  
## "これは、ヨセミテ国立公園のハーフドームの頂上にある枯れ木の写真です."  
## "枯れ木は、高さ約 10 メートル、幅約 15 メートルで、岩の上に立っています."  
## "枯れ木の周りには、山々と雲が広がっています."  
##  
## $`https://matutosi.github.io/r-auto/data/image_03.jpg`  
## "これは、ピンク色の花を咲かせたアザミの画像です."  
## "アザミは、ヨーロッパ、アジア、アフリカ原産のキク科の植物です。"  
## "草地や道端、空き地などに生息しています."  
## "アザミは、高さ 1~2 メートルになる多年草で、茎には鋭い棘があります。"  
## "葉は羽状に切れ込み、縁には棘があります."  
## "花は 6~8 月頃咲き、ピンク、赤、白などがあります."  
## "アザミは、民間薬として古くから利用されており、"  
## "止血や抗菌、解熱などの効果があると言われています。"
```

全てが正しい説明とはいえませんが、写真の概要は把握しており、それなりの説明ができるています。

画像中の文字認識をすることもできますが、AI が間違える場合もありますので、使用時には確認が必要です。例として、祝日と都道府県の一覧の画像(図 14.20)の文字認識をさせてみます。

元日	北海道	石川県	岡山県
成人の日	青森県	福井県	広島県
建国記念の日	岩手県	山梨県	山口県
天皇誕生日	宮城県	長野県	徳島県
春分の日	秋田県	岐阜県	香川県
昭和の日	山形県	静岡県	愛媛県
憲法記念日	福島県	愛知県	高知県

図 14.20: 文字認識用の画像の一部

コード 14.19 (ai-gemini-gemini-images-ocr.R) : Gemini での文字認識

```
url <- "https://matutosi.github.io/r-auto/data/jps.png"
prompt <- "画像内の文字を教えて下さい。"
str <-
  gemini_image(prompt, url) |>
  stringr::str_split_1("\n")
str
## [1] "元日"      "成人の日"    "建国記念の日" "天皇誕生日"
## [5] "春分の日"    "昭和の日"    "憲法記念日"    "みどりの日"
## [9] "こどもの日"    "海の記念日"  "山の日"        "敬老の日"
## [13] "秋分の日"    "スポーツの日" "文化の日"        "勤労感謝の日"
## [17] "北海道路"    "青森県"      "岩手県"        "宮城県"
## [21] "栃木県"       "群馬県"      "埼玉県"        "千葉県"
## [25] "東京都"       "神奈川県"   "新潟県"        "富山県"
## [29] "石川県"       "福井県"      "山梨県"        "長野県"
## [33] "岐阜県"       "静岡県"      "愛知県"        "三重県"
## [37] "滋賀県"       "京都府"      "大阪府"        "兵庫県"
## [41] "奈良県"       "和歌山県"   "鳥取県"        "島根県"
## [45] "岡山県"       "広島県"      "山口県"        "徳島県"
## [49] "香川県"       "愛媛県"      "高知県"        "佐賀県"
## [53] "長崎県"       "熊本県"      "大分県"        "宮崎県"
## [57] "鹿児島県"   "沖縄県"
```

14.9 Gemini でのパラメータの調整

gemini.R パッケージでは、パラメータを直接操作できるようにはなっていません。ただし、`gemini()`, `gemini_chat()`, `gemini_image()` のいずれの関数でも少しだけ中身を変更すれば、パラメータを操作できます。

具体的には、次のように引数 `temperature` を追加します。その内容をパラメータを設定する `generationConfig` に渡します。`temperature = 0.0` とすると決定的な回答になり、`temperature = 1.0` とすると回答の幅が広がります。

【相談】パラメータの変更のところは、とりあえず「(中略)」として入れています。本文中は以下のままで、関数全体は GitHub で示そうと思います。

コード 14.20 (ai-gemini-parameter-fun.R) : パラメータの変更

```
gemini_para <- function(prompt, temperature){  
  # (省略)  
  generationConfig = list(  
    temperature = temperature, # 既定値は 0.5  
    maxOutputTokens = 1024  
  )  
  # (省略)  
}
```

《Tips》上のコードでは、パラメータとして `temperature` だけ設定していますが、他のパラメータも同じ部分で指定可能です。設定可能なパラメータや意味は、以下の URL を参考にしてください。

<https://cloud.google.com/vertex-ai/docs/generative-ai/model-reference/gemini?hl=ja>

コード 14.20 で設定した関数を使うときには、`gemini.R` が依存するパッケージ(`httr`, `jsonlite`, `base64enc`)を事前に呼び出します。そのうえで、引数 `temperature` を設定すると回答の仕方を制御できます。

コード 14.21 (ai-gemini-library-others.R) : 各種パッケージの呼び出し

```
library(httr)  
library(jsonlite)  
library(base64enc)  
prompt <- クイズを 3 つ出してください。  
gemini_para(prompt, temperature = 0)  
## **クイズ 1**  
## 私は鍵を持っているが、ドアも窓も開けません。  
## 私は部屋を持っているが、家具はありません。  
## 私は水が入りますが、魚は泳ぎません。  
## 私は何ですか？
```

```
## **クイズ 2**
## 私はいつもお腹が空いていますが、食べると死にます。
## 私は何ですか？
## **クイズ 3**
## 私は木から生まれますが、木ではありません。
## 私は紙に書かれますが、ペンではありません。
## 私は知識を伝えますが、先生ではありません。
## 私は何ですか？

gemini_para(prompt, temperature = 0)
## **クイズ 1**
## 私は鍵を持っていますが、ドアも窓も開けません。
## 私は部屋を持っていますが、家具はありません。
## (同じ内容のため省略)

gemini_para(prompt, temperature = 0)
## **クイズ 1**
## 私は鍵を持っていますが、ドアも窓も開けません。
## 私は部屋を持っていますが、家具はありません。
## (同じ回答のため省略)

gemini_para(prompt, temperature = 1)
## **クイズ 1**
## 私は車輪があり、2つまたは4つの脚を持っていますが、歩くことはできません。
## 私は何ですか？
## **クイズ 2**
## 私は水の中で生きるが、エラはありません。私は何ですか？
## **クイズ 3**
## 私は食べられますが、あなたではありません。
## 私は話すことができますが、あなたとは違います。
## 私は何ですか？

gemini_para(prompt, temperature = 1)
## **クイズ 1**
## 私は空中にいますが、私を保持しているものは見えません。
## 私が振動すれば、音が出ます。
## 私は何ですか？
## **クイズ 2**
## 私は木ですが、葉はありません。
## 私は何本でも成長できますが、動けません。
## 私は何ですか？
## **クイズ 3**
## 私は鍵を持っていますが、錠前もドアもありません。
## 私は広くて空腹ですが、誰も食べられません。
## 私は何ですか？

gemini_para(prompt, temperature = 1)
## **クイズ 1**
## 私は山などにいるのに動かず、風になびくことはありません。
## 私は何でしょう？
## **クイズ 2**
## 私はたくさんの鍵を持っていますが、開くことはできません。
## 私は何でしょう？
```

```
## **クイズ`3**  
## 私は小さな箱の中にいて、届けられるのを待ちます。  
## 私は何でしょう？
```

`temperature = 0.0` とすると同じ回答が得られ, `temperature = 1.0` とすると回答の幅が広いことがわかります。

15 ウエブの情報収集の操作(スクレイピング)

情報収集のためにウエブページを閲覧する人は多いと思います。閲覧した中から必要な情報を手作業でコピーして、エクセルに貼り付けて整理することがあるかもしれません。一度だけの情報収集ならマウスを使った操作が適していますが、繰り返すときは手間がかかるとともに、失敗もつきものです。マウス操作でブラウザとエクセルを往復しているうちに、どこまで作業したのかわからなくなってしまうことがあります。ブラウザの画面を印刷して保存するときに、同じページを何度か印刷することや印刷忘れがでてきます。

新着情報を得るなど特定のサイトでの定期的な情報収集であれば、ウエブスクレイピングによる作業の自動化が可能です。スクレイピングでは一度コードを書いてしまえば、何度も同じことができる所以、手作業が省略できます。スクレイピングでは、ウエブページの構造でリンクをたどります。そのため、手作業のような間違いがなく、確実な作業ができます。

また、スクレイピングで得た情報をエクセル形式で保存すれば、ブラウザとエクセルの行き来は必要ありません。画面の印刷の代わりに、スクリーンショットを撮影して画像として保存することもできます。

本章ではウエブスクレイピングをするのに必要な基礎知識であるHTMLの概要をまず説明します。続いて、スクレイピングの基本を解説します。その後、スクレイピングの実例として各種サイトでのスクレイピングの方法を説明します。具体的には、CRANでのパッケージ一覧、出版社の新刊情報の、気象庁の雨雲の動きの画像を自動的に取得する方法を説明します。

スクレイピングでは、商品や店舗の検索のようなキーワードをもとにした情報収集も可能です。キーワードで検索した結果の一覧から詳細な情報を一覧として整理できます。ネットを活用した翻訳(第13章を参照)やAIによる要約(第14章を参照)と組み合わせれば、より高度な自動化が可能でしょう。

15.1 HTML 概説

15.1.1 作業用の HTML

ここでは作業用として以下のページを使用します。

<https://matutosi.github.io/r-auto/data/sample.html>

内容は非常に簡単なもので、文章、表、プルダウンの選択、リスト、画像が含まれます(図15.1)。

HTMLの例

matutosi.github.io/r-auto/data/sample.html

kw mt BiSS DL elmo iavs

ヘッダーの部分 [サポートページへ](#)

文章

idが“text_1”的本文です。

直前は改行記号です。

idが“text_2”的本文です。 直前は半角スペース2つです

表1

mpgのデータ

manuf	model	displ	year
audi	a4	1.8	1999
audi	a4	1.8	1999
audi	a4	2.0	2008

表2

irisのデータ

S.L	S.W	P.L	P.W	Sp
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

春の七草

セリ

秋の七種

- ・ハギ
- ・ススキ
- ・クズ
- ・カワラナデシコ
- ・オミナエシ
- ・フジバカラ
- ・キヨウ

画像1

図 15.1: 作業用のウェブページ

HTML の内容は次のとおりです。

```
<!doctype html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <title>HTML の例</title>
</head>
<body>
<div class="header">ヘッダーの部分
  <a href="https://github.com/matutosi/r-auto">サポートページへ</a>
</div>
<hr>
<div class="txt">
  <h1>文章</h1>
  <div id = "text_1">
    <p>id が text_1 の本文です。<br>直前は改行記号です。</p>
  </div>
  <div id = "text_2">
    <p>id が text_2 の本文です。 直前は半角スペース 2 つです</p>
  </div>
</div>
<div class="tbl">
  <h1>表 1</h1>
  <div id = "table_1">
    <table border = 1>
      <caption>mpg のデータ</caption>
      <thead>
        <tr><th>manuf</th><th>model</th><th>displ</th><th>year</th></tr>
      </thead>
      <tbody>
        <tr><td>audi</td><td>a4</td><td>1.8</td><td>1999</td></tr>
        <tr><td>audi</td><td>a4</td><td>1.8</td><td>1999</td></tr>
        <tr><td>audi</td><td>a4</td><td>2.0</td><td>2008</td></tr>
      </tbody>
    </table>
  </div>
  <h1>表 2</h1>
  <div id = "table_2">
    <table border = 1>
      <caption>iris のデータ</caption>
      <thead>
        <tr><th>S.L</th><th>S.W</th><th>P.L</th><th>P.W</th><th>Sp</th></tr>
      </thead>
      <tbody>
        <tr><td>5.1</td><td>3.5</td><td>1.4</td><td>0.2</td><td>setosa</td></tr>
        <tr><td>4.9</td><td>3.0</td><td>1.4</td><td>0.2</td><td>setosa</td></tr>
        <tr><td>4.7</td><td>3.2</td><td>1.3</td><td>0.2</td><td>setosa</td></tr>
```

```

        </tbody>
    </table>
</div>
</div>
<h1>春の七草</h1>
<div class="txt list">
    <select id="spring_species">
        <option>セリ</option><option>ナズナ</option><option>ゴギョウ</option>
        <option>ハコベラ</option><option>ホトケノザ</option>
        <option>スズナ</option><option>スズシロ</option>
    </select>
</div>
<div class="txt list">
    <h1>秋の七種</h1>
    <ul id="autumn_species">
        <li>ハギ</li><li>ススキ</li><li>クズ</li><li>カワラナデシコ</li>
        <li>オミナエシ </li><li>フジバカマ</li><li>キキョウ</li>
    </ul>
</div>
<div class="img">
    <h1>画像 1</h1>
    <div id = "image_turtle"></div>
    <h1>画像 2</h1>
    <div id = "image_R"></div>
</div>
<hr>
<div class="footer">
    <span>フッターの部分</span>
    <a href="https://www.morikita.co.jp/">森北出版へ</a>
</div>
</body>
</html>

```

15.1.2 タグ

HTML は、内容そのものを示す文字列と表示方法などの役割を与えるタグからなります。

タグは<>で囲んで記述します。たとえば、<a>(アンカー)タグはリンクの役割を与えるタグです。多くのタグには開始タグと終了タグがあります。<a>が開始タグ、が終了タグでこの間に文字列を記載します。タグには属性といわれる情報を追加することができます。<a>であれば、
`href="https://github.com/matutosi/r-auto"`のようにリンク先の URL を追加します。href を属性、=のうしろの""で囲った部分を属性値といいます。また、
`リンク説明`のように開始タグと終了タグで囲った全体を HTML の要素といいます。要素を入れ子状に組み合わせることで、HTML が構成され、ウェブページが出来上がります。

タグはたくさんありますが、表 15.1 にスクレイピングで使用する主なタグをまとめました。

表 15.1: 主なタグの役割と属性・属性値

タグ	役割	主な属性と属性値
<div> 	範囲指定	id="固有名" class="クラス名"
<p>	段落	id="固有名" class="クラス名"
<h1> - <h6>	見出し	
<a>	リンク	href="リンクの URL"
	画像	scr="画像の URL"
<form>	入力フォーム	type="フォームのタイプ"
<input>	入力欄	type="入力タイプ"
<select>	セレクトボックス	name="メニュー名"
<option>	選択肢	value="選択肢の値"
<textarea>	複数行の入力欄	
<button>	ボタン	
<table>	表	
 	リスト(順序あり/なし)	
	リストの項目	

<div>とは範囲を指定するだけです。 それ自体では文字列の表示方法はそのままですが、 CSS と合わせて使うことで指定した書式で表示されます。 id や class と合わせて使われることが多いので、スクレイピングの要素を特定する目印に使えます。 id は 1 つのページ内で固有のもので、 class は複数の要素に対して使えます。 <p>は段落を示すタグで、 id や class が使われることがあります。 id や class は上記のタグだけでなく、他のタグでも使われます。

<h1>から<h6>までのタグは、見出しを示します。 数値が小さいほど表示が大きくなります。 見出し自体を取り出することもありますし、要素の目印として使うこともできます。

<a>はリンクを貼るのに使い、 href 属性でリンク先の URL を示します。 スクレイピングではリンク先の URL を知りたいときに使えます。

は画像を挿入するのに使います。 scr 属性に画像の URL があるので、画像の URL を入手できます。

<form>, <input>, <select>, <option>, <textarea>, <button>は、入力フォームで使われます。 全体を<form>タグで囲んでその中に<input>から<button>までの各種タグが配置されることが多いです。

<table>は表を作るのに使われます。 表は各種データを表示するのに使われますので、スクレイピングではここからデータ入手することがよくあります。

は順序あり、は順序なしのリストをそれぞれ作ります。でリストの項目を示します。

上記以外にも多くのタグと属性があります。使われ方も複雑ですので、詳細はHTMLについての書籍をご覧ください。

15.1.3 要素の特定

ここでは、要素の特定方法として基本的な方法として開発者ツールを使った説明します。実際のスクレイピングでは、GoogleChromeの拡張機能であるSelectorGadgetで要素を取得するのも非常に便利です。

【相談】 SelectorGadget(最近、存在を知りました)の使用方法(I-2ページ程度だと思います)も簡単に紹介した方がよいでしょうか。開発者ツールでの方法は基本的なものですので、一応は知っておくべき方法ですが、実践的にはSelectorGadgetも便利です。日本語ではあまり紹介されていない気がします。

スクレイピングをするには、実際の表示とHTMLの内容との対応を把握しなければなりません。そのために、ブラウザの開発者ツールを使います。GoogleChromeであれば、F12で開発者ツールが表示されます。メニュー等の表示を日本語に変更するには[歯車のアイコン] - [Preferences] - [Language]で「日本語」を選んでから、F12を2回押して開発者ツールを開き直してください。

開発者ツールには、要素(Elements), コンソール(Console), ソース(Sources)などのタブがありますが、ここでは要素のタブを使います。要素には、HTMLの内容が入れ子状に折りたたんで表示されます。折りたたまれた部分は、三角形の部分をクリックすると開閉できます。

開発者ツールのHTMLの要素にマウスを移動すると、HTMLに対応したウェブ表示の部分が強調され、対応関係を特定できます(図15.2)。逆にウェブ表示に対応したHTMLから要素を特定するには、開発者ツールの左上にある矢印のアイコン(図15.3)をクリックしてから、マウスを特定したいところに移動します。そうすると、対応するHTMLの要素が強調表示されます。このようにしてスクレイピングで取得したい要素を特定できます。

文章

これはidがtext_1の文章です。
この直前に改行記号があります。

ht 509.6 x 36 kt_2の文章です。この直前には半角スペースが2つあ

春の七草

セリ

秋の七種

- ハギ
- ススキ
- クズ
- カラマツノデシコ
- ブラバエジ
- フジバガマ
- キキョウ

画像1

```
<!DOCTYPE html>
<html lang="ja">
  <head> ... </head>
  <body>
    <div class="header"> ... </div>
    <h1>春の七草</h1> == $0
    <div class="list">
      <select id="spring_species">
        <option>セリ</option> <slot>
        <option>ナズナ</option> <slot>
        <option>ゴギョウ</option> <slot>
        <option>ハコベラ</option> <slot>
        <option>ホトケノザ</option> <slot>
        <option>スズナ</option> <slot>
        <option>スズシロ</option> <slot>
      </select>
    </div>
    <div class="list"> ... </div>
  </body>
</html>
```

図15.2: HTMLに対応したウェブの強調表示



図 15.3: 開発者ツール左上のアイコン

スクレイピングでは、HTML の要素を CSS セレクタあるいは XPath で指定します。CSS(Cascading Style Sheets)は、HTML の配置・色・フォントなどの装飾の仕方を定義するものです。CSS には、要素を特定するセレクタと装飾の内容を示す宣言部分があります。このセレクタをスクレイピングでも要素を特定するために使います。XPath は、積み木や入れ子のような HTML の構造を利用して指定する方法です。

CSS セレクタと XPath は、開発者ツールの要素タブで取得します。HTML の要素を右クリックしてから、コピーを選択してから「selector をコピー」「XPath をコピー」「完全な XPath をコピー」のいずれかを押すと、CSS セレクタや XPath が取得できます。完全な XPath はファイルを指定するときのフルパスにあたります。XPath はそれ以外の方法で HTML の要素を特定する方法で、個別の XPath は HTML の内容によって異なります。



図 15.4: CSS セレクタや XPath のコピー

15.2 スクレイピング時の注意

スクレイピングをする際には、対象のサイトがスクレイピングの許可状況を事前に確認してください。また、スクレイピングが許可されていても、適切な間隔を空けるのがマナーです。通常は 5 秒以上を求めていることが多いですが、特に決まりがない場合は最低でも 1 秒は空けましょう。決められたルールを守らず頻繁なアクセスをすると、サーバに対する攻撃とみなされて訴えられる危険性があります。

スクレイピング時に適切な間隔を空けるのは、サーバ負荷の軽減だけでなく、実用的な意味もあります。十分な間隔がないと HTML の要素を完全に取得できていない可能性があります。HTML が取得できていないと、ブラウザに何も表示されていのと同じ状態です。スクレイピングでも HTML の情報がなくては、内容を分析できません。サーバからの情報を待つ意味でも適度な間隔を空けるのが望ましいです。

スクレイピングが許可されているかは、polite パッケージを使って確認できます。

コード 15.1 (scrape-bow-install.R) : polite パッケージのインストールと呼び出し

```
install.packages("polite")
library(polite)
```

bow()を使ってスクレイピング可能かどうかを確認します。引数には URL を指定します。

コード 15.2 (scrape-bow.R) : スクレイピングの許可状況の確認

```
bow("https://cran.r-project.org/web/packages/")
## <polite session> https://cran.r-project.org/web/packages/
##   User-agent: polite R package
##   robots.txt: 20699 rules are defined for 1 bots
##   Crawl delay: 5 sec
##   The path is scrapable for this user-agent
```

The path is scrapable for this user-agent とあるので、スクレイピング可能であることがわかります。また、Crawl delay: 5 sec とあるので、5秒以上の間隔が必要です。

一般的にウェブページには robots.txt というテキストファイルがあり、その中にスクレイピングのルールが記載されています。CRAN のウェブページの robots.txt には、以下のように記載されています。

<https://cran.r-project.org/robots.txt>

```
User-agent: *
Disallow: /web/packages/A3/DESCRIPTION
Disallow: /web/packages/AalenJohansen/DESCRIPTION
Disallow: /web/packages/AATtools/DESCRIPTION
(以下省略)
```

Disallow で各パッケージの DESCRIPTION は不許可であると指定されており、bow()の返り値の 20695 rules として表示されています。ただし、スクレイピングの間隔は指定されていないので、一般的な間隔である 5 秒間隔が bow()では示されています。

なお、本章で紹介するサイトは bow()でスクレイピングが許可されていることを確認済みです。

15.3 スクレイピングで使うパッケージ

R のパッケージでスクレイピングのために作られたものはいくつかあります。静的サイトのスクレイピングでは rvest パッケージが、動的サイトのスクレイピングでは RSelenium パッケージがよく使われているようです。ただし RSelenium は、selenium, Java, ブラウザのドライバをインストールするなど準備に手間がかかります。そこで、ここでは動的サイトのスクレイピングには seleniader パッケージを使います。seleniader を使うには chromote パッケージをインストールしますが、RSelenium ほどの手間はかかりません。

ところで、上記では静的サイトと動的サイトと区別しましたが、`rvest` はバージョン 1.0.4 から `read_html_live()` が導入されて動的なサイトのスクレイピングに対応するようになりました。ただし、2024 年 6 月の時点では、`selenider` の方が安定して動作するように感じています。`selenider` は動的サイトにも対応しますが、コードの記述は簡潔なため静的サイトのスクレイピングでも十分使うことができます。`rvest` と `selenider` の両方を使えるようになれば、各種サイトのスクレイピングが可能です。また、組み合わせて使うことも可能です。

`rvest` と `selenider` はどちらも CRAN に登録されていますので、`install.packages()` でインストールします。

コード 7.15 (eval.R) : `rvest` と `selenider` のインストール

```
install.packages("rvest")
install.packages("selenider")
```

`selenider` を使うときには `chromote` パッケージか `selenium` パッケージのどちらかが必要です。おすすめは `chromote` です。`selenium` は、Java のインストールや設定が必要なためです。また、`selenider` ではスクレイピング中の画面をスクリーンショットとして保存可能ですが、`showimage` パッケージが必要です。そのためこれらのパッケージをインストールしておきます。

コード 15.3 (scrape-selenider-chromote.R) : `chromote` と `showimage` のインストール

```
install.packages("chromote")
install.packages("showimage")
```

15.4 `rvest` の基本操作

15.4.1 HTML の読み込み

`rvest` で HTML の内容を読み込むには、`read_html()` を使います。第 1 引数には URL を指定します。

コード 15.4 (scrape-read-html.R) : HTML の内容の読み込み

```
url <- "https://matutosi.github.io/r-auto/data/sample.html"
html <- read_html(url)
html
## {html_document}
## <html lang="ja">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset ...
## [2] <body>\n<div class="header">ヘッダーの部分\n  <a href="https://github.com ...
```

`read_html()` の返り値は、HTML の内容を含むオブジェクトです。スクレイピングでは、このオブジェクトから要素を取得します。

15.4.2 要素の取得

HTML の要素を取り出すには、`html_elements()`を使います。第1引数には HTML のオブジェクトを指定します。その次に、取得したい要素として `css` か `xpath` を指定します。`css` と `xpath` のどちらで指定しても構いません。HTML が完全に固定されているときは、開発者ツールで取得した `css` か `xpath` を使えば問題ありません。ただし、サイトによっては PHP や JavaScript によって制御されており、HTML の内容が変化することがあります。そのため、実際のウェブページで必要な要素を取り出す際には、若干の試行錯誤が必要です。なお、CSS セレクタと XPath の詳細は HTML 関連の書籍等をご覧ください。

CSS セレクタのときは、タグ、`id`、`class` などで指定する方法があります(表 15.2)。

表 15.2: 主な CSS セレクタの指定方法

CSS セレクタ	指定方法	例
タグ	タグそのもの	"h1"
<code>id</code>	#	"#text_1"
<code>class</code>	.(ドット)	".txt"
属性名	[属性名]	"[href]"
属性名と値	[属性名='値']	"[width='10%']"

複数の CSS セレクタを指定することもできます。たとえば、`".class_A .class_B"`(半角ペースで繋げる)で `class_A` の中にある `class_B` の全て階層の要素を指定できます。また、`".class_A > .class_B"` で `class_A` の 1 階層下の `class_B` の要素を、`".class_A , .class_B"` で `class_A` または `class_B` をそれぞれ指定できます。属性名と値で要素を指定するときに、`"[alt*='hogehoge']"` のようにすると 'hogehoge' という文字列が `alt` 属性の値として含まれる要素を抽出できます。

見出しだある `<h1>` タグの要素を取り出すには "h1" と指定します。

コード 15.5 (scrape-html-element-tag.R) : タグでの要素の取得

```
h1 <- html_elements(html, "h1") # h1 タグ
h1
##  {xml_nodeset (7)}
##  [1] <h1>文章</h1>
##  [2] <h1>表 1</h1>
##  [3] <h1>表 2</h1>
##  [4] <h1>春の七草</h1>
##  [5] <h1>秋の七種</h1>
##  [6] <h1>画像 1</h1>
##  [7] <h1>画像 2</h1>
```

`id`, `class`, 属性名で取り出すには以下のようにします。

コード 15.6 (`scrape-html-element-id.R`) : `id` や属性名での要素の取得

```
html |> html_elements("#text_1") # id = "text_1"
html |>
  html_elements(".list") |>      # class = "list"
  html_elements("option")         # さらに option タグで絞り込み
html |> html_elements("[href]") # href 属性
```

`XPath` で指定するときには、引数が `xpath` であることを明示します。

コード 15.7 (`scrape-html-xpath.R`) : `XPath` での要素の取得

```
html |>
  html_elements(xpath = "/html/body/div[5]/h1")
## [1] <h1>秋の七種</h1>
```

15.4.3 文字列の取り出し

取得した要素から、ページ内容の文字列を取り出すには、`html_text()` や `html_text2()` を使います。

コード 15.8 (`scrape-html-text.R`) : 文字列の取り出し

```
html_text(h1)
## [1] "文章"      "表 1"       "表 2"       "春の七草"   "秋の七種"   "画像 1"
## [7] "画像 2"
```

`html_text()` は文字列がそのままのままの表示で取り出します。たとえば、連続して半角スペースが複数ある場合、`html_text()` では複数の半角スペースがそのまま残りますが、`html_text2()` では 1つだけになります。また、`
` は `html_text()` では除外されますが、`html_text2()` では改行記号(`\n`)に変換されます。

たとえば、作業用の HTML からパラグラフを示す `<p>` と `</p>` で囲まれた文章を取り出すと次のようになります。`html_text()` と `html_text2()` での内容の違いがわかります。

コード 15.9 (`scrape-html-text2.R`) : 文字列の取り出し(ブラウザ的な表示)

```
html |>
  html_elements("p") |>
  html_text()
```

```

## [1] "id が text_1 の本文です。直前は改行記号です。"
## [2] "id が text_2 の本文です。直前は半角スペース 2 つです"
html |>
  html_elements("p") |>
  html_text2()
## [1] "id が text_1 の本文です。 \n 直前は改行記号です。"
## [2] "id が text_2 の本文です。直前は半角スペース 2 つです"

```

15.4.4 属性値の取り出し

リンク先の URL は[<a>](#)タグの属性値に href="https://github.com/matutosi/r-auto"など、画像のパスはの属性値に src="r-01.jpg"などとしてそれぞれ指定されています。これらを取り出すことで、次に閲覧する URL や画像のパスがわかります。これらをもとに html_elements() と html_attr() を組み合わせることで、移動先や画像の URL を取り出せます。

コード 15.10 (scrape-html-attr.R) : 属性値の取り出し

```

html |>
  html_elements("a") |>
  html_attr("href")
## [1] "https://github.com/matutosi/r-auto"
## [2] "https://www.morikita.co.jp/"
html |>
  html_elements("img") |>
  html_attr("src")
## [1] "image_01.jpg" "r_07.png"

```

15.4.5 表の取り出し

スクレイピングでは表形式のデータを取り出すことがあります。そのため、rvest では表の部分を簡単に取り出す関数として html_table() が用意されています。

コード 15.11 (scrape-html-table.R) : html の読み込み

```

tables <-
  html |>
  html_table()
tables[[1]]
## # A tibble: 3 × 4
##   manuf model displ year
##   <chr> <chr> <dbl> <int>
## 1 audi  a4      1.8  1999
## 2 audi  a4      1.8  1999
## 3 audi  a4      2     2008

```

```
tables[[2]]
## # A tibble: 3 × 5
##   S.L    S.W    P.L    P.W Sp
##   <dbl> <dbl> <dbl> <dbl> <chr>
## 1 5.1    3.5   1.4   0.2 setosa
## 2 4.9    3     1.4   0.2 setosa
## 3 4.7    3.2   1.3   0.2 setosa
```

作業用の HTML には 2 つの表があります。ここでは、mpg と iris のデータの一部がそれぞれの表に入っています。表の内容はデータフレームになっているので、その後の分析などでも使いやすいです。

15.5 selenider の基本操作

15.5.1 セッションの開始

selenider では、サーバとユーザとのやり取りであるセッション(session)をもとに作業を進めます。selenider_session()でセッションを開始します。次のコードでは session に返り値を代入していますが、任意の変数名で構いません。ただし、別の変数を使ったときは、以下のコードでは session を適宜変更してください。selenider_session()の引数 session には上記でインストールした"chromote"を、タイムアウトの時間として 10 秒をそれぞれ指定しています。

コード 15.12 (scrape-selenider-session.R) : セッションの開始

```
session <- selenider_session(session = "chromote", timeout = 10)
## Setting global deferred event(s).
## i These will be run:
##   * Automatically, when the R session ends.
##   * On demand, if you call `withr::deferred_run()` .
## i Use `withr::deferred_clear()` to clear them without executing.
```

selenider についての注意が表示されます。selenider では、返り値を遅延要素として扱うことがあるという注意です。対話的に使用しているときは気にする必要はありません。ただし、selenider の関数を自分が定義する関数内で使うときには、適宜 elem_cache() を活用する方が良いでしょう。

セッションを開始してから、open_url()で指定した URL を開きます。open_url()を実行しても、表面上は何も起きません。開いているページの様子を確認するには、session\$driver\$view()を使います。

コード 15.13 (scrape-selenider-open-url.R) : URL を開いてブラウザで確認

```
url <- "https://matutosi.github.io/r-auto/data/sample.html"
open_url(url)
session$driver$view()
```

`session$driver$view()`を実行すると、開発者ツールが開いた状態のブラウザが起動して url のページが表示されます(図 15.5)。

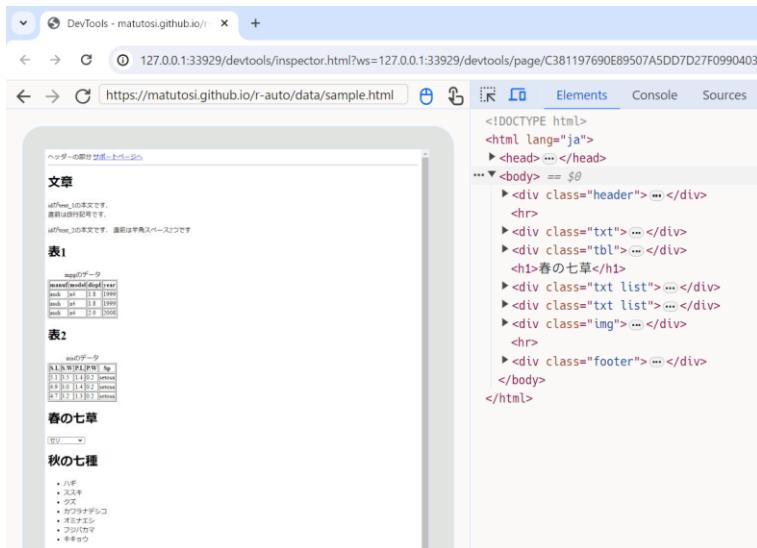


図 15.5: `session()$drive$rview()`で起動したブラウザ

15.5.2 HTML の内容の取得

rvest に慣れている場合は、selenider のセッションから HTML の内容を取り出して、rvest の関数を使うほうが楽かもしれません。その場合は、`get_page_source()`を使います。引数 `session`にはセッションを指定しますが、省略すると呼び出し時の環境のセッションが使われます。返り値は、rvest の関数と組み合わせて使えます。

コード 15.14 (scrape-selenider-get-page-source.R) : HTML の内容の取得

```
html <- get_page_source(session)
{html_document}
<html lang="ja">
[1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">\n<meta charset="UTF-8">\n<title>HTML の例</title>\n< ...
[2] <body>\n<div class="header">ヘッダーの部分\n  <a href="https://github.com/matutosi/r-auto">サポートページへ</a>\n</div>\n<hr>\n<div class="t ...>
rvest::html_table(html) |>
`[[(`_, 1)
## # A tibble: 3 × 4
##   manuf model displ year
##   <chr> <chr> <dbl> <int>
## 1 audi  a4      1.8  1999
## 2 audi  a4      1.8  1999
## 3 audi  a4      2     2008
##
```

15.5.3 要素の取得

要素を取得するには、`s()`か`ss()`を使います。`s()`は単独の要素を、`ss()`は複数の要素を返します。`s()`で複数の要素とマッチしたときには、1つ目の要素を返します。

コード 15.15 (scrape-selenider-s.R) : 要素の取得

```
s("#spring_species") # id"spring_species"
## { selenider_element }
## <select id="spring_species">
##   \n      <option>セリ</option><option>ナズナ</option><option>ゴギ...
## </select>
s("ul") # ul タグ
## { selenider_element }
## <ul id="autumn_species">
##   \n      <li>ハギ</li><li>ススキ</li><li>クズ</li><li>カワラナデシコ...
## </ul>
ss(".img") # img クラス
## { selenider_elements (1) }
## [1] <div class="img">\n  <h1>画像 1</h1>\n  <div id="image_turtle"><i...
```

XPath で指定するときには、引数が`xpath`であることを明示します。

コード 15.16 (scrape-selenider-s-xpath.R) : XPath での要素の取得

```
s(xpath = "/html/body/div[2]/h1")
## { selenider_element }
## <h1>
##   文章
## </h1>
```

その他、要素の指定方法は`rvest`と同じです。詳細は 15.4 節を参照してください。

15.5.4 文字列の取り出し

文字列を取り出すには、`elem_text()`を使います。引数には`s()`や`ss()`の返り値を指定します。ただし、`ss()`の返り値は複数の要素ですので、直接`elem_text()`は使えません。`as.list()`でリストに変換してから、`map()`等を使ってください。

コード 15.17 (scrape-selenider-text.R) : 文字列の取り出し

```
s("#text_1") |>
  elem_text()
## [1] "\n      id が text_1 の本文です。直前は改行記号です.\n      "
ss("option") |>
  as.list() |>
```

```
purrr::map_chr(elem_text)
## [1] "セリ" "ナズナ" "ゴギョウ" "ハコベラ" "ホトケノザ" "スズナ" "スズシロ"
```

15.5.5 属性値の取り出し

属性値を取り出すには、`elem_attr()`を使います。`ss()`で取り出したときの返り値は複数の要素ですので、`as.list()`と`map()`を使ってください。

コード 15.18 (`scrape-selenider-attr.R`) : 属性値の取り出し

```
ss("a") |>
  as.list() |>
  purrr::map_chr(elem_attr, "href")
## [1] "https://github.com/matutosi/r-auto" "https://www.morikita.co.jp/"
```

15.5.6 JavaScript の命令の実行

サイトによっては`rvest`や`selenider`の関数を使ってもうまく動作しないことがあります。その場合はJavaScriptのコードを実行する`selenider::execute_js_expr()`を使うと、うまくいくことがあります。

15.6 CRAN のパッケージ一覧の取得

CRANには膨大な数のパッケージが登録されています。選択肢が多いのは嬉しいのですが、欲しいものを見つけ出すのが難しいことも事実です。ここでは、CRANのパッケージ一覧の表を`rvest`で取り出し、目的とするパッケージを見つけやすくします。

まず、`read_html()`を使ってCRANのパッケージ一覧のURLからHTMLの内容を読み込みます。

コード 15.19 (`scrape-cran-read-html.R`) : HTML の内容の取得

```
url <-
  "https://cran.r-project.org/web/packages/available_packages_by_name.html"
html <- read_html(url)
html
  ## {html_document}
  ## <html>
  ## [1] <head>\n<title>CRAN Packages By Name</title>\n<link rel="stylesheet"...
  ## [2] <body lang="en">\n<div class="container">\n<h1>Available CRAN Packag...
```

次に、`rvest::html_table()`で表を取得します。パッケージの一覧は表(`table`タグ)になっています(図 15.6)。

Available CRAN Packages By Name

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
A3																										
AalenJohansen																										
AATtools																										
ABACUS																										
abasequence																										
abbreviate																										
abc																										
abc.data																										
ABC.RAP																										
ABCanalysis																										
abclass																										
ABCOptim																										
ABCp2																										
abcrf																										
abcrda																										
abctools																										
abd																										
abdiv																										
abe																										
aberrance																										

Elements Console Sources Network Performance Memory Application >

```
<!DOCTYPE html>
<html>
  <head> ...</head>
  <body lang="en">
    <div class="container">
      <h1>Available CRAN Packages By Name</h1>
      <p style="text-align: center;">...</p>
    ...<table> == $0
      <tbody>
        <tr id="available-packages-A">...</tr>
        <tr>
          <td>...</td>
          <td>Accurate, Adaptable, and Accessible Error Metrics for Predictive Models</td>
        </tr>
        <tr>...</tr>
      </tbody>
    </table>
  </body>
</html>
```

図 15.6: CRAN のパッケージ一覧

表は 1 つだけですが、リスト形式になっているので [[1]] で個別の表として取り出します。その後、列名を設定するとともに、説明(description)中の改行文字("\n")をスペースに変換します。

コード 15.20 (scrape-cran-html-table.R) : パッケージ一覧の取得

```
pkgs <-
  html |>
  html_table(header = TRUE) |>
  `[[`(_, 1) |> # [[1]]と同じ
  magrittr::set_colnames(c("pkg", "description")) |>
  dplyr::mutate(
    description = stringr::str_replace_all(description, "\n", " "))
pkgs
## # A tibble: 20,669 × 2
##   pkg      description
##   <chr>    <chr>
## 1 A3      Accurate, Adaptable, and Accessible Error Metrics for P...
## 2 AalenJohansen Conditional Aalen-Johansen Estimation
## 3 AATtools Reliability and Scoring Routines for the Approach-Avoid...
## 4 ABACUS   Apps Based Activities for Communicating and Understandi...
## 5 abasequence Coding 'ABA' Patterns for Sequence Data
## 6 abbreviate Readable String Abbreviation
## 7 abc      Tools for Approximate Bayesian Computation (ABC)
```

```

## 8 abc.data      Data Only: Tools for Approximate Bayesian Computation (ABC)
## 9 ABC.RAP       Array Based CpG Region Analysis Pipeline
## 10 ABCanalysis   Computed ABC Analysis
## # i 20,659 more rows
## # i Use `print(n = ...)` to see more rows

```

これまでの内容をまとめて、CRAN からパッケージ名とその説明を取得する関数を定義します。

コード 15.21 (scrape-scrape-cran-fun.R) : CRAN のパッケージを取得する関数

```

scrape_cran_pkgs <- function(){
  url <-
    "https://cran.r-project.org/web/packages/available_packages_by_name.html"
  html <- read_html(url)
  pkgs <-
    html |>
    html_table(header = TRUE) |>
    `[[(`_, 1) |> # [[1]]と同じ
    magrittr::set_colnames(c("pkg", "description")) |>
    dplyr::mutate(
      description = stringr::str_replace_all(description, "\n", " "))
  return(pkgs)
}

```

scrape_cran_pkgs() でパッケージの情報が入手できますので、次に検索する関数を定義します。filter() と str_detect() を使ってデータフレームを絞り込んで、pkg 列を取り出します。また、パッケージの URL をあわせて返すようにします。

コード 15.22 (search.R) : パッケージを検索する関数

```

search_cran_pkgs <- function(pkgs, pattern){
  pkgs <-
    pkgs |>
    dplyr::filter(stringr::str_detect(pkg, pattern) |
                  stringr::str_detect(description, pattern)) |>
    `$(_, "pkg")
  url <- "https://cran.r-project.org/web/packages/"
  urls <- paste0(url, pkgs)
  return(list(pkg = pkgs, url = urls))
}

```

関数が完成したので、ChatGPT 関連のパッケージを探します。複数の文字列を探すには、ブラウザでは何度も同じ操作が必要ですが、ここでは正規表現(3.4 節を参照)が使えるのは便利です。"GPT" と "OpenAI" の両方を検索するために、"GPT|OpenAI" とします。また、大文字と小文字を区別せずに検索するために、regex(ignore_case = TRUE) と指定します。

コード 15.23 (scrape-search-cran.R) : パッケージの検索例

```
pkgs <- scrape_cran_pkgs()
pattern = stringr::regex("GPT|OpenAI", ignore_case = TRUE)
pkg_gpt <- search_cran_pkgs(pkgs, pattern)

## $pkg
## [1] "askgpt"                  "chatgpt"                 "chngpt"
## [4] "detectors"                "gitGPT"                  "gptoolsStan"
## [7] "gptstudio"                "gptzeror"                "gym"
## [10] "oaii"                     "openai"                  "openair"
## [13] "openairmaps"              "openaistream"            "sparseSEM"
## [16] "symbol.equation.gpt"      "TheOpenAIR"
## $url
## [1] "https://cran.r-project.org/web/packages/askgpt"
## [2] "https://cran.r-project.org/web/packages/chatgpt"
## [3] "https://cran.r-project.org/web/packages/chngpt"
## (中略)
## [16] "https://cran.r-project.org/web/packages/symbol.equation.gpt"
## [17] "https://cran.r-project.org/web/packages/TheOpenAIR"
```

全てが目的のパッケージとは限らないので、詳細はブラウザを起動して確認します。 `walk()` と `shell.exec()` を組合せて使えば、ブラウザの起動が自動化できて、URL を 1 つずつクリックする必要がありません。ただし、検索で大量のパッケージがマッチしたときには、ブラウザのタブが大量に開きますので、注意してください。

コード 15.24 (scrape-browse-cran-shell-exec.R) : パッケージの検索例

```
purrr::walk(pkg_gpt$url, shell.exec) # ブラウザで閲覧して確認
```

15.7 新刊情報の取得

【報告：スクレイピングの負荷】 確かに数百人が一斉に試したら、ちょっとよろしくないと思います。私はサーバの管理をしたことがないので、正直なところよく分かりません。森北出版さんのサーバ管理の担当者の方に、同時アクセス可能な数を聞いてもらった方が良いかもしれません。また、同時アクセス数を超えたときにサーバがダウンするのか、動作がゆっくりになるのか、なども聞いてもらった方が良いかもしれません。（超楽観的に考えて）1万部売れたとしても同時アクセスは 1/100 ぐらいでしょうから、100 アクセスに耐えられれば大丈夫な気がします。

多くの出版社のサイトには月ごとで新刊書籍のデータがあります。複数月の新刊情報を得るにはスクレイピングで各月のデータ入手する必要があります。ここでは、`rvest` を使って森北出版の新刊書籍のデータを取得します。

森北出版の新刊情報は、以下の URL に新刊情報の一覧があります。

<https://www.morikita.co.jp/news/category/newbook>

図 15.7: 森北出版の新刊情報のページ

ここでは以下の内容を自動化するコードを作成します。

- ・ 一覧のページから各月の新刊情報のページの URL を取得
- ・ 各月のページで個別の書籍のページの URL を取得
- ・ 個別のページで書籍の著者, 価格, ページ数, ISBN, 発行年月, 内容, 目次を取り出し
- ・ 取り出した情報をデータフレーム形式に整理

まず、一覧のページで各月の新刊情報の URL の一覧を取得します。各月の新刊書籍の URL は news_item クラスにあり、そこにタグの href にリンク先の URL があります(図 15.7)。

`html_elements(".news_item")`で news_item クラスを、`html_elements("a")`でタグを指定します。さらに `html_attr("href")`で href 属性を取り出します。

(#thm:scrape-code) (scrape-.R) : 新刊情報のページ一覧を取得する関数

```
get_monthly_urls <- function(){
  Sys.sleep(5)
  "https://www.morikita.co.jp/news/category/newbook" |>
    rvest::read_html() |>
    rvest::html_elements(".news_item") |> # "news_item"クラス
    rvest::html_elements("a") |> # <a>タグ
    rvest::html_attr("href") # href の属性値
}
```

上のコードでは、わかりやすさを重視して、`html_elements()`では".news_item"と"a"を分けましたが、`html_elements(".news_item a")`と一度に指定することも可能です。

コード 15.25 (scrape-monthly-urls.R) : 新刊情報のページ一覧を取得

```
monthly_urls <- get_monthly_urls()
head(monthly_urls, 2)
## [1] "https://www.morikita.co.jp/news/category/newbook/101"
```

```
## [2] "https://www.morikita.co.jp/news/category/newbook/99"
# shell.exec(monthly_urls[2]) # 1つ目のページを開く
```

monthly_urls には、各月の新刊を紹介したページの URL が入っています。たとえば、2024 年 1 月の新刊 4 冊の概要は、以下の URL にあります。

<https://www.morikita.co.jp/news/category/newbook/99>

各書籍のタグの "book_title show_pc" クラスにある href 属性の値が、個別の紹介ページの URL です(図 15.8)。html_elements("a.book_title.show_pc")でタグとクラスで絞り込み、html_attr("href")で属性値である URL を取得します。

```
Elements
<div class="book_inner">
  <div class="book_img"> ... </div>
  <p> ... </p>
  <div class="book_text"> ... </div>
</div>
<p>
  <a href="https://www.morikita.co.jp/books/mid/005742" class="book_title show_pc" target="_blank">応用数学 (第2版) </a> = $0
</p>
<div class="book_wrapper"> ... </div>
<div class="book_wrapper"> ... </div>
</div>
```

図 15.8: 2024 年 1 月の新刊書籍

サーバへの負荷を考えて、Sys.sleep(5)で 5 秒間待機します。

コード 15.26 (scrape-book-urls-fun.R) : 新刊紹介の個別ページを取得する関数

```
get_new_book_urls <- function(url){
  Sys.sleep(5)
  url |>
    rvest::read_html() |>
    rvest::html_elements("a.book_title.show_pc") |>
    rvest::html_attr("href")
}
```

ここでは、2 ヶ月分の書籍の一覧を取り出します。

コード 15.27 (scrape-books-urls.R) : 新刊紹介の個別ページの取得

```
new_book_urls <-
  monthly_urls[1:2] |> # 全ての月のときは不要
  purrr::map(get_new_book_urls) |>
```

```

  unlist()
new_book_urls
## [1] "https://www.morikita.co.jp/books/mid/067731"
## [2] "https://www.morikita.co.jp/books/mid/008341"
## [3] "https://www.morikita.co.jp/books/mid/005742"
## [4] "https://www.morikita.co.jp/books/mid/071091"
## [5] "https://www.morikita.co.jp/books/mid/069201"

```

個別の紹介ページの URL が分かったので、それぞれのページから HTML の内容を読み込みます。ここでは、2024年3月発行の『固体力学入門』のデータ入手します。

コード 15.28 (scrape-books-detail-urls-fun.R) : 個別ページの内容を取得する関数

```

get_book_html <- function(url){
  Sys.sleep(5)
  rvest::read_html(url)
}

```

`new_book_urls` には、複数の URL がありますので、`map` を使って個別ページの内容を取得します。

コード 15.29 (scrape-books-detail-urls.R) : 個別ページの内容の取得

```

bk_details <- purrr::map(new_book_urls, get_book_html)
bk_details[[1]]
## {html_document}
## <html lang="ja">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
## [2] <body class="lower">\r\n\r\n  <div class="header_wrapper">\n    ...

```

個別の紹介ページから取り出した HTML の内容をもとに、新刊書籍の詳細な情報を取り出します。

関数 `detail2df()` では、取り出す要素を CSS セレクタで指定します。書籍の著者、タイトル、サブタイトル、目次などは、それぞれ `author`, `book_title`, `book_subtitle`, `index`, などのクラスです(図 15.9)。そのためこれらを CSS セレクタとして指定します。ただし、タイトルは紹介する書籍以外にも、ページの下部に掲載されている他の書籍も同じクラスを使っているため、`book_titles_wrapper` クラスも合わせて指定しています。また、ページ数、ISBN、発行年月はクラスだけでは取り出しにくううなので、CSS セレクタを開発者ツールでコピーします。

さらに各 CSS セレクタで該当部分を取り出して、文字列を取得する関数として `detail2elm_txt()` を定義します。ここでは、複数著者のときがあるので、カンマ区切りの文字列として結合します。

`detail2df()` では最後にデータフレームに変換します。

代数学入門

群・環・体の基礎とガロワ理論

紙版 電子版



早稲田大学教授 博 (数理科学) 永井保成 (著)

定価 ￥2,750

ページ 192

判型 菊

ISBN 978-4-627-08341-7

発行年月 2024.01

試し読み

The screenshot shows the browser's developer tools with the 'Elements' tab selected. It highlights a specific part of the DOM where the author information is located. The highlighted code block is as follows:

```
<div class="book_author" style="margin: 35.4048px 0px;">

This highlights the structure of the author information block, including the author's name and their title.


```

図 15.9: 2024 年 1 月の新刊書籍

コード 15.30 (scrape-books-details-fun.R) : 新刊の詳細情報を取得する関数

```
detail2df <- function(details){<br>
  selectors <- # 取得する要素の CSS セレクタ<br>
  c(".book_titles_wrapper > .book_title",<br>
    ".book_subtitle", ".index", ".author",<br>
    ".price.js-bookPage-bookPrice", ".content",<br>
    # 以下は、css セレクタをコピーしたもの<br>
    ".book_data.js-bookDataHeight > div:nth-child(1) > span.data",<br>
    ".book_data.js-bookDataHeight > div:nth-child(3) > span.data",<br>
    ".book_data.js-bookDataHeight > div:nth-child(4) > span.data")<br>
  names(selectors) <- # 名前を付ける<br>
  c("title", "subtitle", "toc", "author", "price", "content",<br>
    "page", "isbn", "yyyymm")<br>
  selectors |><br>
  purrr::map(\(x){ detail2elm_txt(details, x) }) |><br>
  tibble::as_tibble()<br>
}<br>
detail2elm_txt <- function(details, css){<br>
  details |><br>
  rvest::html_elements(css) |> # CSS セレクタ<br>
  rvest::html_text2() |> # 文字列のみ<br>
  paste0(collapse = ", ") # 複数著者への対応<br>
}
```

『固体力学入門』のデータをデータフレームに整理します。

コード 15.31 (scrape-books-details.R) : 新刊の詳細情報の取得

```
bk_details[[1]] |>
  detail2df()
## # A tibble: 1 × 9
##   title      subtitle  toc    author price content page  isbn  yyyyymm
##   <chr>       <chr>     <chr>  <chr>  <chr>  <chr>  <chr> <chr>  <chr>
## 1 数論幾何入門 モジュラ... "第1... 東京... 3,300 "《数... 224   978-... 2024...
```

これまでの関数を使用して新刊情報をエクセルにまとめるコードは次のとおりです。

get_book_html()は map()ではなく for ループ内で実行しています。これは、多くの情報を得るときのエラーなどへの対策で、途中まで得た情報を失わないようにするためです。map()を使うときは、possibly()などでエラー対策します(2.8 節を参照)。

コード 15.32 (scrape-books-all.R) : 新刊情報の取得(まとめ)

```
monthly_urls <- get_monthly_urls()
new_book_urls <-
  purrr::map(monthly_urls, get_new_book_urls) |>
  unlist()
bk_details <- list()
for(i in seq_along(new_book_urls)){
  bk_details[[i]] <-
    new_book_urls[i] |>
    get_book_html() |>
    detail2df()
}
bk_details <- dplyr::bind_rows(bk_details)
bk_details
## # A tibble: 5 × 9
##   title      subtitle  toc    author price content page  isbn  yyyyymm
##   <chr>       <chr>     <chr>  <chr>  <chr>  <chr>  <chr> <chr>  <chr>
## 1 数論幾何入... "モジ... "第1... 東京... 3,300 "《数... 224   978-... 2024...
## 2 加群とホモ... ""       "第1... 千葉... 3,960 "現代... 256   978-... 2024...
## 3 よくわかる... ""       "第1... 東京... 2,750 "いま... 160   978-... 2024...
## 4 よくわかる... ""       "Cha... 法政... 2,640 "電気... 176   978-... 2024...
## 5 固体力学入... ""       "第1... 名古... 3,520 "難解... 200   978-... 2024...
```

最後に、入手した情報をエクセルに保存します。

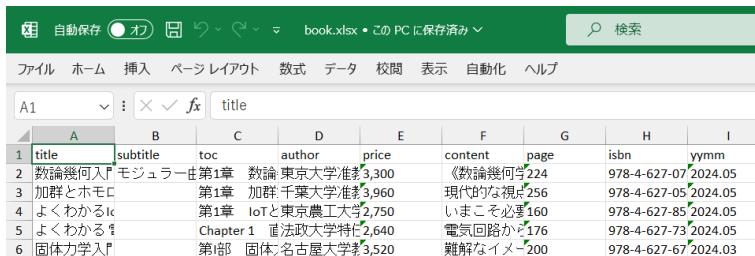
コード 15.33 (scrape-books-write-xlsx.R) : 新刊の詳細情報の取得

```

bk_xlsx <- fs::path_temp("book.xlsx")
openxlsx::write.xlsx(bk_details, bk_xlsx)
# shell.exec(bk_xlsx)

```

スクレイピングによって新刊書籍の一覧を入手することができました(図 15.10).



The screenshot shows a Microsoft Excel spreadsheet titled "book.xlsx". The table has columns labeled A through I, with headers: title, subtitle, toc, author, price, content, page, isbn, and yymm. The data consists of six rows of books:

	A	B	C	D	E	F	G	H	I
1	title	subtitle	toc	author	price	content	page	isbn	yymm
2	数論幾何入門モジュラー	第1章	数論 東京大学准教授 3,300		《数論幾何学》224		978-4-627-07	2024.05	
3	加群と木とモ	第1章	加群 千葉大学准教授 3,960		現代的な視点	256	978-4-627-05	2024.05	
4	よくわかるIoT	第1章	IoTと東京農工大学 2,750		いまこそ必要	160	978-4-627-85	2024.05	
5	よくわかる電気回路	Chapter 1	電気工学専門出版社 2,640		電気回路から	176	978-4-627-73	2024.05	
6	固体力学入門	第一部 固体	名古屋大学准教授 3,520		難解なイメージ	200	978-4-627-67	2024.03	

図 15.10: 入手した新刊書籍の一覧

エクセルのファイルは、とりあえずデータを入れただけの状態です。列幅の指定、オートフィルタの設定、罫線、特定文字列の強調で見やすくするには第 9 章を参考にしてください。

15.8 フォームへの入力

【報告】 form への入力・検索は必要な項目ですので、説明を少しだけ追加しました。

商品や店舗の検索などを自動化するときは、入力フォームに文字列を入力して送信する必要があります。

rvest では、html_form() でフォームの要素をまず取得します。

コード 15.34 (scrape-books-form.R) : フォーム要素の取得

```

url <- "https://www.morikita.co.jp/news/category/newbook"
form <-
  url |>
  rvest::read_html() |>
  rvest::html_form() |>
  `[[`(_, 1)
form
## <form> '<unnamed>' (GET https://www.morikita.co.jp/books/result)
##   <field> (hidden) condition: only
##   <field> (text) keywords:
##   <field> (button) :

```

フォームに文字列を入力するには、html_form_set() を使います。keywords の field に "テキストマイニング" という文字列を入力して送信するには、次のようにします。

コード 15.35 (scrape-books-form-set.R) : フォームへの文字列の入力

```

search <- rvest::html_form_set(form, keywords = "テキストマイニング")
search
## <form> '<unnamed>' (GET https://www.morikita.co.jp/books/result)
##  <field> (hidden) condition: only
##  <field> (text) keywords: テキストマイニング
##  <field> (button) :

```

検索文字列を入力したフォームを送信するには `html_form_submit()` を使用します。検索結果は、`read_html()` で読み込むことができます。その後は、これまでと同様に要素の取得などをしてください。

コード 15.36 (scrape-books-form-submit.R) : フォームの送信

```

response <- html_form_submit(search)
html <- rvest::read_html(response)
html_elements(html, "h3")
## {xml_nodeset (7)}
## [1] <h3>R によるテキストマイニング入門(第2版)</h3>
## [2] <h3>実践 R によるテキストマイニング</h3>
## [3] <h3>情報アクセス技術入門</h3>
## [4] <h3>データ解析の実務プロセス入門</h3>
## [5] <h3>Python ではじめる 情報検索プログラミング</h3>
## [6] <h3>事例+演習で学ぶ機械学習</h3>
## [7] <h3>Keras によるディープラーニング</h3>

```

`selenide` では、`s()` で取得した要素に対して、`elem_set_value()` を使うとフォームへの入力ができる、送信ボタンは `elem_click()` で要素のクリックができます。コード 15.35 と 15.36 と同様のことをするには、以下のようにします。

コード 15.37 (scrape-books-form-selenide.R) : selenide でのフォームの取得と送信

```

session <- selenide::selenide_session(session = "chromote", timeout = 10)
selenide::open_url(url)
# session$driver$view()                                # ブラウザを表示するとき
selenide::s(".searchWindow-input") |>                # フォームの入力位置の要素
  selenide::elem_set_value("テキストマイニング") # フォームへの入力
selenide::s(".btn") |>                               # 検索ボタンの要素
  elem_click()                                         # 検索ボタンをクリック
selenide::ss("h3")
## { selenide_elements (7) }
## [1] <h3>R によるテキストマイニング入門(第2版)</h3>
## [2] <h3>実践 R によるテキストマイニング</h3>
## (以下省略)

```

《Tips》 読者が定義する関数内で selenider の関数を使っていると要素の遅延取得のためにコードがうまく動作しないことがあるかもしれません。その場合は、 elem_cache()で要素のキャッシュを取得するとうまくいくことがあります。

15.9 雨雲の動きの画像を取得

スクレイピングでは、文字列としての情報を得るだけでなく、画像を得ることもできます。表示される画像をスクリーンショットとして保存すれば、ブラウザを起動しなくても目的とする画像を入手できます。

ここでは気象庁のページから雨雲の動きの画像を入手してアニメーション化して保存します。雨雲の動きの複数の時間帯の画像を入手するには、時間を進めるボタンをクリックしなければなりません。ボタンのクリックは selenider で簡単に実行でき、スクリーンショットも可能です。さらに、magick の関数を利用してアニメーション化します。自動化コードを一度書いてしまえば、気象庁のページを閲覧するよりも簡単に雨雲の動きを知ることができます。

雨雲の動きは詳細な画像を提供しています。必要な場所の雨雲の動きを知るためにには、その場所に対する URL を事前に取得する必要があります。以下の雨雲の動きのサイトにアクセスした後に、「現在地を取得」と拡大・縮小のボタンで得たい画像の範囲に設定します。

<https://www.jma.go.jp/bosai/nowc/>

画像の範囲と拡大率が決まれば、アドレスバーの URL をコピーします。ここでは、著者の職場である甲南女子大学を中心とした範囲で設定します。次のコードでは緯度(latitude)・経度(longitude)と拡大率(zoom)をもとに表示範囲を指定する方法を使います。普通はアドレスバーに表示される URL をそのまま指定すれば大丈夫です。

コード 15.38 (scrape-jma-open-url.R) : 雨雲の動きを開く

```
latitude <- "34.72"
longitude <- "135.30"
zoom <- "12"
url <-
  paste0("https://www.jma.go.jp/bosai/nowc/#",
         "lat:", latitude, "/lon:", longitude, "/zoom:", zoom,
         "/colordepth:normal/elements:hrpns&slmcs&slmcs_fcst")
session <- selenider_session(session = "chromote", timeout = 10)
open_url(url)
session$driver$view()
```

下の画像では白く塗りつぶしていますが、右側には広告が表示されます。より広い範囲の画像を得るために、広告非表示をクリックします。この部分は "div.mdc-button__label" クラスで特定できます(図 15.11)。

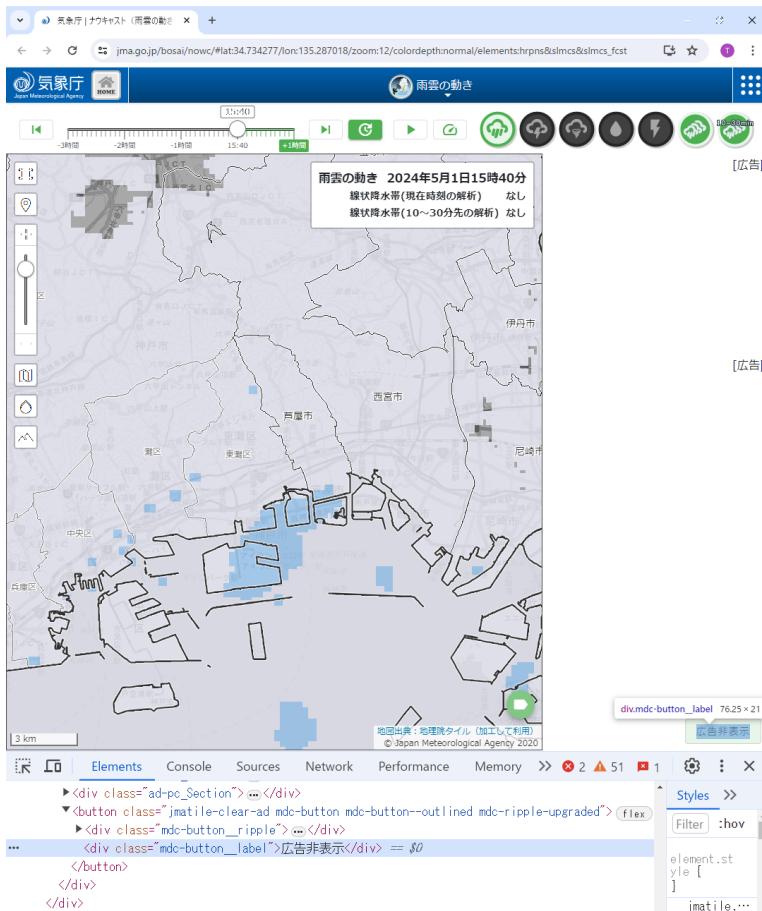


図 15.11: 広告非表示の要素

コード 15.39 (scrape-jma-elem-click-1.R) : 広告非表示をクリック

```
s("div.mdc-button__label") |>
  elem_click()
```

画像を保存するために、スクリーンショットを撮ります。ここでは、`session()$driver$view()` でブラウザの画面を出しながら作業します。なお、表示したブラウザ画面のスクリーンショットはイメージしやすいと思いますが、画面を表示していなくてもスクリーンショットは可能です。

コード 15.40 (scrape-jma-elem-click-2.R) : スクリーンショットを撮る

```
png <- fs::file_temp(ext = "png")
take_screenshot(png)
# shell.exec(png)
```

画面上のボタンを押すことで、時間を進めることができます。ボタンの要素は CSS セレクタでは取得しにくかったので、Xpath で取得して、それをクリックします(図 15.12)。

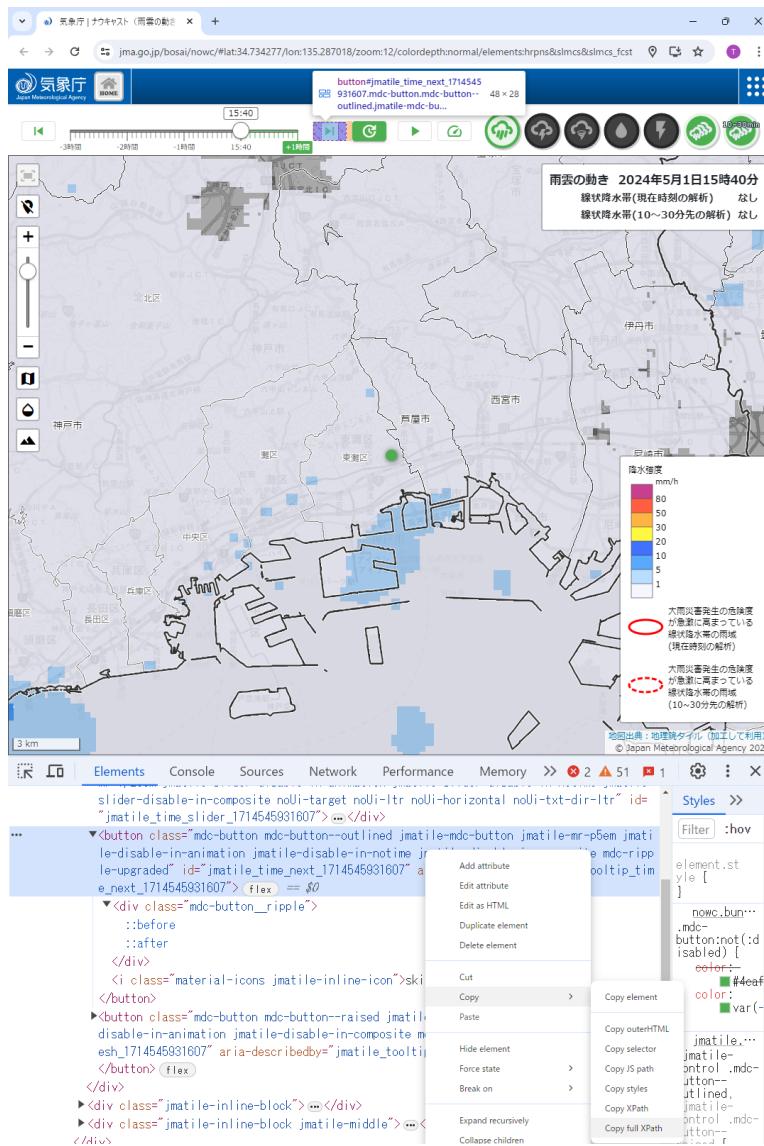


図 15.12: Full Xpath の取得

コード 15.41 (scrape-jma-elem-click-3.R) : 時間を進める

```
s(xpath = "/html/body/div[2]/div[1]/div[3]/div[1]/button[2]") |>
  elem_click()
```

スクリーンショットの撮影と時間の進行を繰り返せば、各時間帯の雨雲の動きの画像を取得できます。取得した各時間帯の画像を magick パッケージの関数でつなぎ合わせれば、アニメーションの画像が完成します。

コード 15.42 (scrape-jma-whole-game.R) : 雨雲の動きの動画の取得

```
scrape_jma <- function(url){
  session <- selenider::selenider_session(session = "chromote", timeout = 10)
  selenider::open_url(url)
  selenider::s("div.mdc-button__label") |>
    selenider::elem_click()
```

```

pngs <- list()
n <- 13
for(i in 1:n){
  no <- stringr::str_pad(i, width = 2, side = "left", pad = "0")
  png <- fs::path_temp(paste0(no, ".png"))
  pngs[[i]] <- selenider::take_screenshot(png)
  move_forward <- "/html/body/div[2]/div[1]/div[3]/div[1]/button[2]"
  if(i < n){
    selenider::s(xpath = move_forward) |>
      selenider::elem_click()
  }
}
rain_gif <- fs::path_home("desktop/rain.gif")
pngs |>
  unlist() |>
  magick::image_read() |>
  magick::image_animate(fps = 2) |>
  magick::image_write(rain_gif)
# shell.exec(rain_gif)
return(rain_gif)
}

```

雨雲の動きとは違って、今後の雨ではもう少し長い数時間程度の雨の降り方を予測しています。
初期ページを変更するだけで、同様のことができます。ただし、今後の雨は n が 13 だと最後まで動画が進まないので n <- 16 とすると良いでしょう。

逆引き一覧

- 一時
 - パス `fs::path_temp()` 1.2
 - ファイル `fs::path_file()` 1.2
- パッケージの
 - インストール `install.packages()` 1.1
 - 呼び出し `library()` 1.1
 - ディレクトリ取得 `fs::path_package()` 1.2
- ファイルの
 - 変更 `fs::file_move()` 1.6
 - 取得 `fs::dir_ls()` 1.4
 - 解凍 `unzip()` 5.6
- ディレクトリの
 - 一覧取得 `fs::dir_ls()` 1.4
 - ツリー表示 `fs::dir_tree()` 1.4

参考文献

Wickham, Hadley. 2012. グラフィックスのための *R* プログラミング. 丸善出版.

Wickham, Hadley, Mine Çetinkaya-Rundel, and Garrett Grolemund. 2024. *R* ではじめるデータサイエンス 第2版. Translated by 大橋真也. オライリー・ジャパン.

あとがき

技術書にとって、あとがきは必要なのだろうかと疑問を抱いたことはありませんか。書籍を執筆する機会があれば、あとがきを書いてみたいという憧れを昔から持っていました。30年ほど前に、大学の先輩で現在も研究仲間の人から、図鑑のまえがきを読むと面白いと教えてもらいました。半信半疑で読んでみると、本文では伝わらないような作成時の苦労や喜びが、ありありと伝わってきました。まえがきとあとがきは、役割は違えど、どちらもその本にまつわる大切な情報です。そこで、本書の執筆をはじめたきっかけについて、少しだけ書きたいと思います。

自分で書いた文章の保存場所を忘れてしまい、書き直した経験が何度もあります。探すのをやめたら紛失したものが見つかるのと同じで、完成してから最初に書いた文書が見つかることがしばしばあります。その時に文章を見直すと、細かな表現までほぼそのままということが多いです。自分が書いたものなので、内容を覚えていることも確かですが、それでも個人の思考回路は一定なのだと感じます。

プログラミングのコードでも同じことが言えます。書いたコードが見つからなかったときに、改めてコードを書き直したあとにコードが見つかることがあります。それを防ぐために、Rのコードの書き方をまとめ始めたのが、本書のきっかけです。

書き始めると思いのほか長くなってしまいました。執筆する中で、多くの有用なパッケージがあることを知りました。せっかくなのでそれも含めていると、さらに長くなってしまいました。それでも既存のパッケージでは達成できないことがあります。自分でもいくつかパッケージを作りました。ただし、著者が知らないだけで、さらに有用なパッケージがあるかもしれません。

Rは、統計分析やデータ可視化などを効率的にこなせますが、使いこなすためには、ある程度の知識と経験が必要です。本書では、Rの実践的な自動化テクニックを丁寧に解説していますので、Rによる自動化を簡単に始めることができます。その上でさらに改善を重ね、読者の環境にあわせたコードを書くと独自の自動化手法ができます。本書の内容に対して、さらなる改善案があればぜひ教えてください。

本書のテーマの自動化ですが、自動化といえばかっこいいです。でも、要は手抜きです。つまり、Rを使って面倒な作業を手抜きするための本です。読者のみなさんは「手抜き」にどのような印象をお持ちでしょうか。どちらかといえば、よくない印象を持っているかもしれません。私には魅力的すぎることばです。私は、執筆中に本書を「R友の本」と勝手に命名しました。最初は「R手抜き本」としていたのですが、「抜」という漢字から「手(手偏)を抜く」と「友」だけが残るためです(漢字の「抜」の成り立ちとは異なります)。本書が、みなさんの「友の本」として、少しでも役に立てば幸いです。