R にる作業の効率化・自動化 -パッケージ活用術-

Toshikazu Masumura

2023-05-09

はじめに

誰でもそうだろうが，面倒くさい仕事はしたくない．というか，したくないことが面倒くさいのだろう．ニワトリとタマゴの議論は別として，できることなら面倒な作業はしたくない．でも，しなければならないのなら自動化したい．もちろんすべての仕事を自動化できるわけでもないし，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る．

作業の自動化には，プログラミング言語を使うことが多い．自動化でよく使われる言語に Python がある．Python は比較的習得しやすい言語らしく，多くの人が使っている．自分自身も多少は Python を使えるものの，それよりも R の方が慣れている．できることなら (ほぼ) 全ての作業を R でやってしまいたい．そんなわけで，この文章では R を使った作業の自動化や効率化手法を紹介する．

基本的に独学でここまで来たので，我流や汚いコードが多くあると思われるがご容赦頂きたい．また，改善案をご教示いただければありがたい．

matutosi@gmail.com

対象とする読者

R とは

この文章では，好みとして R を使っている．R は，統計解析環境であるとともに，プログラミング言語である．プログラミング言語としては，やや特殊な文法をもっている．そのため，他の言語よりも好き嫌いが激しいと思われる．どうしても R を使うのが嫌であれば，Python での自動化について書いた書籍が多くあるので，それらを参考にして欲しい．

特徴

プログラミング言語としての R が文法的に特殊な点では，代入での <- 使用とパイプ (|>(R 4.1 以降) や%>%) の多用が挙げられる．

他の多くのプログラミング言語では，代入には = を使用する．R でも = を使えるが，<- を好んで使う人が多いと思われる．少なくとも私はそうしている．理由を問われても特には思いつかないが，慣れていることや，コードを見た時にすぐに R だとわかるぐらいだろうか．実用的には，= を入力するよりも手間がかかるし面倒なはずであるが，すでに手が慣れてしまっている．

パイプを最初に見たときには違和感を覚えたが，使い始めるとクセになる．クセになるだけではなく，同じ変数名を何度も使ったり，中間の変数名を考えなくて良い点で優れている．第 1 引数を省略できるため，入力の手間が少ない．パイプだけの恩恵でなく，tidyverse の利用も大きいが，コードが簡潔になって，コードの使い回しがし易い．パイプにはこのような多くの利点がある．

他にも :: がやたらと出てくることや，実行速度が遅いなどの欠点もそれなりにある．:: は `library()` でパッケージを呼び出せば，使わなくても良いことが多い．ただし，この文章で関数がどのパッケージのものかを示すために，不要な場合でも明示している場合が多い．実行速度が遅いのは，R だけでなくインタプリタの宿命である．Python もインタプリタでありそれ自体は実行速度は遅い．しかし，R でも Python

でも内部的では C や C++ を使っており，R や Python 自体で過度にループを使ったりしなければ実用的にはほとんど問題ない．

そもそも完璧なプログラミング言語などそ存在せず，それぞれに利点・欠点がある．それぞれの得意な分野でうまく使うことが重要である．とはいいながら，多くのプログラミング言語を習得するのは困難である．ちょっとだけでもこれまでにかじったことのある言語としては，FORTRAN，Perl，Ruby，C，C++，VBA，Java，Python，JavaScript，R などがある．それぞれなんとなく読むことはできるが，実際によく使うのは R だけである．JavaScript はその次に使っているが，頻度は非常に低い．Python は勉強中である．

R にはヘルプ・ドキュメントがしっかりしているというのも非常に良い．ヘルプは，「? 関数名」として R から直接呼び出すことがでる．関数の引数，返り値，使用例などが詳しく解説されていることが多い．ユーザーとしてはいちいちネットや書籍で調べなくても良いのが心強い．パッケージの開発者としては，既存のパッケージのドキュメントがしっかりしている．そのため，それに合わせるべく，しっかりとしたドキュメントを書かなければならないという心理的な圧力がある．ただ，ドキュメントをしっかり作っておかないと，開発者も関数の使い方を忘れてしまうことになりかねないため，結局は「他人のためならず」である．

1 点突破

プログラミング言語にはそれぞれ得意分野があることは確かだが，垣根を超えて使うことができる．例えば，R から Python を使うパッケージとして reticulate があり，Python から R を使うパッケージとして rpy2 がある．つまり，1 つのプログラミング言語でしか実行できないものはほとんどなく，使いたい言語を使って勉強したい言語を勉強すれば良い．

汎用的なプログラミング言語では，Python，C，C++，Java が，Web 関連では JavaScript が広く使われている．これらの言語に関連した多くのパッケージが，R の総本山である CRAN に登録されている．そのため，R を通してこれらの言語やそのパッケージが利用可能である．多くの言語を習得するのも良いが，習得にはかなりの時間が必要である．いっそのこと 1 つの言語をある程度極めて，そこから使うのは良い方法と言えるだろう．つまり一点突破の手法である．そこで，R のパッケージを使って，各種操作をすることを目的にこの文章を執筆した (している)．

もちろんだが，エラーが出たときの対処やより良い利用のためには，それぞれの言語のことを少しは知っておいた方が良い．場合によっては，R 以外の言語でコードを書く方が良い場合もある．私自身の例としては，編集距離を計算するコードを C で書いたことがある．編集距離は，植物の学名や和名の間違い候補を提案するための関数を作成するために必要であったが，R での実装では実行速度に問題があった．そのため，部分的に C で書いてそれをパッケージ Rcpp を利用して自作のパッケージに組み込んだ．正直なところは C で書いたというよりも，参考になるコードをネットから探して，多少アレンジしただけである．このような利用は実際の R のパッケージでも多く採用されており，R 本体や各種パッケージの多くの関数は C や C++ で実装されている．

結局のところ，表面的には R を使っていても，他の言語のお世話になっていることは多い．R だけでもかなりのことはできるし，他の言語であっても結局は同じようなことが言える．R に限らず自分の得意とする言語を深く勉強するとともに，他の言語も少し知っておくのが良いだろう．

R のインストール

R のインストール方法は，ネットでも多く掲載されている．ここでは，オプションの個人的な好みを強調しつつ説明する．

ダウンロード

OS に合わせたインストーラをダウンロードする．Windows の場合は，「Download R-4.x.x for Windows」(x はバージョンで異なる) である．

https://cran.r-project.org/bin/windows/base/

インストーラの起動

ダウンロードしたファイルをクリック．「…許可しますか?」に対して，「はい」を選択する．

- インストール中に使用する言語
  何でも大丈夫なので，好きなものを選ぶ．
- インストールの確認
  「次へ」をクリック．
- インストール先のフォルダ
  そのままで OK である．好みがあれば変更する．
- インストールするもの

とりあえず，すべてチェックしておくくと良い．Message translation は，R からのメッセージを日本語に翻訳するかどうか．チェックを入れないと，英語表示になる．

結論としては，とりあえずチェックを入れておき，必要に応じて英語で表示させるという方法が良いかもしれない．チェックを入れておくと，エラーメッセージは日本語で表示できる．「そら日本語のほうが良いやん」と思うかもしれない．よくわからないエラーメッセージが英語で表示されたら，わけがわからないからだ．ただ，プログラミングの世界では，英語でのエラーメッセージのほうが便利なことが結構ある．それは，エラーメッセージをそのままネットで検索するときである．日本語でのエラーメッセージではネット上の情報が限られる．一方，英語でのエラーメッセージで検索すると，原因や対処方法をかなりの確率で知ることができる．インストール後の設定変更は以下を参考にして欲しい．

```
# https://cell-innovation.nig.ac.jp/SurfWiki/R_errormes_lang.html
Sys.getenv("LANGUAGE") # 設定の確認
# 設定の変更方法
Sys.setenv(LANGUAGE="en") # 英語に変更
Sys.setenv(LANGUAGE="jp") # 日本語に変更
```

- オプションの選択

とりあえず「Yes」を選択する．以下のオプションを選択するかどうか．

- ウィンドウの表示方法 (MDI / SDI) の選択

個人的な好みは SDI だが，好みの問題なので正直どちらでも大丈夫である．. MDI(左) は大きな 1 つの Window の中に，コンソール (プログラムの入力部分)，グラフ，ヘルプなどが表示される．SDI(右) はコンソール，グラフ，ヘルプが別々の Window として表示される．どちらかといえば，自由度が高い．

- ヘルプの表示方法 (Plain text / HTML help) の選択

個人的な好みは Plain text だが，好みの問題で正直どちらでも構わない．Plain text はテキストファイルで表示されるシンプルなつくりである．HTML help はヘルプがブラウザ (GoogleChrome 等) で表示される．関連する関数などへのリンクが表示されるので，それらを参照するときは便利である．

- その後の設定

その他は，既定値 (そのまま) で OK である．

インストール完了

インストールが完了すると，アイコンがデスクトップに表示される．

アイコンをクリックすると，R が起動する．

パッケージのインストール

R 単体でも多くの機能があるものの，実際には各種パッケージを利用することが多い．パッケージのインストールには，R で簡単なコマンドを実行するだけである．

多くのパッケージが，R の総本山である CRAN に登録されている．https://cran.r-project.org/

CRAN に登録するには，それなりに厳しいチェックがある．ただし，私でも登録できていることが証明しているが，CRAN に登録されたからといってバグが無いわけではない．そのため R 本体もそうだが，R のパッケージ利用はあくまで自己責任である．

CRAN に登録されたパッケージの開発バージョンは，GitHub で公開されていることが多い．また，CRAN には登録されず GitHub のみで公開されているパッケージも存在する．

CRAN から

CRAN では R 本体だけでなく，各種パッケージが公開されている．

https://cran.r-project.org/web/packages/available_packages_by_name.html

CRAN に登録されたパッケージで名前がわかっていたら，以下のようにすればインストールできる．

```
  # ミラーサイト（ダウンロード元）の設定
options(repos = "https://cran.ism.ac.jp/")
  # 1 つの場合
install.packages("tidyverse")
  # 複数の場合
pkg <- c("xlsx", "magrittr", "devtools")
install.packages(pkg)
```

実行すると，ファイルをダウンロードし，成功 (あるいは失敗) したことが表示される．

GitHub から

たいていは CRAN に登録されているが，GitHub にしかないパッケージのときは以下のようにする．

```
install.packages("devtools")
devtools::install_github("matutosi/ecan")
```

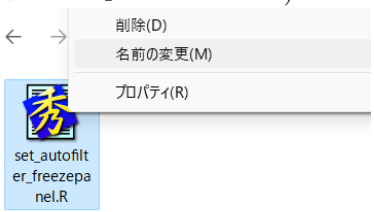
スクリプトの関連付け

ここでは，Windows でのファイルの関連付けについて説明する．

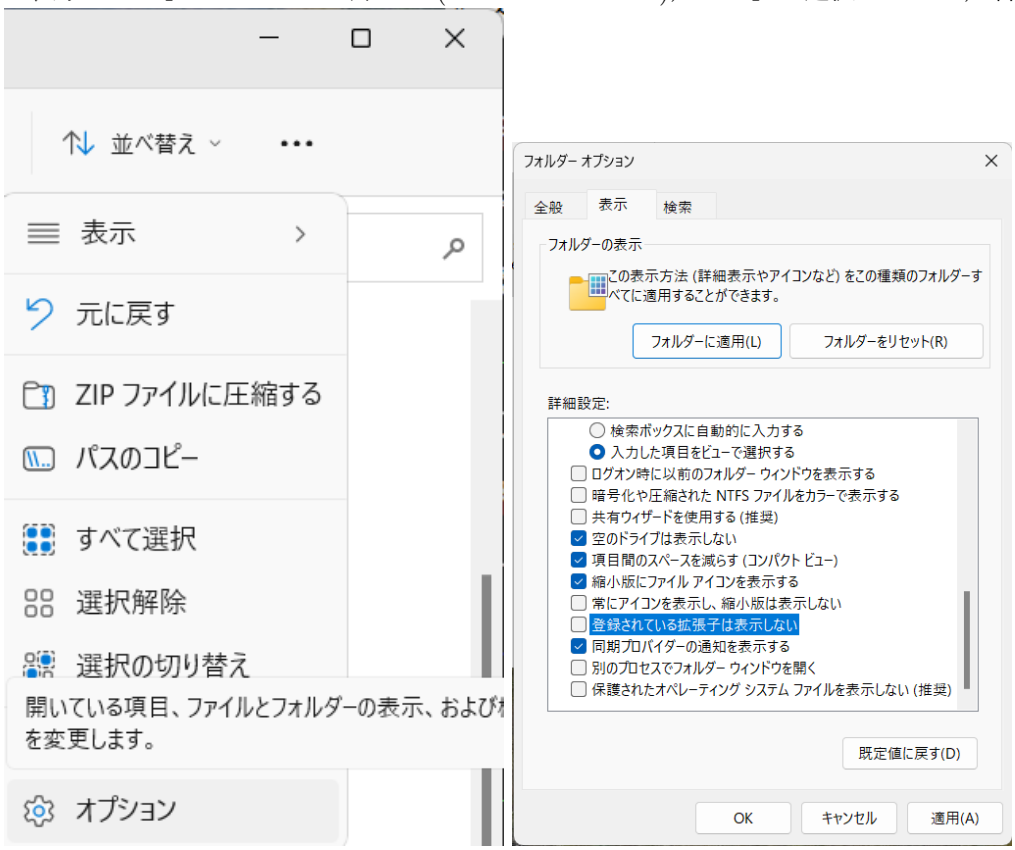
R のプログラムのファイルは拡張子「script.R」のように「R」という拡張子を付けて保存することが多い．拡張子「docx」をワードで，「xlsx」をエクセルで開くのと同様に，私は「R」をテキストエディタで開くするように設定している．その後，開いたファイルを R のコンソールに貼り付けて，プログラムを実行する．

このような使い方でもちろん良いのだが，コードの内容を変更しないのであれば，いちいち R を起動してコードを貼り付けるのは面倒臭い．ファイルをクリックするだけで，プログラムが実行されれば便利である．プログラムのファイルを R に関連付けることで，これが実現できる．

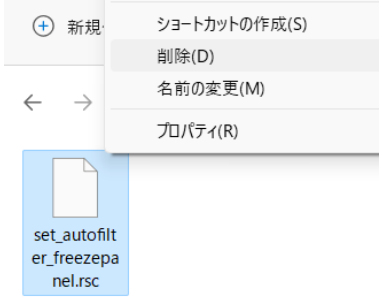
1. プログラムのファイル名を「R」から「scr」に変更する (「scr」は大文字小文字は関係なく，「Rsc」や「RSC」などでも OK)．



2. 拡張子が表示されていない場合は，エクスプローラの表示のオプションで，「登録されている拡張子は表示しない」のチェックを外して (チェックしないで)，「OK」を選択してから，名前を変更する．

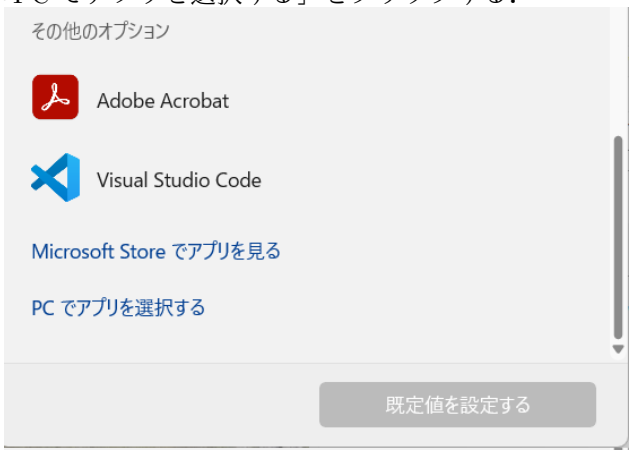


3. ファイルを右クリックして，「プロパティ」を選択する．

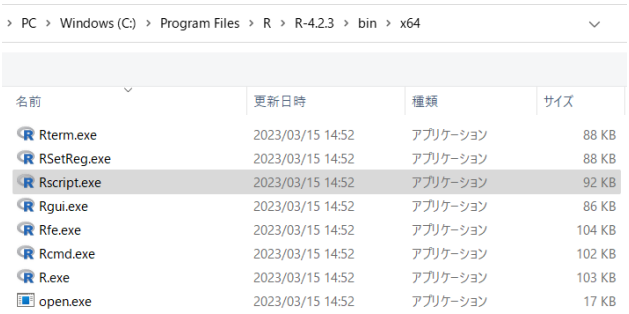


4.「全般」タブのやや上にあるプログラムの「変更」を選択する．

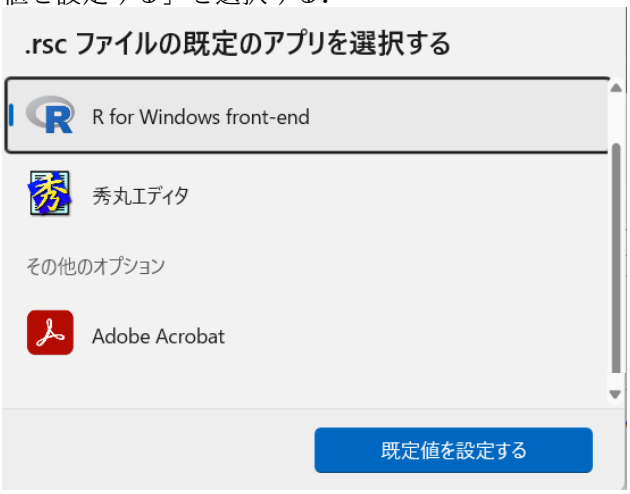
5.「PC でアプリを選択する」をクリックする．



6. ファイル選択画面で，R をインストールしたフォルダまで辿っていき（「c:\Program files\R\R-4.2.3\bin\x64」など），「Rscript.exe」を選択する．



7.「.rsc ファイルの既定のアプリを選択する」で「R for windows front-end」が表示されるので，「既定値を設定する」を選択する．



8. 全般タブのプログラムが「R for windows front-end」になっていることを確認して．「OK」を選択する．

9. ファイルのアイコンが R のアイコンになっていたら OK である．

ダブルクリックすると，ファイルの内容が実行される (はず)．

fs でファイル操作

Windows ならコマンドプロンプト (古い言い方なら，いわゆる dos 窓)，Mac なら Terminal，Linux ならシェルで，各種ファイル操作をコマンドラインで実行できる．もちろん，マウスを使った操作でも構わないが，多くのファイルで名前の変更やファイル名によるフォルダの振り分けなら，マウス操作よりもコマンドを使った操作が早いし確実である．なお，Windows では [Win] + [R] - [ファイル名を指定して実] -

[cmd] でコマンドプロンプトが，Mac では [Command] + [Space] - [Spotlight 検索] - [terminal] で Terminal が起動する．

コマンドプロンプトやバッチファイル (あるいはシェルスクリプト) などでの操作に慣れていれば，それが便利である．ただ，dos コマンドの変数の扱いは，慣れていないと結構難しい (慣れていても?)．そんなときは，R の関数 (`shell()` や `system()`) と，dos コマンドを駆使して，ファイル名を取得・名前の変更をすることができる．既に dos コマンドを書いていれば，`shell()` などを使うのは良い方法である．ただし，複数の OS を使う場合はコマンドが異なるためそれぞれでコマンドを覚えなければならず，また OS とコマンドとの対応で混乱することがある．OS ごとに異なるコマンドを覚えるよりも，R の関数で操作可能ならばどの OS でも同じように動作してくれて楽ができる．

R の base パッケージにはファイル操作のための関数が多くある．例えば，`list.files()` でファイル名一覧を取得でき，`file.rename()` でファイル名の変更ができる．しかし，base の関数には名前が分かりにくい点や引数の一貫性が無い点などの難点がある．これは，R が発展していく中で徐々に関数が追加されたことによるようだ．

fs パッケージでは，base の関数を整理するとともに，新たな有用な関数が追加されている．そのため，命名規則が一貫しており，ベクトル化した引数を受け付けるため，非常に使いやすい．

なお，fs，base，shell の詳細な比較が，以下の URL にあるので，参照してほしい．
https://cran.r-project.org/web/packages/fs/vignettes/function-comparisons.html

準備

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

```
library(fs)
```

shell，base パッケージ, fs パッケージ

a.pdf, b.pdf, …, j.pdf を 01.pdf, 02.pdf, …, 10.pdf のように 10 個のファイル名を変更したいとする．

shell を使う

shell なら，以下のようなコマンドだ．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かもしれないが，残念ながら私にはそのような技術がない．テキストファイルで書いてもそれほど時間がからないだろうが，ファイル数が多くなれば大変だ．

```
rename a.pdf 01.pdf
rename b.pdf 02.pdf
rename c.pdf 03.pdf
...
rename j.pdf 10.pdf
```

base パッケージ

R の標準の関数を使った例は次のとおりである．`sprintf()` は使い慣れていないと，指定方法が分かりにくい．

```
old <- paste0(letters[1:10], ".pdf")
new <- paste0(sprintf("%02.f", 1:10), ".pdf")
file.rename(old, new)
```

fs パッケージ

fs パッケージとともに，stringr を使った例を示す．ファイル操作をする際には，文字列の置換・検索などをすることが多いので，stringr が役立つ．stringr パッケージの関数は，`str_*` の名前になっているため，

覚えやすい．fs パッケージの関数は，パス操作は path_*，ディレクトリ操作は dir_*，ファイル操作は file_* という名前がついている．

```
library(stringr)
old <- str_c(letters[1:10], ".pdf")
new <- str_c(str_pad(1:10, width = 2, side = "left", pad = "0"), ".pdf")
file_move(old, new)
```

fs の関数

パス操作 (path_*)，ディレクトリ操作 (dir_*)，ファイル操作 (file_*) の関数に分けることができる．パス操作には，base や shell にはない機能が多くあって，使いやすい．拡張子を取り除く関数を自作したことがあるが，同じような関数があることを見つけたときには，下位機能の車輪を再発明してしまったと後悔した．しかも，fs のほうがしっかりしているはずだ．

fs，base，shell の比較は次の URL を参照して欲しい．

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

パス操作

パス操作では，stringr を駆使して自作しないといけないような関数が多くある．特に，パスからディレクトリ名，ファイル名，拡張子を抽出してくれる関数は便利だ．自作してもそれほど難しくはないが，自作した関数にはバグが入っている可能性がある．予想外のパスを指定した場合には，予想外の結果になることがあるだろう．そのような不具合を防ぐためにも，fs パッケージのパス関数を使うほうが良さそうである．

```
path("top_dir", "nested_dir", "file", ext = "ext")  # パス作成
path_temp(), path_temp("path")                      # 一時パス名の作成
path_expand("~/path")                               # "~" をユーザのホームディレクトリに変換したパス
path_dir("path")                                    # パスからディレクトリ名抽出
path_file("path")                                   # パスからファイル名抽出
path_ext("path")                                    # パスから拡張子抽出
path_ext_remove("path")                             # パスから拡張子を削除
path_home()                                         # ホームディレクトリ
path_package("pkgname", "dir", "file")              # パッケージのパス名
path_norm("path")                                   # 参照や ".." の削除
path_real("path")                                   # 実体パス（シンボリックリンクを実体パスに）
path_abs("path")                                    # 絶対パス
path_rel("path/foo", "path/bar")                    #  相対パス
path_common(c("path/foo", "path/bar", "path/baz"))  # パスの共通部分
path_ext_set("path", "new_ext")                     # 拡張子変更
path_sanitize("path")                               # 無効な文字を削除
path_join("path")                                   # 結合
path_split("path")                                  # 分割
```

ディレクトリ操作

shell や base でも同様の機能があるが，複数処理の dir_map() やツリー表示の dir_tree() は単純に嬉しい．

```
dir_ls("path")               # 一覧
dir_info("path")             # 情報
dir_copy("path", "new-path") # 複写
dir_create("path")           # 作成
dir_delete("path")           # 削除
dir_exists("path")           # 有無確認
```

```
dir_move() (see file_move)   # 移動
dir_map("path", fun)         # 複数処理
dir_tree("path")             # ツリー表示
```

ファイル操作

ファイル操作は shell や base とそれほど変わらない感じがする．

```
file_chmod("path", "mode")                # 権限変更
file_chown("path", "user_id", "group_id") # 所有者変更
file_copy("path", "new-path")             # 複写
file_create("new-path")                   # 作成
file_delete("path")                       # 削除
file_exists("path")                       # 有無確認
file_info("path")                         # 情報
file_move("path", "new-path")             # 移動
file_show("path")                         # 開く
file_touch()                              # アクセス時間等の変更
file_temp()                               # 一時ファイル名の作成
```

fs を使ったファイル操作例

ごく個人的なことだが，R のバージョンアップ時には Rconsole と RProfile.site を古いバージョンから複製して，カスタマイズした設定を引き継いでいる．バージョンアップをそれほど頻繁にしないのであれば，手作業でコピーしてもそれほど問題はない．普通の R ユーザなら常に最新版を使わなくても良い．ただ，パッケージの開発をしていると，開発中のパッケージが依存しているパッケージが最新版の R で開発されている旨の警告がでることが結構ある．ごく最近までは手作業でファイルをコピーしていたが，よく考えたらこういった作業は自動化するべきだと気づいた．そこで，fs パッケージを使ってファイルをコピーするスクリプトを作成した．

```
  # Script to copy Rconsole for updating R
  # R をバージョンアップしたときの Rconsole の複製スクリプト
  #   https://gist.github.com/matutosi/6dab3918402662f081be5c17cc7f9ce2
wd <-
  fs::path_package("base") %>%
  fs::path_split() %>%
  unlist() %>%
  .[-c((length(.) - 2):length(.))] %>%
  fs::path_join()
setwd(wd)
dir <- fs::dir_ls(type = "directory")
d_old <- dir[length(dir)-1]
d_new <- dir[length(dir)]
files <- c("Rconsole", "Rprofile.site")
f_old <- fs::path(d_old, "etc", files)
f_new <- fs::path(d_new, "etc")
fs::file_copy(f_old, f_new, overwrite = TRUE)
```

path_package() で base パッケージつまり R がインストールされているディレクトリを取得する．取得したディレクトリを文字列として分割して，その後ろから 2 つの”library”と”base”を削除して，wd(作業ディレクトリ) とする．複数のバージョンインストールされているので，dir_ls(type = "directory") で各バージョンのディレクトリを取得する．古いものから順番に入っているので，後ろから 2 番目と 1 番後ろのディレクトリをそれぞれ d_old と d_new とする．files にはコピーしたいファイル名を入れているので，他のファイルをコピーしたい場合は，ここを修正する．最後に新旧のファイル名を生成して，

`file_copy()` でファイルをコピーする．

このように，定期的あるいはバージョンアップなどに伴うファイルのコピーや移動はそれなりにあるように思う．そのような場合は，fs を活用して作業を自動化するとよいだろう．なお，fs で対応していない部分の文字列操作には，stringr を使うと便利である．

おまけ：GUI での作業フォルダの指定

GUI(Graphical User Interface) による作業フォルダを指定するには，tcltk パッケージを使うと良い．なお，`tkchooseDirectory()` で得たオブジェクトをそのまま `setwd()` で指定するとエラーになるので，`fs::path()` でパスに変換しておく．

```
getwd()
install.packages("tcltk")
library(tcltk)
wd <- tcltk::tkchooseDirectory()
setwd(fs::path(wd))
getwd()
```

magrittr でコードを簡潔に

パッケージ magrittr はちょっと変わったパッケージである．そもそも名前が変わっていて何と読んで良いのか分からない．公式ページには「magrittr (to be pronounced with a sophisticated french accent) 」と書かれている．フランス語は，大学の第 2 外国語で習ったが，すでに記憶の彼方に沈んでしまっている．主な関数がパイプ (%>%) である点も変わっている．

このように，ちょっと変わったパーッケージではあるものの，R を使う際には欠かせないパッケージといっても過言ではない．R のバージョン 4.1 以降では，base 機能のパイプとして |> が使えるようになったが，第 1 引数 (_) とその名前を明示しないといけないなど若干使いにくい．そこでここでは，コードを簡潔に書くための magrittr のパイプ (%>%) と関連した関数を紹介する．

準備

例によって，インストールとパッケージの呼び出しをしておく．

```
install.packages("magrittr")
```

```
library(tidyverse)
library(magrittr)
```

magrittr と tidyverse

tidyverse は，R でのデータ解析には欠かせないものになっている．R の起動時に tidyverse を読み込む人は多いだろう．tidyverse を読み込むと，その中のパッケージ (forcats，tibble，stringr，dplyr，tidyr，purrr) がインポートした%>% を使うことができる．そのため，私は%>% が tidyverse の独自のものだと勘違いをしていた．%>% はもとはパッケージ magrittr の関数である．

ただし，tidyverse と%>% の相性は非常によい．tidyverse の関数では，第 1 引数のオブジェクトが dplyr ではデータフレーム，stringr では文字列などのように，それぞれのパッケージで引数とするオブジェクトが統一されている．そのため，簡潔にコードを書くことができる．なお，パイプを手入力すると手間がかかるので，エディタにマクロやスクリプトを組んでショートカットを設定することをお勧めする．

%>% は，慣れるまでは何が便利なのか分からないが，慣れると欠かせなくなる．さらに使っていると，癖なってしまって無駄にパイプを繋ぐこともある．長過ぎるパイプは良くないのは当然であるものの，適度に使うと R でのプログラミングは非常に楽になる．

%>% でコードを簡潔に

最近の R では，パイプ (%>%) を多用したコードをよく目にする．はじめてみると，面食らって思考停止に陥ってしまうかも知れない．しかし，恐れることはない．以下では簡単な例を使って説明する．

```
a <- a %>% fun() # あるとき：慣れていないと変な感じ
a <- fun(a)      # ないとき：こっちのほうが分かりやすい
```

ある時とない時と比べると，ある時の方がコードが長くなっていて，何が便利なのかわからないだろう．著者も正直なところ，少し見たときには便利さが全くわからなかった．

```
  # ないとき：「a <- 」が何度も必要
a <- fun1(a)
a <- fun2(a)
a <- fun3(a)
  # 力技！：括弧が見づらい
a <- fun3(fun2(fun1(a)))
  # あるとき
a <- a %>%   # a に（以下の結果を）代入
  fun1() %>% # fun1 を実行
  fun2() %>% # fun2 を実行
  fun3() %>% # fun3 を実行
```

パイプを使うと，左側のオブジェクトを右側の関数の第 1 引数として使うため，引数が 1 つだけの場合は引数を書く必要がなくなる．このようにオブジェクト a に対して，fun1，fun2，fun3 と順に関数を適用するということは，しばしば出てくる．特に，dplyr で `select()`，`filter()`，`group_by()`，`summarise()` などを使う時はそうである．その時，代入先のオブジェクト名を 1 回ずつ新たに考えるのは，非常に面倒である．どこまでを同じ変数名として，どこで変えるかなど考えるのは手間だ．キーボードでの打ち込みが多いほど，オブジェクト名の重複や入力間違いが発生する可能性が高くなり，バグの温床である．これを避ける方法は入れ子状に書くことであるが，括弧の対応がよくわからず頭が混乱する．正直なところ，著者は時々このようなコードを書くことがある．しかし，他人から見たらこんな馬鹿げて見にくいコードは無い．

パイプを使うと，これらの問題が解決する．また，見た目にもわかりやすくコードの再利用もしやすい利点もある．見た目では，1 つの作業が 1 行にあるため，可読性が高く人間の思考回路にも近い．再利用するときに，処理対象のオブジェクトを引数として明記しなくても良いため，前後の文脈に左右されずに必要な部分のコードだけを複写・貼付できる．慣れるには，ほんの少しの時間がかかるかもしれないが，ぜひとも活用してほしい．

パイプの仲間

- %<>%
- %T>%(T パイプというらしい)
- %$%

これらの関数は，tidyverse には含まれていないため，使用するには magrittr を読み込む必要がある．%>% と似た機能を持つ．

%<>% でオブジェクトを代入

%<>% は，パイプを使って処理した内容を，最初のオブジェクトに再度代入するときに使う．ほんの少しだけ，コードを短くできる．

```
head(mpg) # 燃費データ
```

```
## # A tibble: 6 x 11
```

```
##   manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
##   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa~
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa~
## 3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa~
## 4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compa~
## 5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p     compa~
## 6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p     compa~
```

```
tmp <- mpg
tmp <-
  tmp %>%
  dplyr::filter(year==1999) %>%
  tidyr::separate(trans, into=c("trans1", "trans2", NA))

tmp <- mpg
tmp %<>%
  dplyr::filter(year==1999) %>%
  tidyr::separate(trans, into=c("trans1", "trans2", NA))
```

注意点としては，試行錯誤でコードを書いている途中は，あまり使わないほうが良いだろう．もとのオブジェクトが置き換わるので，処理結果が求めるものでないときに，もとに戻れなくなってしまうためだ．コードを短くできるのは 1 行で，可読性が特に高くなるというわけでもない．便利なことは便利で，私も一時期はよく使用していた．しかし，上記の理由もあって，最近はほとんど使用していない．

%T>% でコードを分岐

処理途中に分岐をして別の処理をさせたいときに使う．例えば，ちょっとだけ処理して，変数に保存するときに使う．imap と組み合わせて，保存する画像のファイル名を設定する時に使うと便利である．

%T>% は便利ではあるが，以下の点で注意が必要である．

- 分岐途中の結果をオブジェクトに代入するときには，<-ではなく，<<-を使う
- 明示的に . を使う
- 複数処理があれば，{と} で囲う
- 処理終了後に%>% が必要

%T>% を使うとコードの途中に，ちょっとだけ枝分かれしたコードを挿入できる．有用な機能ではあるが，トリッキーなコードになる可能性があるため，使いすぎには気をつけたい．

以下の例は，あまり実用的なものではないが，簡単な例として示す．

```
iris %>%
  tibble::tibble() %>%
  split(.$Species) %>%          # Species で分割
  purrr::map(print, n = 3) %T>% # それぞれ 3 行だけ表示
  {
    sp_path <<- paste0(names(.), ".txt") # ファイル名：種名.txt
  } %>%
  purrr::map(dplyr::select_if, is.numeric) %>% # 数値だけを選択
  purrr::map(dplyr::summarise_all, mean) %>%   # 平均を算出
  purrr::map2(.x = ., .y = sp_path, readr::write_csv) # 種ごとに csv を保存
```

T パイプは，T の文字の形が意味をなしており，左から来たデータを分岐させて右側と下側に流す役割をする．上の例では，分岐した途中でファイル名 sp_path を生成するために，(わざと) 分岐を入れている．

このコードで本来 {} は不要だが，入れておくと T パイプを使っている部分がわかりやすい．ファイル名を生成したあと (} の後ろ) には，上から流れてきたものがそのまま purrr::map() の第 1 引数として使われる．select_if() と dplyr::summarise_all() をしてから，最終的にファイルとして出力する．map2() は map() の引数を 2 つとるバージョンであり，第 1 引数に上から来たデータフレームを，第 2 引数にファイル名を使っている．

なお，最後の map2() を purrr::walk2 とすると種ごとの平均値は画面には出力されず，ファイルへの出力だけになる．さらに余談だが，上で「(わざと)」と書いたのは，T パイプを使わずに，最後のところを map2(.x = ., .y = paste0(names(.), ".txt"), readr::write_csv) とすれば同じ結果が得られるためである．

```
iris %>%
  tibble::tibble() %>%
  split(.$Species) %>%
  purrr::map(dplyr::select_if, is.numeric) %>%
  purrr::map(dplyr::summarise_all, mean) %>%
  map2(.x = ., .y = paste0(names(.), ".txt"), readr::write_csv)
```

%$% で $ のショートカット

%$% は，%>% と .$ の組み合わせのショートカットである．

```
mpg %>% .$manufacturer %>% head()
```

```
## [1] "audi" "audi" "audi" "audi" "audi" "audi"
```

```
mpg %$%   manufacturer %>% head()
```

```
## [1] "audi" "audi" "audi" "audi" "audi" "audi"
```

パッケージ開発ではパイプを使った場合の . が推奨されていない．パッケージ開発時に欠かせないチェック (R CMD check) では possible problem として Warning が出力される．そのため，そのままでは CRAN では受け付けてもらえない．Github でパッケージを公開するならそれでも問題はないが，Check で毎回 Warning が出力されるのは，心理的に嬉しくない．

そこで，DESCRIPTION で次のように%$% をインポートしておくと，.$ を使わなくても良い．

```
importFrom(magrittr,"%$%")
```

なお余談ではあるが，この場合は $ の代わりに [[と]] を使っても同じ結果が得られる．[と] ではデータフレームの 1 列をそのまま取り出すので，結果が異なる．

```
mpg %>% .$manufacturer      %>% head()
```

```
## [1] "audi" "audi" "audi" "audi" "audi" "audi"
```

```
mpg %>% .[["manufacturer"]] %>% head()
```

```
## [1] "audi" "audi" "audi" "audi" "audi" "audi"
```

```
mpg %>% .["manufacturer"]   %>% head()
```

```
## # A tibble: 6 x 1
##   manufacturer
##   <chr>
## 1 audi
## 2 audi
## 3 audi
## 4 audi
## 5 audi
```

```
## 6 audi
```

[[ ]] と [ ] は，それぞれ [[と [という関数であるため，以下のように書くことができる．この場合，第 1 引数がパイプの前から引き継がれるため，. を明示しなくてもよい．

```
mpg %>% `$`(manufacturer)    %>% head()
```

```
## [1] "audi" "audi" "audi" "audi" "audi" "audi"
```

```
mpg %>% `[[`("manufacturer") %>% head() # mpg %>% `[[`(., "manufacturer") と同じ
```

```
## [1] "audi" "audi" "audi" "audi" "audi" "audi"
```

```
mpg %>% `[`("manufacturer")  %>% head()
```

```
## # A tibble: 6 x 1
##   manufacturer
##   <chr>
## 1 audi
## 2 audi
## 3 audi
## 4 audi
## 5 audi
## 6 audi
```

パイプ以外の関数

magrittr にはパイプとともに使うと便利な関数も含まれている．例えば，パイプを使ったコードの中で列名を変更したいことがある．set_colnames() はデータフレームの列名を変更する時に便利だ．これを使わずに列名を変更しようとすると，ちょっとトリッキーな関数 colnames<-() を使うか，途中でコードを区切る必要がある．

```
hoge <- dplyr::select(mpg, 1:2)
cnames <- c("foo", "bar")
 # colnames(hoge) <- cnames      # 通常のコード
`colnames<-`(hoge, cnames)       # トリッキーなコード
hoge %>%
  magrittr::set_colnames(cnames)
```

おまけ：magrittr の不思議な関数たち

magrittr には以下のような関数もある．

```
not          # `!`, `n'est pas`(フランス語)も同じ
add          # `+`
subtract     # `-`
multiply_by  # `*`
divide_by    # `/`
and          # `&`
or           # `|`
equals       # `==`
```

?not とすれば，似たような不思議な関数の一覧と説明が出てくる．パイプと一緒に使うと add であれば，+ と同じ機能であるがパイプの中でつかたときに若干読みやすい気がする．

```
1:10 %>% `+`(100)
```

```
##  [1] 101 102 103 104 105 106 107 108 109 110
```

```
1:10 %>% add(100)
```

```
##  [1] 101 102 103 104 105 106 107 108 109 110
```

stringr で文字列操作

はじめに

stringr は stringi パッケージのラッパー関数群である．stringi は文字列操作の関数群で，文字コードの変換なども含む多様な関数を含んでいる．通常のユーザの文字列操作なら，stringr で大丈夫なことが多い．万が一，込み入った文字列操作が必要なときは，stringi の関数を探してみると良いかもしれない．

stringr には，

準備

```
install.packages("stringr")
```

```
library(tidyverse)
library(stringr)
```

stringr と base

base パッケージ

stringr パッケージ

stringr の関数

stringr の利点

少なくとも自分の経験では，stringr だけで操作が完結することはほとんどない．逆に，パッケージ開発をしていて stringr(や dplyr) を使わずに一日が終わることもあまりない．つまり，stringr はかなり便利で必要不可欠なツールである．もちろん，base パッケージの同様の関数を使っても機能上は問題ないことが多い．でも，引数の指定方法に一貫性があると，コードを綺麗に書くことができる．綺麗なコードは，汚いコードよりも書きやすいし，見た目も良いし，何よりもバグが入りにくい (入らないわけではない).

lubridate で日付・時刻を扱う

年月日や曜日を扱う場合，パッケージ lubridate を利用するのが便利である．lubridate は，tidyverse に含まれているパッケージの 1 つで，日付や時刻・時間データを扱う際には必須と言っても過言ではない．

準備

例によってパッケージのインストールと呼び出しだが，lubridate は tidyverse に含まれている．そのため，tidyverse を呼び出せばそれで OK である．日付の確認用としてカレンダーを最後に表示する．そのためのパッケージである calendR をインストールしておく．

```
install.packages("calendR")
```

```
library(tidyverse)
library(calendR)
```

1 月後・1 年後の同一日

例えば，1 月後や 1 年後の同一の日付を得たいとする．これは単純なようで実はややこしい問題を含んでいる．月には大の月・小の月があるし，年には閏年がある．そのため，同一日がないときがあるため，自分で関数を作成するにはこれらを考慮しなければならない．lubridate を活用すると簡単に計算できる．

1 年後の同一の日付を得るには `+ years(1)` とすれば良い．単純に 365 日加えるのとは結果が異なる．1 月後の場合には `months(1)` を使う

```
today() + years(0:4)
```

```
## [1] "2023-05-09" "2024-05-09" "2025-05-09" "2026-05-09" "2027-05-09"
```

```
today() + days(365 * 0:4)
```

```
## [1] "2023-05-09" "2024-05-08" "2025-05-08" "2026-05-08" "2027-05-08"
```

```
today() + months(0:4)
```

```
## [1] "2023-05-09" "2023-06-09" "2023-07-09" "2023-08-09" "2023-09-09"
```

```
today() + months(0:4)
```

```
## [1] "2023-05-09" "2023-06-09" "2023-07-09" "2023-08-09" "2023-09-09"
```

文字列から Date クラスへの変換

日本語の表記でよく出てくる年・月・日の順の日付表記は，関数 `ymd()` で Date クラスに変換できる．`ymd()` は，日付っぽい文字列を Date クラスにしてくれる．よく使うような以下の文字列は，普通に変換してくれる．ちなみに，日付の後ろに (火) のような曜日が入っていても問題ない (曜日は無視される)．

```
c("2023 年 4 月 10 日", "2023-4-10", "2023_4_10", "20230410", "2023/4/10") %>%
  ymd()
```

```
## [1] "2023-04-10" "2023-04-10" "2023-04-10" "2023-04-10" "2023-04-10"
```

```
c("2023 年 4 月 10 日 (月)", "2023-4-10(月)", "2023_4_10(月)", "20230410(月)", "2023/4/10(月)") %>%
  ymd()
```

```
## [1] "2023-04-10" "2023-04-10" "2023-04-10" "2023-04-10" "2023-04-10"
```

年が入っていない場合はうまくいかないので，年を追加する必要がある．

```
c("4 月 10 日", "4/10") %>%
  ymd()
```

```
## Warning: All formats failed to parse. No formats found.
```

```
## [1] NA NA
```

```
c("4 月 10 日", "4/10") %>%
  paste0("2023-", .) %>%
  ymd()
```

```
## [1] "2023-04-10" "2023-04-10"
```

ここでは日付を中心に扱うが，時刻の計算もうまくやってくれる．

```
ymd_hms("2023-5-1-12-23-34") %>%
  print() %>%
  `+`(minutes(40))
```

```
## [1] "2023-05-01 12:23:34 UTC"
```

```
## [1] "2023-05-01 13:03:34 UTC"
```

曜日を求める

日付をもとに wday() を用いて曜日を求めることができる．ただし，デフォルトでは日曜日を 1，月曜日を 2 のように日曜始まりの場合での曜日番号を示す．label = TRUE とすると，factor としての曜日を返してくれる．

```
x <- today()
wday(x) # week of the day
```

```
## [1] 3
```

```
wday(x, label = TRUE)
```

```
## [1] 火
## Levels: 日 < 月 < 火 < 水 < 木 < 金 < 土
```

活用例

日付固定 (同じ月日) あるいは位置固定 (m 月の第 n の w 曜日) のときでの一年後の年月日を求めることを考える．日付固定の場合は，既に説明したように非常に簡単に求められる．

```
x <- today()
x + years(1)
```

```
## [1] "2024-05-09"
```

位置固定の場合は，関数を作成する必要がある．年月日から第何の何曜日か分からなければ，位置を固定できない．曜日は wday() で求められるため，第何の曜日かを求める関数が必要だ．

```
mweek <- function(x){
  (mday(x) - 1) %/% 7 + 1
}
```

mday(x) で月の中で何日目か計算し (つまり day(x) と同じ)，そこから 1 日引いた数字を 7 で割る．7 で割ったときに第 1 曜日は 1 未満，第 2 曜日は 1 以上 2 未満であるため，7 で割ったときの整数部分に 1 を足す．これで第何の曜日かがわかる．

```
real   <- seq(as.POSIXct("2020-10-1"), as.POSIXct("2020-10-31"), by="day") %>% mweek()
expect <- rep(1:5, each=7)[1:31]
sum(real != expect, na.rm = TRUE)
```

```
## [1] 0
```

念のため計算した real と求めるべき expect が同じか確認する．real != expect，つまり real と expect 異なるときは TRUE になる．この合計が 0 であれば全部が同じなので，計算結果は正しいといえる．

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 の日付とする．

1 日から 7 日までは第 1 の，8 日から 14 日までは第 2 の曜日なので，base に「mweek(x) - 1) * 7」を足してやる．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する．ただし，差が負の場合は 7 から引いて正にする．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ある．入力が 1 日だけの場合は必要ないが，他の部分がベクトルに対応おり，せっかくなので複数の日付 (Date クラスのベクトル) を受け入れるようにした．

これで，一応出来上がった．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る．そこで，月がずれていないか確認して，ずれている場合は「NA」を返す．

```
same_pos_next_yr <- function(x){
  yr <- year(x)
```

```
  mn <- month(x)
  base <- ymd(paste0(yr + 1, "-", mn, "-", 1))
  diff <- wday(x) - wday(base)
  for(i in seq_along(diff)){
    if(diff[i] < 0){ diff[i] <- diff[i] + 7 }
  }
  same_pos <- base + (mweek(x) - 1) * 7 + diff
  for(i in seq_along(same_pos)){
    if(month(same_pos[i]) != mn[i]){
      same_pos[i] <- NA
      warning("No same position day with ", x[i], "!")
    }
  }
  return(same_pos)
}
```

実際の日付で確認してみる.

```
days   <- today() + (0:30)
days_n <- same_pos_next_yr(days)
```

```
## Warning in same_pos_next_yr(days): No same position day with 2023-05-29!
```

```
## Warning in same_pos_next_yr(days): No same position day with 2023-05-30!
```

```
days
```

```
##  [1] "2023-05-09" "2023-05-10" "2023-05-11" "2023-05-12" "2023-05-13"
##  [6] "2023-05-14" "2023-05-15" "2023-05-16" "2023-05-17" "2023-05-18"
## [11] "2023-05-19" "2023-05-20" "2023-05-21" "2023-05-22" "2023-05-23"
## [16] "2023-05-24" "2023-05-25" "2023-05-26" "2023-05-27" "2023-05-28"
## [21] "2023-05-29" "2023-05-30" "2023-05-31" "2023-06-01" "2023-06-02"
## [26] "2023-06-03" "2023-06-04" "2023-06-05" "2023-06-06" "2023-06-07"
## [31] "2023-06-08"
```

```
days_n
```

```
##  [1] "2024-05-14" "2024-05-08" "2024-05-09" "2024-05-10" "2024-05-11"
##  [6] "2024-05-12" "2024-05-20" "2024-05-21" "2024-05-15" "2024-05-16"
## [11] "2024-05-17" "2024-05-18" "2024-05-19" "2024-05-27" "2024-05-28"
## [16] "2024-05-22" "2024-05-23" "2024-05-24" "2024-05-25" "2024-05-26"
## [21] NA           NA           "2024-05-29" "2024-06-06" "2024-06-07"
## [26] "2024-06-01" "2024-06-02" "2024-06-03" "2024-06-04" "2024-06-05"
## [31] "2024-06-13"
```

計算できているはずだが，日付だけを見てもよくわからない.

各日付が第何の曜日かを確認してみる.

```
mweek(days)   # 第何の曜日か
```

```
##  [1] 2 2 2 2 2 2 3 3 3 3 3 3 3 4 4 4 4 4 4 4 5 5 5 1 1 1 1 1 1 1 2
```

```
mweek(days_n)
```

```
##  [1]  2  2  2  2  2  2  3  3  3  3  3  3  3  4  4  4  4  4  4  4 NA NA  5  1  1
## [26]  1  1  1  1  1  2
```

```
sum(mweek(days) != mweek(days_n), na.rm = TRUE)
```

```
## [1] 0
```

```
  # testthat::expect_equal(mweek(days), mweek(days_n))
```

最後の方でエラーが出ている．さらに，曜日も確認してみる．

```
wday(days, label = TRUE)   # 曜日
```

```
##  [1] 火 水 木 金 土 日 月 火 水 木 金 土 日 月 火 水 木 金 土 日 月 火 水 木 金
## [26] 土 日 月 火 水 木
## Levels: 日 < 月 < 火 < 水 < 木 < 金 < 土
```

```
wday(days_n, label = TRUE)
```

```
##  [1] 火   水   木   金   土   日   月   火   水   木   金   土   日   月   火
## [16] 水   木   金   土   日   <NA> <NA> 水   木   金   土   日   月   火   水
## [31] 木
## Levels: 日 < 月 < 火 < 水 < 木 < 金 < 土
```

```
sum(wday(days) != wday(days_n), na.rm = TRUE)
```

```
## [1] 0
```

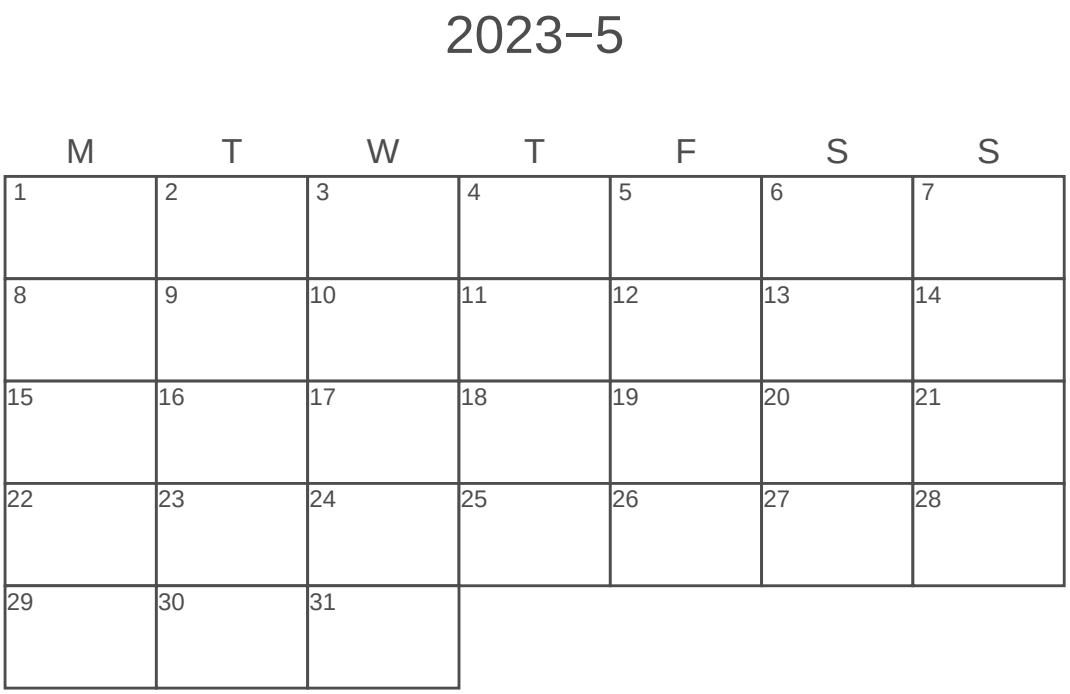
```
  # testthat::expect_equal(wday(days), wday(days_n))
```

こちらも，最後の方でエラーが出ている．このエラーは，第 5 週でずれるためでてくる部分である．ただし，閏年以外の 2 月だけは 4 週にピッタリ収まるので，エラーが出ないはずだ．

テストもだいたいあっていそうだが，分かりにくいため，カレンダーで表示してみる．

```
weeknames <-  c("M", "T", "W", "T", "F", "S", "S")
title_1 <- paste0(year(x)    , "-" ,month(x))
title_2 <- paste0(year(x) + 1, "-" ,month(x))

calendR::calendR(year(x)    , month(x), title = title_1, start = "M", weeknames = weeknames)
```




```
calendR::calendR(year(x) + 1, month(x), title = title_2, 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		

ggplot2 で楽に綺麗に作図

準備

```
library(tidyverse)
library(ggplot2)
```

R の作図環境の概要

R の作図環境として主なものは以下の 4 つがある．

- base graphics
- lattice
- grid
- ggplot2

base graphics は古典的な作図環境で長らく使われてきた．R が統計解析のシステムとして使われるようになった理由の 1 つとして強力な作図環境があり，まさしくこの base graphics システムがそれに当たる．すごく便利なものと当時は考えていた．ただし，base graphics は紙と鉛筆を使って作図していくようなものだと喩えられることがあるように，作図済みのものは修正できない．また，作図する関数によって引数の取り方が異なるなど，発展するなかで継ぎ接ぎだらけになってしまった．システムが急速に発展する中ではこのような状況はよくあり，途中から綺麗に整理し直すことは困難である．新しいシステムを作り直す方が楽であり現実的である．

そのような状況もあってか，lattice，それをもとにした grid，さらにはこの 2 つをベースにした ggpot2 が開発された．これら 3 つの作図環境のうち，最近では ggplot2 が最も使われているものである．ggplot2 で

は，Grammar of Graphics，つまり作図の文法という考え方が用いられており，洗練された作図が可能である．詳細は「ggplot2」(Hadley) を参照して欲しい．

ggplot2 とは

ggplot2 は，作図環境を提供するパッケージである．統一的なインターフェスを持っており，非常に使いやすい．散布図を作成したデータをもとにして，簡単に箱ひげ図などの他の形式の作図やグループ分けした作図も簡単である．

ggplot2 の利点

ggplot2 では，第 1 引数として tidy なデータフレームを受け取る．

- 1 つのデータから各種作図が可能
  ちょっとの変更で棒グラフ，散布図など各種の作図が可能
- 図が綺麗
- テーマの変更が簡単
- facet によるグループ分けが便利
- magrittr によるパイプとの相性が良い
  特にファイル名を設定するときの`%$%` や`%T%` など
- ggplot2 をサポートするパッケージも豊富
  凡例の自動的な位置決めや配置など ggpubr なども

ggplot2 の基本

aesthetics

geom_point() geom_bar() aes() colour group size

facet で簡単に分割作図

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のようだが，ggplot2 においては全体のデータを分割して表示させることを意味する．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 facet が使われている．例えば，iris のデータを全種でプロットするのではなく，種ごとに作図するのが facet である．ggplot2 ではこれが簡単にできて，しかも作図したものが見やすく，色々と指定できる点でも優れている．

for ループや subset，あるいは dplyr::filter を使っていたものが，一気にできて便利コードも簡単で見やすいコードの転用が簡単

group VS facet

ggsave

- png と PDF
  PDF で日本語文字が化ける場合は，png を使う
- 指定しないと，直前のプロット

文字化けへの対処 (windows)

-cario?

theme を少しだけ説明

- デフォルト
- theme_bw()

作図の自動化

例を示す.

- 入力：readr, readxl
  エクセルか csv でデータ入力
- 分析：dplyr, stringr
  filter(), summarise(), tally()
- 作図：ggplot2
  ggplot() geom_point() geom_jitter() geom_boxplot() ggsave()

参考書

- ggplot2
- ggplot2 のレシピ
- unwin GDA
- チートシート

shell

- R からシェルのコマンドを使う
  - ファイルの移動
  - PDF ファイルの結合
  - png から PDF へ変換

手作業でも良いが，ファイル数が多かったり，作業回数が多かったりするなら，自動化するのが便利である．例えば，ファイルの操作やちょっとした CUI アプリをコマンドでの動作を R でやってしまおうという邪道中の邪道である．上記の操作をする際は，Linux や Mac であれば shell スクリプトとして，Windows であればバッチファイルとしてコードを書くのが本来の方法である．しかし，shell スクリプトやバッチファイルのコマンドを体系的に勉強したことはない (その意味では R の勉強もかなり怪しい)．ウェブの情報をもとにしつつ，なんとなくコードを書いたことはある．とはいえ変数の使い方などは特によくわからないので，ちょっとした操作にも時間がかかりそう．そこで，慣れた R を使って雑多な操作をやっつけてしまおうと考えた．

以下のような操作を自動化する．・複数のフォルダに入った PDF ファイルを 1 つの PDF に結合・結合後のファイルを指定場所に移動・元ファイルを削除

なお，以下は基本的に windows での操作を前提としているが，Linux や Mac でも同じあるいは類似のコマンドで代用できる可能性が高い．日本語文字が入っていると，操作に若干手間がかかることが多い．

dos コマンド ls, dir ファイル，ディレクトリの一覧を取得 move, copy, remove, rename ファイルの移動，コピー，削除, リネーム cd ディレクトリの移動

R の関数 shell(), system() コマンドの実行 setwd() ワーキングディレクトリの設定ディレクトリ名にスペースや日本語が入っていて，cd コマンドがうまくいかないときは，こっちのほうが便利 paste0() 文字列の結合 stringr の関数 stringi の関数多くの関数は stringr にラッパーがあるが，文字コードの変換などは stringi の関数が必要日本語文字を使わなければ不要ファイル名の命名規則を決めておき，お世話にならない方が幸せ purrr::map() for loop の代わり # ファイル名を取得する関数など

その他ツール concatPDF PDF の結合など (win10 OK，win11 NG) # ConcatPDF /outfile Merged.pdf File1.pdf File2.pdf File3.pdf

pdftk PDF の結合など (win11 OK) pdftk File1.pdf File2.pdf File3.pdf cat output Merged.pdf

ImageMagick 画像変換など

準備

Python のスクリプト実行

```
wd <- "D:/matu/work/tmp"
setwd(wd)
system("c:/windows/py.exe pdf.py", intern = TRUE)
shell("pdf.py")
```

rvest でスクレイピング

スクレイピング

ここでのスクレイピングとは，ウェブスクレイピングの省略のことで，ウェブサイトにある情報を収集することである．ウェブサイトから植生調査データを収集することはほとんどないものの，関連データの収集は可能である．例えば，気象庁のページから気象データが収集可能である．もちろん，気象データは手動でも収集可能ではあるが，多大な手間と長い時間が必要である．研究に必要なデータを自動で取得できれば，手間と時間の節約が可能である．

そこで，本稿ではウェブでの情報収集の方法を紹介することを目的とする．世界の各地点の気象データをプロット情報収集には R のパッケージである rvest を用いる．rvest を用いて気象庁のページから世界の気象データを入手して，気候ダイアグラムを描画する．

R のパッケージ作成では，rvest を用いて作成した関数と収集したデータをまとめたパッケージの作成方法を紹介する．著者自身，他人のためにパッケージをつくることは考えておらず，基本的には自分の研究や作業のための関数をまとめることを目的としてパッケージをいくつか作成した．作成したら，ついでに他人にも使ってもらえれば嬉しいという程度である．

過去に作成した関数は，しばらくすると関数の引数や返り値がどのようなものであったのか忘れてしまいがちである．パッケージをつくる (特に CRAN に登録する) には，引数，返り値，使用例などをまとめる必要がある．きっちりまとめなくても良いのではあるが，決まった形式の方がむしろまとめやすい．また，RStudio と usethis, testthat, devtools などのパッケージを使ってパッケージ開発すると，各種チェックやテストが可能である．各種チェックやテストでたくさんのエラーを見ると，チェックやテストは正直なところ煩わしいと感じる．特に，パケージ開発に慣れていないと特にそうである．しかし，チェックやテストをすることで，関数の完成度を確実に高めることができるため，パッケージとしてまとめる利点である．

rvest と RSelenium

スクレイピングをするために使われる主な R のパッケージとしては，rvest と RSelenium がある．rvest は，静的なサイトを対象とするときに役立つ．つまり，URL を指定すれば対象のサイトのページが決まるときである．気象庁での気象データを提供しているページがこれに当たる．一方，RSelenium は動的なサイトを対象とするときに役立つ．例えば，テキストボックスへのデータ入力やプルダウンメニューの選択あるいはその後のマウス操作でページが遷移する場合である．このような動的なサイトでは，Selenium だけでなく，Javascript を部分的に用いるのも効果的である．なお，rvest でもユーザ名とパスワードを用いた一般的なログインは可能である．また，polite パッケージと組み合わせることである程度の動的なサイトのスクレイピングは可能である．

rvest のできること

- HTML の取得
- DOM の取得: id, class, tagName などを用いる
- table の取得
  - HTML 内の取得したいデータは table にあることが多いため，非常に便利そもそも，table でないデータを取得するのは非常に不便
- リンクの取得
  ページ遷移に使用する
  - stringr と組み合わせて使うと良い
  - 文字コードの変換には stringi を用いる
  - tidyverse や magrittr との合せ技が便利
- Form の入力・選択
  - radio ボタンはちょっと工夫が必要
    `-moranajp::html_radio_set()`
    無理やりな感じではあるが，同一名称の radio ボタンを全て同じ値に変更する
    本来なら，不要な radio ボタンのフォームを削除
    可能だが，インデクスがずれるので結構厄介
- polite パッケージとの連携
  使えば便利だが，ここでは説明せず

準備

例によって rvest をインストールする．curl と polite パッケージは少しだけ使うので予めインストールしておく．tidyverse は既にインストールしているはずだが，まだの場合はインストールする．

```
install.packages("rvest")
install.packages("curl")
install.packages("polite")
  # install.packages("moranajp")
  # install.packages("tidyverse") # 未インストールの場合
```

```
library(rvest)
library(tidyverse)
```

HTML の取得

スクレイピングによってデータを取得するには，取得したいページの URL を特定しなければならない．静的なページあるいは固定された URL であれば，ブラウザのアドレスバーにある URL をそのまま使えば良い．動的あるいは特定の規則に従った URL であれば，取得したいページの URL の規則性を知らなければならない．

ここでは，「日本のレッドデータ検索システム」から都道府県の RDB 指定状況とその地図情報の画像を入手することを考える．

http://jpnrdb.com/search.php?mode=spec

まずはブラウザでページにアクセス，手作業で検索，指定状況とその地図情報の画像を入手してみる．上記 URL で例として示されているニッコウキスゲをキーワード (種名) として入力すると，ページ遷移する．アドレスバーにはカタカナがそのまま表示されている．しかし，アドレスをコピーしてテキストエディタに貼り付けると文字化けしたようになる．これは URL エンコードによってコード変換された結果であ

るが安心して欲しい．rvest を使って HTML を取得するときには，日本語をそのまま使用することができる．上記の URL のうち「http://jpnrdb.com/search.php?mode= 」まではここで使用するページに共通する部分であるため，main としておく．検索したい種名は変更する部分で，とりあえず sp に入れておく．キーワード検索の命令 (php によるクエリ) と種名の文字列を結合し，さらに main と結合する．これで得た URL を `read_html()` に与えると，ページの HTML を得ることができる．

```
main <- "http://jpnrdb.com/search.php?mode="
sp <- "ニッコウキスゲ"
find_sp <- paste0("key&q=", sp)
html <-
  paste0(main, find_sp) %>%
  rvest::read_html()
html
```

```
## {html_document}
## <html>
## [1] <body>\n<em></em>\n\n<title>日本のレッドデータ検索システム</title>\n<meta http-equiv="co ...
```

必要な情報の取得

取得した HTML には必要な情報が含まれているが，そのままの状態では使い物にならない．また，文字列に変換して stringr を駆使すれば，情報を得ることはできるだろうが，多大な苦労が待っている．

```
as.character(html)
```

```
## [1] "<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.0 Transitional//EN\" \"http://www.w3.org/TR/REC-html4
```

```
rvest::html_table(html) %>%
  purrr::map(colnames) # とりあえず，colnames だけ表示
```

```
## [[1]]
## [1] "X1"
##
## [[2]]
## [1] "X1"
##
## [[3]]
## [1] "X1" "X2" "X3" "X4" "X5" "X6"
##
## [[4]]
##  [1] "X1"  "X2"  "X3"  "X4"  "X5"  "X6"  "X7"  "X8"  "X9"  "X10" "X11" "X12"
##
## [[5]]
## [1] "X1" "X2"
##
## [[6]]
## [1] "目録No "
## [2] "上位分類群 "
```

```
## [3] "科名 "
## [4] "和名 ／学名\r\n\t\t                                    "
## [5] "指定都道府県数 "
## [6] "環境省 "
##
## [[7]]
## [1] "X1" "X2"
##
## [[8]]
## [1] "X1" "X2"
```

```
rvest::html_table(html) %>%
  `[[`(6)
```

```
## # A tibble: 2 x 6
##   `目録No ` `上位分類群 ` `科名 ` 和名 ／学名\r\n\t\t~1   `指定都道府県数 `
##       <int> <chr>          <chr>    <chr>                                <int>
## 1      5266 単子葉類       ユリ     ゼンテイカ  Hemerocalli~                 6
## 2      5259 単子葉類       ユリ     ニッコウキスゲ  Hemeroc~                 3
## # i abbreviated name: 1: `和名 ／学名\r\n\t\t                                `
## # i 1 more variable: `環境省 ` <chr>
```

和名としてゼンテイカとニッコウキスゲの 2 つが示されている．ゼンテイカはニッコウキスゲの別名である．生物学的には同じものなので，本来は両方の情報を合わせる必要がある．別名かどうか判定するには生物の種について考える必要があり，この問題はかなり根深くてややこしいため，ここではあえて立ち入らない．単純に検索したものと同じ文字列の和名の目録 No を得ることを考える．

table の列名とその内容をもとにして目録 No を取得するには，dplyr の select と filter が便利だ．select は列名を指定する以外に列番号を指定できるので，それを使う．さらに，filter，stringr，stringi の関数の合せ技で sp と同じ文字列の no を取り出す．

途中でちょっと面倒な点があるので補足する.separate() の sep(区切り文字) として stringi::stri_unescape_unicode("
を指定している．これは，普通の半角スペースに見えるが，No-Break Space と言われる改行を防ぐ特殊なスペースである．これをそのままコードに入力しても良いが，どう見ても普通のスペースと見分けがつかない．後からコードを書く時に普通のスペースを使ってしまうと，区分しようとしてもうまくいかない．そこで，これは普通のスペースではないことを明示的に示した．また，str_detect() の引数で，paste0("^", sp, "$") としたのは，「ニッコウキスゲ」以外にマッチさせないためである．例えば，「ギンラン」を検索すると，「ギンラン」以外にも「エゾギンラン」と「ササバギンラン」も出てくる．この場合に正規表現の ^(行頭の意味)$(行末の意味) を使うことで，「ギンラン」にしかマッチさせない．

```
no <-
  rvest::html_table(html) %>%
  `[[`(6) %>%
  dplyr::select(no = 1, wamei = 4) %>%
  tidyr::separate(wamei, into = "wamei", sep = stringi::stri_unescape_unicode("\\u00a0"), extra = "drop
  dplyr::filter(stringr::str_detect(wamei, paste0("^", sp, "$"))) %>%
  `[[`("no")
```

URL の生成・データの取得

目録 No が取得できれば，完成したようなものである．ブラウザで表示した地図や指定状況の URL を生成する．“map&q=0605009”の詳細な意味はよくわからないが，“0605009”あたりは分類群を指定しているのだと考えられる．これに維管束植物 (コケなどを除くシダ植物と花の咲く植物) の中での目録 No を結合して，さらに main を結合すると URL の出来上がりだ．

```
show_sp <- paste0("map&q=0605009", no)
paste0(main, show_sp)
```

```
## [1] "http://jpnrdb.com/search.php?mode=map&q=06050095259"
```

```
html <-
  paste0(main, show_sp) %>%
  rvest::read_html()
```

生成した URL をブラウザで表示させると地図ページが表示される．一覧表の表示にしても URL は変更されないため，内部的に表示を変更させている可能性が高い．そこで，とりあえず HTML から table データを取得してみる．

```
html %>%
  rvest::html_table()
```

```
## [[1]]
## # A tibble: 1 x 1
##   X1
##   <lgl>
## 1 NA
##
## [[2]]
## # A tibble: 1 x 1
##   X1
##   <chr>
## 1 "ホーム | \r\n >>"
##
## [[3]]
## # A tibble: 1 x 6
##   X1    X2    X3    X4    X5    X6
##   <lgl> <lgl> <lgl> <lgl> <lgl> <lgl>
## 1 NA    NA    NA    NA    NA    NA
##
## [[4]]
## # A tibble: 49 x 5
##    `都道府県名 ` 和名  学名  RDBカテゴリ名 統一カテゴリ
##    <chr>          <chr> <chr> <chr>         <chr>
##  1 環境省RDB      -     -     -             ""
##  2 北海道         -     -     -             ""
##  3 青森県         -     -     -             ""
##  4 岩手県         -     -     -             ""
##  5 宮城県         -     -     -             ""
##  6 秋田県         -     -     -             ""
##  7 山形県         -     -     -             ""
##  8 福島県         -     -     -             ""
##  9 茨城県         -     -     -             ""
## 10 栃木県         -     -     -             ""
## # i 39 more rows
##
## [[5]]
## # A tibble: 8 x 13
##   ニッコウキスゲ学名：Hemerocallis dumo~1 ニッコウキスゲ ニッコウキスゲ `学名：`
##   <chr>                                  <chr>          <lgl>          <lgl>
## 1 "ニッコウキスゲ"                       "ニッコウキス~ NA             NA
## 2 "学名："                               "Hemerocallis~ NA             NA
## 3 "分類："                               "単子葉類 \t\~ NA             NA
## 4 "登録別名："                           ""             NA             NA
## 5 "環境省カテゴリ："                     "なし"         NA             NA
```

```
## 6 "都道府県のRDB指定状況："                ""            NA            NA
## 7 ""                                       ""            NA            NA
## 8 "*埼玉県・東京都・神奈川県では、季節~   ""            NA            NA
## # i abbreviated name:
## #   1: `ニッコウキスゲ学名：Hemerocallis dumortieri var. esculenta分類：\r\n\t\t\r\n\t\t 単子葉類 \t
## # i 9 more variables: `Hemerocallis dumortieri var. esculenta` <lgl>,
## #   `分類：` <lgl>,
## #   `単子葉類 \t\t\t\t \t\t\r\n\t\t\t\t\t\t\t \r\n\t\t\t\t\t\r\n\t\t\t\t ユリ科` <lgl>,
## #   `登録別名：` <lgl>, `` <lgl>, `環境省カテゴリ：` <lgl>, なし <lgl>,
## #   `都道府県のRDB指定状況：` <lgl>, `` <lgl>
##
## [[6]]
## # A tibble: 6 x 2
##   X1                    X2
##   <chr>                 <chr>
## 1 ニッコウキスゲ        "ニッコウキスゲ"
## 2 学名：                "Hemerocallis dumortieri var. esculenta"
## 3 分類：                "単子葉類 \t\t\t\t \t\t\r\n\t\t\t\t\t\t\t \r\n\t\t\t\t\t\~
## 4 登録別名：            ""
## 5 環境省カテゴリ：      "なし"
## 6 都道府県のRDB指定状況： ""
```

果たして，table の 4 番目に欲しいデータがあった．あとは，filter を使って指定されていない都道府県データを除去する．さらに必要に応じて，入手したデータを整形・変換・保存して欲しい．

```
html %>%
  rvest::html_table() %>%
  `[[`(4) %>%
  dplyr::filter(和名 != "-")
```

```
## # A tibble: 4 x 5
##   `都道府県名 `                          和名    学名    RDBカテゴリ名 統一カテゴリ
##   <chr>                                  <chr>   <chr>   <chr>         <chr>
## 1 "埼玉県\r\n        *"                  "ニッ~ "Heme~ "絶滅危惧 類~ ""
## 2 "滋賀県"                               "ニッ~ "Heme~ "分布上重要~  ""
## 3 "島根県"                               "ニッ~ "Heme~ "絶滅危惧 類~ ""
## 4 "*埼玉県・東京都・神奈川県では、季節~ "*埼~  "*埼~  "*埼玉県・~  "*埼玉県・~
```

地図画像の取得

指定状況の地図画像を取得するには，まずブラウザで画像の URL を得る必要がある．GoogleChrome で画像を右クリックして，「画像アドレスをコピー」を選択する．ニッコウキスゲの場合は，以下の URL を得ることができる．

http://jpnrdb.com/png/06/06050095259.png

指定状況の一覧表データの HTML にも (ほぼ) 同じものが含まれているはずである．rvest で目的とするファイルの URL を得るコードは以下のとおりである．

```
html %>%
  rvest::html_elements("img") %>%
  rvest::html_attr("src") %>%
  `[`(., stringr::str_detect(. , as.character(no)))
```

```
## [1] "./png/06/06050095259.png"
```

上のコード使用したように，rvest で便利な関数として html_elements() と html_attr() がある．それぞれ次のように id，class，tag，属性によって HTML から DOM を取得可能である．

- html_elements()
  - html_elements(“#id”)
  - html_elements(“.class”)
  - html_elements(“tag”)
- html_attr(“attribute”)

DOM とはドキュメントオブジェクトモデルのことで，HTML の各要素をオブジェクトとするモデルのことである．id，class，tag，属性を指定することで，効率的にオブジェクトを取り出すことができて，便利である．

id，class，tag，属性についての詳細は，HTML の解説などを別途参照していただきたい．簡単に説明をすると，id は HTML 内で一意に決定できるもので，日本のレッドデータ検索システムでは <id = “header” > などが使われている．class は，HTML 内で複数出てくることがあり，<class = “kind_list” > のように指定される．tag は，上記のやを含めたすべてのタグのことで，他にも

,

,

など多くの物がある．`html_table()` は `html_elements("table")` と同等であるが，table タグは入手したいデータを含むことが多いため個別の関数が作成されたのだろう．属性は tag の，「href = “index.html”」の部分で，`html_attr("href")` とすると，“index.html” を取り出すことができる．href が複数ある場合は，すべてを含むベクトルが返り値になる．

ただし実際には，上のようにブラウザでの右クリックか，以下の手順で実行するのが手っ取り早い．- ブラウザで地図ページを表示させる
- F12 を押して開発者ツールを開く
- 左上の　と　の結合したアイコンをクリック後に画像をクリック
- Elements のところに出てきた URL が求める URL
- タグを右クリックして [Copy] - [Copy element] や [Copy outerHTML] で内容をコピーできる

画像の URL がわかれば，ファイルをダウンロードして保存するだけだ．これは，curl パッケージの `curl_download()` で簡単にできる．引数として url には URL を，destfile にはダウンロード後のファイル名を指定する．ファイル名自体は指定が必要である．パスを指定しないと作業ディレクトリ (`getwd()` で取得可能) に保存されるが，作業ディレクトリ以外に保存したい場合は，相対パスや絶対パスを指定する．

```
  # wd <- "set_your_directory"
  # setwd(wd)
url_img <- "http://jpnrdb.com/png/06/06050095259.png"
curl::curl_download(url = url_img, destfile = paste0(sp, ".png"))
```

このようにしてスクレイピングが可能ではあるが,URL の生成規則は,変更されることがある.`read_html()` で HTML が取得できない場合は，URL が正しいか確認する必要がある．また，動的なサイトでは，id が固定ではない可能性がある．サイトの仕様変更によって，タグ，クラス，その他の構造が変更されることがある点も注意しなければならない．

綺麗な構造のサイトであっても，手作業が混入していることはある．例えば，括弧が正しく対応しているはずだと思っていても，開く側が”『“で閉じる側が”」“になっていることがあった．その場合に正規表現”『.+』“ではうまく鉤括弧内の文字列を取得できないことになる．

複数種への対応

前節のようにすれば，レッドデータへの指定状況とその地図データを得ることができる．1 種だけのデータ・画像の入手方法を紹介したが，複数種についてもこれを応用すれば可能である．その際には，for ループか，purrr::map を使うと良いだろう．

複数ページのデータを取得する場合は一般的には 5 秒程度の間隔を置く必要がある．ただし，サイトによってはそれ以上の間隔を求めているときがある．その内容はドメインのトップに置かれた「robots.txt」で確認できる．「http://jpnrdb.com/ robots.txt polite %60bow()‘でスクレイピングについて調べてみる．

```
polite::bow("http://jpnrdb.com/")
```

```
## <polite session> http://jpnrdb.com/
##     User-agent: polite R package
##     robots.txt: 1 rules are defined for 1 bots
##    Crawl delay: 5 sec
##   The path is scrapable for this user-agent
```

「Crawl delay: 5 sec」とあるため，5 秒間隔でスクレイピング可能であると思われる．これ未満の間隔でデータを頻繁に求めると，「攻撃」と見なされて接続できな状態になる可能性がある．さらに悪質なときには法的手段を取られることもありえるので，注意が必要である．ただし，http://jpnrdb.com/ robots.txt polite ．大量にデータを入手する必要がある場合は，あらかじめ管理者に連絡する方が無難である．

おまけ：webshot でウエブページを画像に変換

rvest でスクレイピングしたウエブページを画像として残しておきたいことがある．つまり，ウエブのスクリーンショットを保存したい場合だ．まさにこの文章を書いているときがそうだが，データを入手してそのサイトについて他人に説明したいときがあるだろう．そのようなときはパッケージ webshot が便利だ．

例によってまずはパッケージをインストールする．

```
install.packages("webshot")
```

webshot は内部で Phantomjs というブラウザを使っているので，webshot から Phantomjs をインストールするための関数を実行する．なお，Phantomjs はヘッドレスブラウザの 1 つである．ヘッドレスブラウザは，画面を描画しないブラウザのことである．つまり，画面上で HTML を表示させずにデータのやり取りだけをするもので，プログラムやスクレイピングでは重宝する．

```
webshot::install_phantomjs()
```

イントールに若干時間がかかるが，終わればあとは簡単だ．関数 webshot に URL と保存するファイル名を指定すれば，画像を作業フォルダに保存してくれる．

```
webshot::webshot("http://jpnrdb.com/search.php?mode=spec", "rvest_1.png")
webshot::webshot("http://jpnrdb.com/search.php?mode=key&q=ニッコウキスゲ", "rvest_2.png")
webshot::webshot("http://jpnrdb.com/search.php?mode=map&q=06050095259", "rvest_3.png")
```

なお，パッケージ magick を使うと保存した画像に対してトリミングなどの加工ができる．magick で画像編集

Rselenium

Selenium は，ブラウザを使って動的に巡回しつつ，スクレイピングをするのに適している．

Javascript や PHP などを使って，動的に作成されるサイトでは，URL だけではページを特定することはできない．そのため，rvest だけではデータを取得するのが困難である．

準備

- RSelenium: CRAN からインストール
- Selenium: 本家サイトからインストール

  - 注意: ver3.xxx をインストールする
    ver4.0 以上は RSelenium が対応していない (Python なら可)

- ChromeDriver
  - 注意: 自身の利用しているブラウザのドライバが必要 (バージョンも合致する必要あり)
    GoogleChrome は自動的に update されるので，バージョンをよく確認する通常は，安定版の最新版で大丈夫である
  - Selenium と同じフォルダに保存する

```
install.packages("RSelenium")
```

```
library(tidyverse)
library(RSelenium)
```

ブラウザの自動化

使い方

注意点

要素の取得

id がわかるとき document.getElementByID()

xpath document.selectQueryAll()[] 動的にサイトが作られているときには，変化する可能性があるので注意

使用されている JavaScript の関数がわかる script <- “” rem$excute(script)

例 BiSS の文字サイズの変更主命リストの列数の変更

スクレイピングの実行時には，適切な時間間隔を空ける - 通常は 5 秒以上を求めていることが多い

ページ遷移の命令を送信後，十分な間隔がないと HTML の要素を取得しきれていないことがある極端な場合，サーバーからの情報がほとんど何も送られていない，つまりページの内容がほとんど何もないことにある．この状況は，通常のマウス操作では何も表示されていないところをクリックするのと同じ状態である．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

動的なサイトの場合は，HTML の構成中の可能性もあるログイン等のページでも，遷移途中のことがある．

その他

- R からシェルのコマンドを使う
  - png から PDF へ変換
  - ファイルの移動
  - Seleniumu の起動・終了
  - MeCab や GINZA の実行

magick で画像編集

パッケージ magick を使うと，各種の画像編集ができる．magick は，画像編集ソフトである ImageMagick を使うためのパッケージである．ImageMagick は，png，gif，pdf などをはじめ多くの画像形式を扱うことができ，拡大・縮小，形式変換，回転，切り出し，色加工など多くの機能を備えている．ImageMagick 自体は，コマンドラインで使うことができるので，関数 `system()` を使って R から操作することもできる．でも，車輪の再発明は時間と労力の無駄なので，パッケージ magick を紹介する．

準備

例によってまずはパッケージをインストールする．svg 形式の画像を読み込む場合は，rsvg パッケージもインストールしておく．ウエブのスクリーンショットを使うので，webshot も必要だ．

```
install.packages("magick")
install.packages("rsvg")
install.packages("webshot")
```

```
library(tidyverse)
library(magick)
library(webshot)
```

使い方

ImageMagick 自体に多くの機能があり，magick でもその機能を利用できるため，1 つ 1 つを紹介するとキリがない．ここでは，基本的な機能として画像の読み込み・変換・保存を紹介するとともに，rvest でスクレイピングで使用した画像を作成するために，実際に操作したコードを説明する．

読み込み・表示・変換・保存

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()` をそれぞれ用いる．読み込んだ画像オブジェクトの情報を表示するには，`image_info()` を使う．表示だけなら，オブジェクトをそのまま入力しても同じである．

```
frink <- magick::image_read("https://jeroen.github.io/images/frink.png")
tiger <- magick::image_read_svg("http://jeroen.github.io/images/tiger.svg")
image_info(frink)
```

```
## # A tibble: 1 x 7
##   format width height colorspace matte filesize density
##   <chr>  <int>  <int> <chr>      <lgl>    <int> <chr>
## 1 PNG      220    445 sRGB       TRUE     73494 72x72
```

```
image_info(tiger)
```

```
## # A tibble: 1 x 7
##   format width height colorspace matte filesize density
##   <chr>  <int>  <int> <chr>      <lgl>    <int> <chr>
## 1 PNG      900    900 sRGB       TRUE         0 72x72
```

画像そのものを表示するには，`image_browse()` を用いる．OS で関連付けられているソフトが立ち上がって，画像を閲覧できる．

```
magick::image_browse(tiger)
```

`image_convert()` を用いると画像を png，jpeg，gif，pdf 形式に変換できる．画像形式は，`format` で指定する．また，`type = "grayscale"` とすると，白黒画像似できる．

```
magick::image_convert(tiger, format = "pdf") %>%
  magick::image_browse()
```

画像を保存するには，関数 `image_write()` を使う．この関数の引数にも `format` があり，ここで形式を変換することもできる．`image_write()` の `path` で拡張子を含むファイル名を指定しても，自動的にはその拡張子の形式としては保存されない．形式を変換したい場合は，`format` で形式をしていしなければならない．`format` を指定しなければ，もとの画像の形式のまま保存される．

```
magick::image_write(frink, path = "frink.pdf", format = "pdf")
magick::image_write(frink, path = "frink")
```

切り抜き

画像を切り抜きする前に，webshot を使って画像を取得する．

```
  # 画像の取得
urls <-
  list("http://jpnrdb.com/search.php?mode=spec",
       "http://jpnrdb.com/search.php?mode=key&q=ニッコウキスゲ",
       "http://jpnrdb.com/search.php?mode=map&q=06050095259")
pngs <- paste0("rvest_", 1:3, ".png")
purrr::map2(urls, pngs, webshot::webshot)
```

```
## [[1]]
##
## [[2]]
##
## [[3]]
```

image_crop() をそのまま使っても便利ではあるが，切り出しサイズ・位置の指定方法がどうも個人の性に合わない．そのためラッパー関数で，切り出したい左上の位置と右下の位置を指定できるようにする．切り出しの左上と右下の位置は，画像編集ソフトなどで別途確認する．

もし，切り出しの位置の細かな設定が面倒であれば，いくつかのパターンを作っておき，総当たり的に画像を切り出して，その中から求めるものを探す方法もある．その場合は，出力したファイルがどの設定によるものか分かるように，出力するファイル名に位置情報を付与しておくと良いだろう．

```
magick_crop <- function(image, 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)
  magick::image_crop(image, geometry)
}
```

今回は切り出し位置として 2 つ使う．1 つ目は，rvest_1.png と rvest_2.png で使うもので，2 つ目は，rvest_3.png で使うものである．それぞれの正しい設定だけを使ってコードを書いても良いが，面倒なのでとりあえず 2 つの位置を使って，画像を 3 つとも変換してしまう．変換後に実際に必要な画像だけを使えば良い．

ところで，magick パッケージの関数ベクトルに対応しているので，文字列のベクトルである pngs でもそのまま引数として使うことができる．ただし，2 つのベクトルを引数に使うことはできないため，以下では map2 を使っている．また，その直前で返り値をリストに変換するために as.list() を使っている．

```
crop_1 <-
  magick::image_read(pngs) %>%
  magick_crop(c(100, 0), c(890, 560)) %>%
  as.list() %>%
  purrr::map2_chr(., paste0("crop1_", pngs), magick::image_write)
crop_2 <-
  magick::image_read(pngs) %>%
  magick_crop(c(100, 0), c(890, 980)) %>%
  as.list() %>%
  purrr::map2_chr(., paste0("crop2_", pngs), magick::image_write)
crops <- c(crop_1[c(1,2)], crop_2[3])
```

ここでは，image_crop() を使用したが，上下左右から変化がない部分を除去する image_trim() という関数がある．余白部分 (色は白だけとは限らない) を単純に除去するだけでれば，位置を指定する必要のない

image_trim()‘の方が楽である．

サイズの変更

画像サイズを変更するには，image_scale() を使う．geometry = "200" で幅 200 ピクセルに，geometry = "x200" で高さ 200 ピクセルに変更できる．なお，縦横比を保ったままである．

```
resizes_crops <-
  magick::image_read(crops) %>%
  magick::image_scale(geometry = "600")
```

枠 (余白) の追加・註釈 (テキスト) の追加

画像に枠 (余白) と註釈 (テキスト) をつけるには，それぞれ image_border() と image_annotate() を使う．image_border() で geometry = "x40" は上下に，geometry = "40" は左右に 40 ピクセルの枠を追加する．image_annotate() は位置 location，角度 degrees，文字サイズ size，フォント font なども指定可能である．

```
boders_annotates <-
  resizes_crops %>%
  magick::image_border("white", geometry = "x40") %>%
  as.list() %>%
  purrr::map2(., urls, magick::image_annotate, size = 20)
```

その他の関数

magick には，他にも有用な関数がたくさんある．

```
image_append(image, stack = FALSE)   # 並べて表示，stack = TRUE で縦並べ
image_background(image, color)       # 背景の色付け
image_rotate(image, degrees)         # 回転
image_blur(image, radius, sigma)     # ぼかし処理．radius，sigma：ぼかしの大きさ
image_noise(image, noisetype)        # ノイズの追加
image_charcoal(image, radius, sigma) # 縁取り
image_oilpaint(image, radius)        # 油絵
image_edge(image, radius)            # エッジ
image_negate(image)                  # ネガ
image_morph(image, frames)           # 画像を指定した frames 間隔
image_animate(image, fps, loop)      # image_morph() 画像のアニメ化，fps：frames/秒，`loop = 1`で繰り返し
```

ウエブのスクリーンショットに URL を付加する関数

これまでの内容をもとに，ウエブのスクリーンショットに URL を付加する関数を作成する．主な流れは次のとおりである．

- 一時ファイル名を生成
- 一時ファイルに URL で指定したウエブのスクリーンショットを保存
- スクリーンショットを読み込み，一時ファイルの削除
- 余白のトリミング (オプション)
- URL を書き込む白色の枠を上下に追加

- URL の追加
- 下側の白色の枠の削除
- 形式の変換 (オプション)
- サイズの変更 (オプション)

```
#' Donwload screenshot image from web based on URL and add text annotation of URL on the top of the ima
#'
#' Needs
#'
#' @param url            A string of URL.
#' @param trim           A logical if trim image by magick::image_trim().
#' @param border_size    A string of border size. "x40": Set border on the top for annotation.
#'                       Will be passed like imagick::image_border(img, "white", geometry = border_size)
#' @param annotate_size A string of annnotation size.
#'                       Will be passed like magick::image_annotate(img, url, size = annotate_size)
#' @param format         A string of format of image.
#'                       Will be passed like magick::image_convert(img, format = format)
#' @param resize         A string for resize by magick::image_scale().
#'                       Passed like magick::image_scale(geometry = resize).
#' @return               An magick-image object.
#' @examples
#' # install.packages("magick")
#' # install.packages("webshot")
#' url <- "https://www.deepl.com/translator"
#' tmp <- web_screenshot(url, format = "png")
#' magick::image_browse(tmp)
#' magick::image_write("deepl.png")
#'
#' @export
web_screenshot <- function(url, trim = TRUE, border_size = "x40", annotate_size = 20, format = "png", r
  png <- paste0(fs::file_temp(), ".png")
  webshot::webshot(url, png)
  img <- magick::image_read(png)
  file_delete(path)
  if(trim){ img <- magick::image_trim(img) }
    # add top margin (removed bottom margin by image_crop)
  img <- magick::image_border(img, "white", geometry = border_size)
  crop_size <- stringr::str_split(border_size, pattern = "x", simplify = TRUE)[2]
  h <- magick::image_info(img)$height - as.numeric(crop_size)
  img <- magick::image_crop(img, geometry = paste0("x", h))
  img <- magick::image_convert(img, format = format)
    # add annotation
  img <- magick::image_annotate(img, url, size = annotate_size)
  if(resize != FALSE){ img <- magick::image_scale(geometry = risize) }
  return(img)
}
```

reticulate

R と Python のパッケージは，相互に移植されていることが多い．例えば，Python の logging(と R の futile.logger) をもとに R のパッケージ logger は開発されている．https://cran.r-project.org/web/packages

/logger/index.html また，R の ggplot2 や dplyr は Python にも移植されている．

ただし，どちらか片方でしか利用できなかったり，使用方法が難しいことがある．そんなとき，ちょっとだけ使うのであれば，R のパッケージ reticulate が便利である．もちろん，Python をちゃんと勉強するのも良いだろう．さらに，reticulate を使うと R と Python との変数のやり取りが簡単にできるので，本格的に Python を使うのにも良さそう．

準備

```
install.packages("reticulate")
```

```
library(tidyverse)
library(reticulate)
```

Python のインストール

Python でのモジュール (パッケージ) のインストール

Rstudio で python を書く (reticulate) https://qiita.com/Wa___a/items/42129e529cfb6c38e046

py_install() や conda_install() でパッケージがインストールできないとき - pip でパッケージをインストール
- pip でインストールできた python を reticulate::use_python() で指定

準備 - Python のインストール
- pdf2docx のインストール

```
pip install pdf2docx
```

実行

```
  # pdf2docx の読み込みでエラーになるとき
  #   reticulate::use_python() で python を指定
  #   pip で pdf2docx がインストールできた python を使う
library(reticulate)
  # reticulate::py_install("pdf2docx") エラー
  # https://anaconda.org/conda-forge/python-docx
  # reticulate::conda_install(channel = "conda-forge", packages = "python-docx")  # できたけど，pdf2docx
reticulate::use_python("C:/Python/Python39/python.exe")
reticulate::py_run_string("from pdf2docx import parse")
reticulate::py_run_string("pdf_file = 'D:/a.pdf'")
reticulate::py_run_string("docx_file = 'D:/a.docx'")
reticulate::py_run_string("parse(pdf_file, docx_file)")
```

Pytho と R との変数のやり取り

variable は変数名

```
  # RからPythonへ(Pythonで取り出し)
r.variable

  # PythonからRへ(Rで取り出し)
py$variable
```

DBI でデータ取得

データベースとの連携

リレーショナル・データベースと接続してデータを取得するためのパッケージには色々とある．

CRAN Task View: Databases with R には多くのパッケージが掲載されている．https://cran.r-project.org/web/views/Databases.html

どれを使っても良いが，よく使われているのは DBI のようだ．https://cran.r-project.org/web/packages/DBI/index.html

DBI でできること

- 各種データベースとの接続
- SQL によるデータ操作

SQL を使い慣れていれば，SQL で各種の操作をするのが良いだろう．一方，R でのデータフレームの操作に慣れていれば，取得したデータを R で操作するのが良い．つまり，データ取得だけに DBI を利用して，その後は dplyr や tidyverse の各種パッケージの関数を駆使してデータを処理する．さらに，その結果を図示したい場合は，ggplot2 を使うと良い．

準備

```
install.packages(c("DBI", "RSQLite"))
```

```
library(tidyverse)
library(DBI)
library(RSQLite)
```

```
  # 一時的データの準備
con <- dbConnect(RSQLite::SQLite(), dbname = ":memory:")
dbWriteTable(con, "mpg", mpg)
dbListTables(con)
```

```
## [1] "mpg"
```

使い方

```
  # SQL で選択・フィルタ
res <- dbSendQuery(con, "SELECT year, model, displ, cyl FROM mpg WHERE cyl = 4")
df <- dbFetch(res)
dbClearResult(res)
tibble::as_tibble(df)
```

```
## # A tibble: 81 x 4
##     year model      displ   cyl
##    <int> <chr>      <dbl> <int>
##  1  1999 a4           1.8     4
##  2  1999 a4           1.8     4
##  3  2008 a4           2       4
##  4  2008 a4           2       4
##  5  1999 a4 quattro   1.8     4
##  6  1999 a4 quattro   1.8     4
##  7  2008 a4 quattro   2       4
```

```
##  8  2008 a4 quattro   2       4
##  9  1999 malibu       2.4     4
## 10  2008 malibu       2.4     4
## # i 71 more rows
```

```
  # とりあえず全部取得してから，dplyr で選択・フィルタ
res <- dbSendQuery(con, "SELECT * FROM mpg")
df <- dbFetch(res)
dbClearResult(res)
df %>%
  tibble::as_tibble() %>%
  print() %>%
  dplyr::select(year, model, displ, cyl) %>%
  dplyr::filter(cyl == 4) %>%
  head()
```

```
## # A tibble: 234 x 11
##    manufacturer model      displ  year   cyl trans drv     cty   hwy fl    class
##    <chr>        <chr>      <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
##  1 audi         a4           1.8  1999     4 auto~ f        18    29 p     comp~
##  2 audi         a4           1.8  1999     4 manu~ f        21    29 p     comp~
##  3 audi         a4           2    2008     4 manu~ f        20    31 p     comp~
##  4 audi         a4           2    2008     4 auto~ f        21    30 p     comp~
##  5 audi         a4           2.8  1999     6 auto~ f        16    26 p     comp~
##  6 audi         a4           2.8  1999     6 manu~ f        18    26 p     comp~
##  7 audi         a4           3.1  2008     6 auto~ f        18    27 p     comp~
##  8 audi         a4 quattro   1.8  1999     4 manu~ 4        18    26 p     comp~
##  9 audi         a4 quattro   1.8  1999     4 auto~ 4        16    25 p     comp~
## 10 audi         a4 quattro   2    2008     4 manu~ 4        20    28 p     comp~
## # i 224 more rows
```

```
## # A tibble: 6 x 4
##    year model      displ   cyl
##   <int> <chr>      <dbl> <int>
## 1  1999 a4           1.8     4
## 2  1999 a4           1.8     4
## 3  2008 a4           2       4
## 4  2008 a4           2       4
## 5  1999 a4 quattro   1.8     4
## 6  1999 a4 quattro   1.8     4
```

SQL 使いの方は，「SQL ではじめるデータ分析―クエリで行う前処理、時系列解析、コホート分析、テキスト分析、異常検知」を参考にして SQL でデータ処理をするのも良いだろう．しかし，R 使いにとっては dplyr や ggplot2 を使って処理するほうが楽だと思われる．dplyr や ggplot2 を使ったデータ分析には，「R ではじめるデータサイエンス」が参考になる．https://r4ds.hadley.nz/

その他，DBI パッケージの詳細は以下を参照．

https://cran.r-project.org/web/packages/DBI/vignettes/DBI-1.html

xlsx でエクセル操作

xlsx パッケージを使うと，エクセルのファイルの読み込み・書き込みをはじめ，オートフィルタの設定やウィンドウ枠の固定などの各種操作が可能である．

準備

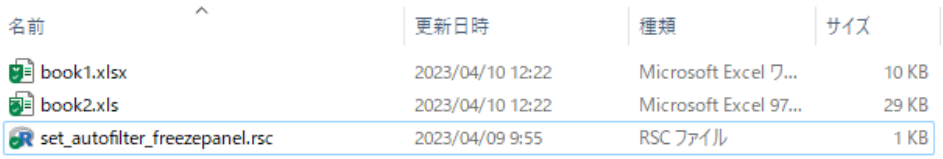
```
library(tidyverse)
```

オートフィルタの設定とウィンドウ枠の固定の自動化スクリプト

xlsx の使用例として，オートフィルタを設定して・ウィンドウ枠を固定する自動化スクリプトを作成した．

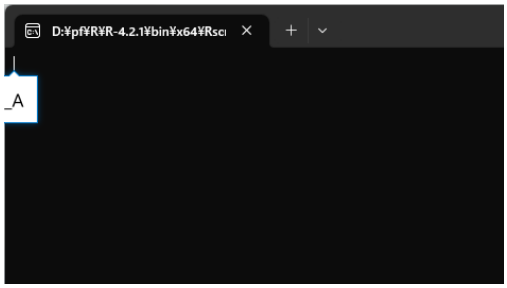
使用方法

- 準備：R のインストール
- 準備：set_autofilter_freezepanel.rsc をダウンロード (右クリックして「名前を付けてリンク先を保存」) して，任意のフォルダに保存．
- 準備：スクリプトの関連付けを参考にして，「.rsc」を「Rscript.exe」に関連付けする (Windows の場合)．Mac の場合は，Mac - 拡張子に関連付けられているアプリを変更する方法などを参考にしてほしい．
- set_autofilter_freezepanel.rsc と同じフォルダに，処理したいエクセルのファイルを保存．



- 実行前のエクセルのファイル

- set_autofilter_freezepanel.rsc をダブルクリックして実行すると，黒い画面が表示されてプログラムが実行される．



プログラムが自動的にエクセルのファイルの 1 行目の A 列から Z 列までにオートフィルタを設定し，1 行目と 1 列目でウィンドウ枠を固定する．複数ファイル・複数シートにも対応している．
なお，初回実行時は，xlsx パケージのダウンロードのため，少し時間がかかるかもしれない．2 回目以降はファイル数が多すぎなければ，一瞬で処理されるはず．
実行後のエクセルのファイル

スクリプトの内容説明

```
  # Package, 準備
if(! "xlsx" %in% installed.packages()[,1]){  # xlsx パッケージ有無の確認
  # パッケージが無い場合
  options(repos = "https://cran.ism.ac.jp/") # ミラーサイトの設定
  install.packages("xlsx")                   # パッケージのインストール
}

  # Functions, 関数
  #   註：xlsx パッケージの関数は返り値の代入がない
  #         副作用でシートなどを操作するため?
  #         参照型を使っているため?
  #   参考：通常の R の関数は，返り値の代入をすることが多いの

  # オートフィルタの設定
set_auto_filter <- function(sh){
  # A1 から Z1 までを設定
  #   もっと多くの列で設定したければ，"A1:Z1" のところを修正する
  xlsx::addAutoFilter(sh, "A1:Z1")
```

```
}

  # ウィンドウ枠の固定
set_freeze_panel <- function(sh){
  # 1 列目と 1 行目のウィンドウ枠を固定
  #   固定する場所の変更方法
  #     2 行目までを固定したい場合は，引数の 2 つ目と 4 つ目を，3 にする
  #     3 列目までを固定したい場合は，引数の 3 つ目と 5 つ目を，4 にする
  xlsx::createFreezePane(sh, 2, 2, 2, 2)
}

  # ワークブックごとで設定
set_af_fp <- function(file){
  wb <- xlsx::loadWorkbook(file)   # ワークブックの読み込み
  for(sh in xlsx::getSheets(wb)){  # シートの数だけ繰り返し
    set_auto_filter(sh)            #   オートフィルタの設定
    set_freeze_panel(sh)           #   ウィンドウ枠の固定
  }
  xlsx::saveWorkbook(wb, file)     # ワークブクの保存
}

  # Main, 本体
files <- list.files(pattern = "xls") # ".xls" と "xlsx" の一覧取得
for(file in files){                  # ファイルの数だけ繰り返し
  set_af_fp(file)                    # set_af_fp() の実行
}
```

その他の操作 (未作成)

qpdf で PDF 操作

準備

いつものように，まずパッケージをインストールする．qpdf は PDF 操作のためのパッケージである．著者作パッケージのである automater をインストールするが，これは CRAN には登録していない．いずれは CRAN に登録したいと思っているが，現段階では GitHub で公開している．そのため，install.packages() ではなく，devtools::install_github() を使ってインストールする．

```
install.packages("qpdf")
install.packages("devtools")
devtools::install_github("matutosi/automater")
```

```
library(tidyverse)
library(qpdf)
library(automater)
```

qpdf でできること

qpdf パッケージでは，PDF ファイルのページ分割・抽出・結合・回転・圧縮・重ね合わせが可能である．あくまでページ単位での操作で，PDF に含まれるテキスト自体の編集はできない．ページ単位での PDF 操作は，Adobe Acrobat でなくても CubePDF Utility や PDFtk を使えば可能である．PDFtk はコマンドラインでの操作も可能であるため，大量の操作をするには適している．とはいえ，R やそのパッケージで操作が自動化できればさらに便利である．

なお，PDF 関連の他のパッケージとしては pdftools がある．pdftools ではテキスト抽出，OCR(画像の文

字認識，内部で tesseract パッケージを使用)，PDF ファイルの分割・結合 (内部で qpdf パッケージの関数を使用)，画像ファイルへの変換などができる．また，Microsoft365R を使えば PDF をワードに変換できる．Microsoft365R

余談であるが，R のパッケージはそれぞれ独自コードを持つ部分がある一方で，他のパッケージの関数をインポートしているものや，ラッパー関数を用意しているものなどがある．例えば，automater ではそれ自体で有用な機能を持っているというよりは，他のパッケージを利用しやすくするためのラッパー関数の集合である．そのため，automater のコードをもとにさらに使いやすく改良可能であり，各自で試してほしい．

PDF の分割

PDF の分割は非常に簡単である．pdf_split() 関数に input 引数として分割するファイルを，output 引数として出力パスを指定すれば良い．パスワードが必要な場合は，引数 password を指定する．

```
  # wd <- "set_your_directory"
  # setwd(wd)
review <- curl::curl_download("https://www.jstage.jst.go.jp/article/vegsci/31/2/31_193/_pdf/-char/ja", 
split_pdf <- qpdf::pdf_split(review)
head(split_pdf)
```

ファイル名の文字列のベクトルが返り値なので，それをもとにしてファイル名を変更すると実用的な自動化ができるだろう．

ページを指定した抽出も可能で，pdf_subset() 関数を使用する．引数として pages を指定する以外は，pdf_split() と同じ使い方である．

```
  # 指定ページを抽出，create a new pdf with a subset of the input pages
pdf_subset(input, pages = 1, output = NULL, password = "")
```

以下の内容は，パッケージ automater の inst/rsc ディレクトリにある split_qpdf.rsc の内容である．拡張子.rsc を Rscript に関連付けすれば，split_qpdf.rsc と同じフォルダに保存した PDF ファイルを split_qpdf.rsc をクリックするだけで分割できる．拡張子の関連付けは，スクリプトの関連付けを参照して欲しい．

```
system.file("rsc/split_qpdf.rsc", package = "automater") %>%
  readLines() %>%
  paste0(collapse = "\n") %>%
  cat()
```

```
##   #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #
##   #
##   # See https://github.com/matutosi/automater/blob/main/vignettes/split_qpdf.md
##   #
##   #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #
##
##   # Prepare
## pkg <- "devtools"
## if(! pkg %in% installed.packages()[,1]){
##   install.packages(pkg, repo = "https://cran.ism.ac.jp/")
## }
##
## pkg <- "automater"
## ver <- utils::packageDescription(pkg, fields = "Version")
## if(utils::compareVersion(ver, "0.2.0") < 0){
##   devtools::install_github("matutosi/automater", upgrade = "never", force = TRUE)
## }
```

```
##
## automater::validate_package("qpdf")
## automater::validate_package("stringr")
##
##   # Run
## files <- list.files(pattern = "\\.pdf")
## for(file in files){
##   output <- qpdf::pdf_split(file)
##   n_page <- qpdf::pdf_length(file)
##   extra <- 0  # to avoid dupulicated file name, add extra degits
##   numbered <- automater::file_numbered(file, n_page, extra = extra)
##   while(automater::is_duplicated(files, numbered)){
##     extra <- extra + 1
##     numbered <- automater::file_numbered(file, n_page, extra = extra)
##   }
##   file.rename(output, numbered)
## }
##
## automater::message_to_continue()
```

```
  # system.file("rsc/split_qpdf.rsc", package = "automater") %>%
  #   readtext::readtext(verbosity = 0) %>%
  #   `[[`("text") %>%
  #   cat()
```

具体的な方法は次のとおりである．- split_qpdf.rsc をディレクトリーに保存する - 以下のコードで split_qpdf.rsc をコピー可能

```
file <- "split_qpdf"
path <- "c:/" # set your path
automater::set_rsc(file, path)
```

- 拡張子.rsc を Rscript.exe に関連付ける
- 分割したい PDF ファイルを split_qpdf.rsc と同じディレクトリにコピーする
- split_qpdf.rsc をクリックする
- 黒いウィンドウが開くので，しばらく待つ

split_qpdf.rsc を初めて実行するときは，パッケージのインストールに時間がかかることがある．出力ファイル名は以下のとおりである．- 入力：「original.pdf」(15 ページ) - 出力: “original_01.pdf”, “original_02.pdf”, …, “original_15.pdf”

PDF の結合

結合させる場合は，input 引数に結合させたいファイル名を指定する．それ以外は，pdf_split() と同様である．

```
  # 結合, join several pdf files into one
pdf_combine(input, output = NULL, password = "")
```

特定のディレクトリ内の PDF ファイルを 1 つの PDF ファイルとして結合することを自動化するスクリプトは以下のとおりである．

```
system.file("rsc/combine_qpdf.rsc", package = "automater")
```

```
## [1] "D:/pf/R/R-4.3.0/library/automater/rsc/combine_qpdf.rsc"
```

```
  readLines() %>%
  paste0(collapse = "\n") %>%
```

```
  cat()
```

- combine_qpdf.rsc をディレクトリに保存
  - 以下のコードで、combine_qpdf.rsc をコピー可能

```
file <- "combine_qpdf"
path <- "c:/" # set your path
automater::set_rsc(file, path)
```

- 拡張子.rsc を Rscript.exe に関連付ける

- 結合したい PDF ファイルを combine_qpdf.rsc と同じディレクトリにコピーする
  ファイルの結合順はファイル名の順序と同じ
- combine_qpdf.rsc をクリックする

- 黒いウィンドウが開くので，しばらく待つ
  初めて実行するときは，パッケージのインストールに時間がかかることがある．

- 結合したファイル名は以下のとおりである．
  出力: 「combined_“2020-11-27_12_00_00.pdf”」(結合した日付・時刻)

PDF ファイルの結合順はファイル名の順序と同じなので，`fs::dir_ls()` でファイル名一覧を取得し，`file_move()` で名前を変更する必要がある．さらに，このあたりも自動化するには，ユーザからの入力を受け取り，それをもとにファイル名の順序を決めれば良い．

```
  # wd <- "set_your_directory"
  # setwd(wd)
library(tidyverse)
input_files <- function(files){
  len <- length(files) %>% log10() %>% ceiling()
  no <- stringr::str_pad(seq(files), width = len, side = "left")
  prompt <-
    files %>%
    paste0(no, ": ", .) %>%
    paste0(collapse = "\n") %>%
    paste0("\n結合するファイルを番号で指定してください（カンマ区切り）．\n  例：3,1,2\n") %>%
    cat()
  input_order <-
    user_input(prompt) %>%
    stringr::str_split(",") %>%
    unlist() %>%
    as.numeric()
  files[input_order]
}
user_input <- function(prompt){
  if (interactive()) {
    return(readline(prompt))
  } else {
    cat(prompt)
    return(readLines("stdin", n=1))
  }
}

files <- fs::dir_ls(regexp = "\\.pdf")
input <- input_files(files)
input
```

```
automater::message_to_continue()
```

上のコードの input_files() では，ファイル名の一覧からファイル数を取り出し，ファイル番号を画面表示用に桁揃えしている．その後，ファイル番号とファイル名，さらにユーザへの註釈を結合して，プロンプトに表示するメッセージ文字列を生成する．メッセージを user_input() を用いて表示するとともに，ユーザからの入力を受け取る．入力された文字列を数値にして，ファイルの順序を決めている．

このコードの input を pdf_combine(input) として使えば，ユーザ入力をもとにして PDF ファイルを結合するスクリプトができる．入力する番号が数個であれば，これでも良いがもっと多くのファイルになった場合は現実的には使いにくい．多くのファイルを結合する場合は，以下のような方法で実装すると良いだろう．

- fs::dir_ls(regexp = "\\.pdf") でファイル名の一覧を入手
- 一覧をもとに 1 行ごとに 1 つのファイル名のテキストファイルとして保存
- テキストファイルをユーザが並び替える (R では並び替えが終わるまで待機)
- 並び替えが終われば，R で Enter(他のキーでも OK) を入力
- テキストファイルを読み込み，や combine_qpdf()(pdf_combine() でも OK) で PDF を結合

PDF の圧縮・最適化

pdf_compress() は圧縮や最適化 (Linealize) をしてくれる．最適化されていない PDF はファイルを全部読み込まないと表示できないのに対して，最適化された PDF は最後まで読み込みが完了しなくてもページ表示できる．ネット上にある重い PDF を表示させる場合に特に役立つ．使い方は次のとおりである．詳細な説明は不要だろう．

```
  # 圧縮，compress or linearize a pdf file
  # 最適化する場合は，linearize = TRUE
pdf_compress(input, output = NULL, linearize = FALSE, password = "")
```

PDF へのページ番号付加

pdf_overlay_stamp() を使うと，PDF ファイルに別の PDF ファイルを重ね合わせることができる．

```
pdf_overlay_stamp(input, stamp, output = NULL, password = "")
```

引数 input にはベースとなる PDF ファイルを，stamp には重ね合わせる PDF ファイルを指定する．stamp として「部外秘」「資料 1」などを記載した PDF ファイルをあらかじめ準備しておく．input の各ページに stamp の 1 ページ目が重ね合わせられる．

これだけでも十分便利な機能であるが，さらに便利に使いたい．例えば，ベースの PDF ファイルの各ページにページ番号を入力したい．ページ番号でなくて別の通し番号を使いたいときもあるだろう．例えば，学会の発表要旨集で左上に「A01」「A02」のような会場番号と通し番号を使うことが多い．さらに欲をだして，重ね合わせの開始・終了ページを指定するようにしたい．

これらの内容を実行するコードは次のとおりである．

```
#' Wrapper functions to overlay page numbers and others using package qpdf.
#'
#' pdf_overlay_stamps_each() overlay PDF for each page in pdf file.
#' validate_page() is a helper function for pdf_overlay_stamps_each()
#' to validate page consistency of among page no. of input, stamp, start and end.
#' pdf_overlay_page_num() and pdf_overlay_session_num() are wrapper functions to
#' overlay page no. and session no. for accademic congress or symposium etc.
```

```
#' pdf_overlay_page_num() can overlay up to 100 pages.
#'
#' Package qpdf <https://cran.r-project.org/web/packages/qpdf/index.html>
#' includes useful functions as shown bellow.
#' pdf_length(), pdf_split(), pdf_subset(), pdf_combine(),
#' pdf_compress(), pdf_rotate_pages(), pdf_overlay_stamp().
#'
#' @name pdf_overlay
#' @param input,stamp  A string of file name or path of pdf file.
#'                     input is a base pdf and stamp will be overlayed.
#'                     No. of pages in stamp PDF should be equal to or over no. of pages in input PDF.
#'                     Pages in stamp exceeding pages over input pages will be ignored.
#' @param start,end    An integer of start and end page to be stamped.
#'                     negative integer can be used for end, which means
#'                     number from the last page.
#' @param session      A string of session name. Can use "a", "b", or "p".
#'                     'session = "a"' uses 'pdf/00_sn_a.pdf' as stamp.
#'                     pdf directory include '00_sn_a.pdf', '00_sn_b.pdf', and '00_sn_p.pdf'
#'                     by default, which invlude 50 pages (eg., A01, A02, ..., A50) respectively.
#' @examples
#' \dontrun{
#' input <- system.file("pdf/00_sn_a.pdf", package = "automater")
#' pdf_overlay_page_num(input, start = 11, end = -3)
#' pdf_overlay_session_num(input, session = "b")
#' }
#'
#' @return  A string of output pdf file.
#' @export
pdf_overlay_stamps_each <- function(input, stamp, start = 1, end = NULL){
  len_input <- qpdf::pdf_length(input)
  len_stamp <- qpdf::pdf_length(stamp)
  if(is.null(end)){ end <- len_input }
  if(end < 0     ){ end <- len_input + end}
  validate_page(len_input, len_stamp, start, end)
  pages_inputs <- seq(to = len_input)
    # +1 means out of bounds, inputs[pages_pre]: NA
  pages_pre    <- if(start != 1      ){ seq(to = start - 1)                } else { len_input + 1 }
  pages_post   <- if(end != len_input){ seq(from = end  + 1, to = len_input)} else { len_input + 1 }
  pages_body   <- pages_inputs[-c(pages_pre, pages_post)]
  inputs <- qpdf::pdf_split(input)
  stamps <- qpdf::pdf_split(stamp)
  out <- list()
  for(i in seq_along(pages_body)){
    out[[i]] <- qpdf::pdf_overlay_stamp(inputs[pages_body[i]], stamps[i])
  }
  out <- stats::na.omit(c(inputs[pages_pre], unlist(out), inputs[pages_post]))
  outfile <- qpdf::pdf_combine(out, "out.pdf")
  file.remove(inputs)
  file.remove(stamps)
  file.remove(out[pages_body])
  return(outfile)
}

#' @rdname pdf_overlay
```

```
#' @export
validate_page <- function(len_input, len_stamp, start, end){
  if(end       < start)    { stop("end must be larger than start!") }
  if(len_input < start)    { stop("input pages must be larger than start!") }
  if(len_input < end  )    { stop("input pages must be larger than end!") }
  if(len_input > len_stamp){ stop("stamp pages must be equal to or bigger than input!") }
}

#' @rdname pdf_overlay
#' @export
pdf_overlay_page_num <- function(input, start = 1, end = NULL){
  stamp <- file.path(find.package("automater"), "pdf/00_page.pdf")
  pdf_overlay_stamps_each(input, stamp, start, end)
}

#' @rdname pdf_overlay
#' @export
pdf_overlay_session_num <- function(input, start = 1, end = NULL, session = "a"){
  stamp <- file.path(find.package("automater"), "pdf/00_sn_", session, ".pdf")
  pdf_overlay_stamps_each(input, stamp, start, end)
}
```

pdf_overlay_stamps_each() の引数には，input，stamp，start，end がある．input と stamp は qpdf の他の関数と同様の引数で，ファイルのパスを文字列で指定する．start と end は，input での重ね合わせ対象とするページ数の開始・終了ページで，整数で指定する．最後からのページ数とするには，end を負の数で指定する．

関数の主な構成は以下のとおりである．- input と stamp のページ数を取得　-start や end との整合性を validate_page() で確認　- 重ね合わせ対象よりも前 (pages_pre)，後ろ (pages_post)，重ね合わせの対象のページ (pages_body) を取得　-pdf_split() で input と stamp を 1 ページごとに分割 -pages_body の部分のみ，stamp の各ページを重ね合わせ　- combine_pdf() で使うためのファイル名を結合 (pages_pre，pages_body，pages_post‘)
- combine_pdf() でファイルの結合
- 使用後のファイルを削除
- 結合したファイル名を返す

なお，ページ番号と学会でのセッション番号を付与するためのラッパー関数として，それぞれ pdf_overlay_page_num() と pdf_overlay_session_num() がある．pdf_overlay_page_num() は，input の PDF だけ指定すれば全ページに番号を付加し，start と end が指定可能である．ただし，最大ページ数は 100 ページである．pdf_overlay_session_num() は，さらに session を指定して「A01」のような番号を左上に付加する．

pdf_overlay_page_num(input, start = 1, end = NULL) pdf_overlay_session_num(input, start = 1, end = NULL, session = “a”)

パッケージ automater の pdf フォルダには，ページ番号と学会でのセッション番号を付加するための PDF ファイル (すべて A4 版) がある．- 00_page.pdf - 00_sn_a.pdf，00_sn_b.pdf，00_sn_p.pdf ページ番号は，下部に「-1-」などの表記があり 100 ページ分からなる．セッション番号は，左上に「A01」(A 会場を想定)「B01」「P01」(ポスター会場を想定) などの表記があり，50 番までが入っている．それぞれの tex ソースファイルも保存されている．tex を使うのが難しければ，ワードなどで同様の書式のファイルを作成したものを PDF として保存すれば良い．

その他の関数

これまでで説明した以外に，qpdf には pdf_length() と pdf_rotate_pages() がある．pdf_length() は入力した PDF ファイルのページ数を返す．pdf_rotate_pages() は PDF ファイルのページを 90 度単位

で回転できる．angle で時計回りの角度を指定する．relative が TRUE のときは入力時点での角度からの相対的な角度で回転し，FALSE のときは angle = 0 のときは縦長で angle = 90 のときは横長になる．

```
pdf_length(input, password = "")
pdf_rotate_pages(input, pages, angle = 90, relative = FALSE, output = NULL, password = "")
```

pdftools でテキストの取り出し

pdftools パッケージでは，テキスト抽出，OCR(画像の文字認識)，PDF ファイルの分割・結合，画像ファイルへの変換などができる．このうち，OCR では tesseract パッケージを，PDF の分割・結合では qpdf パッケージの関数を使っており，直接それぞれのパッケージを使うのと基本的には同じ，画像ファイルへの変換は magick パッケージで可能である．そのため，ここでは他のパッケージでは実装していないテキスト抽出を説明する．なお，テキスト抽出には poppler を使っている．Windows 版の pdftools パッケージでは poppler が含まれているのでパッケージのインストールだけで使用可能である．Mac や Linux では，poppler を別途インストールしなければならない．poppler のインストール方法は以下を参考にして欲しい．

https://docs.ropensci.org/pdftools/

まずは，パッケージのインストールと呼び出しを実行する．

```
install.packages("pdftools")
```

```
library(pdftools)
```

関数 pdf_text() に PDF ファイルのパスを指定すれば，テキストを抽出した結果が得られる．1 ページごとの内容が文字列のベクトルになっている．

```
  # https://docs.ropensci.org/pdftools/
url <- "http://arxiv.org/pdf/1403.2805.pdf"
destfile <- "1403.2805.pdf"
curl::curl_download(url, destfile)
txt <- pdftools::pdf_text(destfile)
tibble::as_tibble(txt)
```

```
## # A tibble: 29 x 1
##    value
##    <chr>
##  1 "                                                The jsonlite Package: A Pract~
##  2 "JSON with R. We refer to Nolan and Temple Lang (2014) for a comprehensive i~
##  3 "homogenous. And indeed, some implementations will now return a list instead~
##  4 "The alternative to class-based method dispatch is to use type-based encodin~
##  5 "2       Converting between JSON and R classes\n\nThis section lists example~
##  6 "encoding. However, the problem with encoding missing values as strings is t~
##  7 "limitations as text based formats such as CSV.\n\n\n2.1.3      Special case~
##  8 "is assuming an array, the application will likely break. Any consumer or cl~
##  9 "We expect this representation will be the most intuitive to interpret, also~
## 10 "colnames(x) <- c(\"Treatment A\", \"Treatment B\")\nprint(x)\n\n        Tre~
## # i 19 more rows
```

文字列内の\n は改行を示しているが，そのままでは読みにくい．\n で改行して画面で表示するには cat() を使う．

```
cat(txt[1])  # [1] で 1 ページ目
```

pdf を docx に変換

```
library(tidyverse)
```

RDCOMClient

https://github.com/omegahat/RDCOMClient CRAN にはないが,

インストール

```
install.packages("RDCOMClient",
  repos = "http://www.omegahat.net/R",
  type = "win.binary")
```

変換実行

https://stackoverflow.com/questions/32846741/convert-pdf-file-to-docx/73720411#73720411

```
library(RDCOMClient)
wordApp <- COMCreate("Word.Application")
wordApp[["Visible"]] <- TRUE
wordApp[["DisplayAlerts"]] <- FALSE
path_To_PDF_File <- "xxx.pdf"
path_To_Word_File <- "xxx.docx"
doc <-
  wordApp[["Documents"]]$Open(normalizePath(path_To_PDF_File),
    ConfirmConversions = FALSE)
doc$SaveAs2(path_To_Word_File)
```

ラッパー関数

```
library(RDCOMClient)
pdf2docx <- function(pdf, docx = NULL){
  if(is.null(docx)){
    docx <- paste0(getwd(), sub("pdf", "docx", pdf))
  }
  wordApp <- RDCOMClient::COMCreate("Word.Application")
  wordApp[["Visible"]] <- TRUE
  wordApp[["DisplayAlerts"]] <- FALSE
  doc <-
    wordApp[["Documents"]]$Open(normalizePath(pdf), ConfirmConversions = FALSE)
  doc$SaveAs2(docx)
  doc$close()
}

wd <- "d:/matu/work/tmp/"
setwd(wd)
path_docx <- function(path_pdf){
  if(grepl("[A-z]:", path_pdf)){
    return(sub("pdf", "docx", path_pdf))
  }
  path <- file.path(getwd(), sub("pdf", "docx", path_pdf))
  return(sub("//", "/", path))
```

```
}
testthat::expect_equal(path_docx("a.pdf"                  ), "d:/matu/work/tmp/a.docx"     )
testthat::expect_equal(path_docx("d:/matu/work/tmp/a.pdf"), "d:/matu/work/tmp/a.docx"     )
testthat::expect_equal(path_docx("test/a.pdf"             ), "d:/matu/work/tmp/test/a.docx")
testthat::expect_equal(path_docx("/test/a.pdf"            ), "d:/matu/work/tmp/test/a.docx")


wd <- "d:/"
setwd(wd)
pdf2docx("a.pdf")
```

pdf2docx

Microsoft365R

Outlook で複数メール送信を一斉送信

複数人に全く同じメールを送る場合は，TO や CC に複数の電子メールアドレスを入力すれば良い．また，宛先を知られるのがよろしくないときは，BCC に送信先のアドレスを，TO に自分のアドレスを入れておけば問題ない．このとき，送り先の全員に全く同じ内容，同じ添付ファイルであればメールは 1 つ作成すれば問題ない．

でも，個々の人に対して少しだけ違う内容のメールを送りたいときとか，添付ファイルを別々のものにしたいときがある．また，単純なことだが，宛先が「みなさま」よりは，「　様」のように宛先だけでも変更したいというときもある．何かお願いをするときには，「みなさま」よりも直接名前を書いたほうが結構効果が高い．例えば，学会での投票のお願いなどは，ML に流すより個別メールの方が確実だ．

そのようなとき，いちいちメールを作成・編集していると面倒だし，間違いのもとになる．名前を中途半端に修正して，3 箇所のうち 1 箇所だけ別の人の名前にしてしまっていたり，日付と曜日があっていないなどの間違いは日常茶飯事だ．このような間違いをなくすには，個別に変更する部分と全体で統一するところを分けておき，あとはパソコンを使ってうまくつなぎ合わせる．でも，このように作成したメールの本文や宛先をいちいちコピー & ペーストするのは，手間がかかるし，個々にも作業のミスが入り込む余地が大きい．

インストールと初期設定

この操作は，最初に 1 回だけ実行すれば OK.

```
  # インストール
install.packages("Microsoft365R")
  # パッケージの読み込み
```

```
library(tidyverse)
library(Microsoft365R)
 # 会社など組織で契約している場合
Microsoft365R::get_business_outlook()
 # 個人利用の場合
  # Microsoft365R::get_personal_outlook()
```

とりあえず使う

まずは，試しにメールを作って送ってみる．

```
 # 会社などで組織で契約している場合
outlook <- Microsoft365R::get_business_outlook()
 # 個人利用の場合
```

```
  # outlook <- Microsoft365R::get_personal_outlook()

  # 個別に email を送る場合
  # メール作成のみ
  # メールは outlook の下書きフォルダにも保存されている
em <-
  outlook$create_email(
    body = "Hello from R\nHello from R\n",
    subject = "Hello",
    to = "matutosi@gmail.com",
    cc = "matutosi@konan-wu.ac.jp"
  )

  # メール送信
em$send()

  # outlook の下書きフォルダからメールを取り出す
drafts <- outlook$get_drafts()$list_emails()
  # 下書きフォルダのメール一覧
drafts
  # 下書きフォルダのメールの 1 つ目を送信
drafts[[1]]$send()

  # 受信トレイのメール一覧
inbox <- outlook$get_inbox()$list_emails()
  # 受信トレイの 1 つ目の内容
inbox[[1]]
```

メールの一斉送信

宛先や本文をエクセルに入力しておき，そこからデータを抽出して一斉にメールを送信できる．

- 送信: send(必須) 1: 送信する，0: 下書きに保存
- 宛先: to(必須)
- CC: cc(任意)
- BCC: bcc(任意)
- 件名: subject(必須でないが，入力推奨)
- 本文: body(必須でないが，入力推奨)
- 添付ファイル: attachment(任意)

宛先が入力されていないとメールは送信できない．CC と BCC は任意．
件名と本文はなくても送信できるが，両方とも何もないとメールの意味がない．
添付ファイルがあれば，ファイル名を指定．複数ファイルを添付するときは，カンマで path(ファイル名)を区切る．絶対 path で指定すると間違いは少ない．

```
  # 宛先や本文をエクセルで作成しておき
  #   一斉にメールを作成・送信する場合

  # 関数の読み込み
source("https://gist.githubusercontent.com/matutosi/bed00135698c8e3d2c49ef08d12eef9c/raw/6acc2de844eeeal
```

```
outlook <- Microsoft365R::get_business_outlook()
  # エクセルファイルの内容
  #   working directory にファイルがない場合は，
  #   絶対パス ("c:/user/documents/outlook.xlsx" など) で指定
path <- "outlook.xlsx"
  # メール作成・送信
create_email(path, outlook, send = TRUE)

  # メール作成のみ
  #   "send = FALSE" にすれば，メールを作成して下書きに保存
create_email(path, outlook, send = FALSE)
```

複数のワード文書の文字列を一括置換

多くのプログラマは，普段はそれぞれの好みのテキストエディタを使っていいるだろう．私は Windows では古典的なエディタである秀丸エディタを長らく使っている．キー割り当てのカスタマイズや自分用の細かなマクロがあるので，今さらエディタを変更できない．ートパソコンでは ThinkPad をずっと使っているので，キーボード自体も変更できない．これを変更すると作業効率が悪くなってしまう．そのため，デスクトップパソコンでも ThinkPad キーボードを愛用している．

このようにエディタとキーボードだけでパソコンの作業が完了すれば良いのだが，仕事上ワードで文書を作成しなければならないことがある．ワードは余計なおせっかいをたくさんしてくれるので，不要なことはしないように設定している．それでも，できればワードでの作業は最小限にしたいのが本音である．起動に時間はかかるし，置換で使える正規表現がちょっと変だからである．

R からワード文書内の文字列を置換すれば，ワードを起動する手間が省略できる．また，正規表現を使った置換や複数の組み合わせの置換もできる．さらに，「A を B」に「B を A」にという入れ替えも，プログラムで途中に別の文字列への置き換えで実現できる．このとき途中で使う文字列が元の文書内にないことは必須条件であるが，これもプログラムで確認可能である．もちろん，複数ファイルでの置換やファイル名を正規表現で指定することもできる．

置換のコードの例

```
  # https://ardata-fr.github.io/officeverse/index.html
  # https://github.com/omegahat/RDCOMClient

  # install.packages("officer")

pkg <- "D:/matu/work/ToDo/automater/R"
devtools::load_all(pkg)
library(officer)
library(tidyverse)
wd <- "d:/"
setwd(wd)

replacement <- read.table("replacement.txt", header = TRUE, sep = "\t")

files <-
  replacement[["file"]] %>%
  stringr::str_c(collapse = "|") %>%
  fs::dir_ls(regexp = .) %>%
  exclude(stringr::str_detect(., "^replaced\\_"))

replacement <- expand_file(replacement, files)
```

```
files %>%
  purrr::walk(replace_docs, replacement)
```

年月日の更新

毎年同じような文書を作成しているが，年だけを更新しなければならないことは多いだろう．手作業で日付を更新すると，どうしても間違いが混入する．単純な見間違いや入力間違いもあれば，日付を変更して曜日を変更し忘れる，あるいは日付を変更し忘れることをやってしまいがちだ．このような更新作業も，Word の検索・置換の機能で可能だし，R から特定の日付を別の日付に変換できる．

いっそのことなら，日付を文書内で自動的に取得して日付あるいは曜日を更新できれば楽ができる．例えば，「2023 年 4 月 10 日 (月)」を 2024 年に変更することを考えよう．何番目の何曜日かで日付が決まっているなら，2023 年 4 月 10 日は第 2 月曜日である．この場合は，2024 年 4 月の第 2 月曜日は「2024 年 4 月 8 日 (月)」なので，「2023 年 4 月 10 日 (月)」を「2024 年 4 月 8 日 (月)」に置換する．一方，日付固定なら「2024 年 4 月 10 日 (水)」に置換する．

さらに，求めた日が日曜日の場合は前日の土曜日あるいは月曜日にずらすとか，10 月 1 日の前後 3 日以内の火曜日のような法則でも可能である．祝日との関連で日付を決定することもあるだろう．そのようなときは，祝日データをあわせてコードに入れれば良い．とにかく，決め方が明確で 1 つに日付を決めることができれば，プログラムによる自動化できる．

ここでは lubridate を活用して，ワード文書の日付を更新する方法を扱う．lubridate で日付固定あるいは位置固定のときでの翌年の年月日を求める方法は以下を参考にして欲しい．

lubridate で日付・時刻を扱う

活用例

ワード文書内の日付は，正規表現を用いて入手できる．

それぞれの曜日なし版が考えられ，月と日が 1 桁の時に「04」のようにパディング (桁合わせ) されていることもあるだろう．これらは，正規表現によって対応可能である．もちろん，日付っぽい表記のすべてを含むことはできないが，よく使う日付表記は網羅できるだろう．年表記が 2 桁の場合，半角や全角のスペースを途中に含んだり，「()」の半角・全角の違いなどの表現揺れもあり得る．表記揺れを修正するための置換や削除などは，stringr(あるいは base) の関数で対応できる．

```
  # 20\d\d年月日
  # /

library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
x <- "21 年 1 月 1 日 (月), 2021 年 1 月 1 日 (月), 2021 年 10 月 10 日 (月), 2 月 2 日 (月), 12 月 22 日 (月), 2021/1/
regrep <- "((20)*[2-5]\\d+[年/_-]*)*\\d\\d*[月/_-]*\\d\\d*[日]*(\\([月火水木金土日]\\))*"

stringr::str_extract_all(x, regrep)
```

```
## [[1]]
##  [1] "21年1月1日(月)"     "2021年1月1日(月)"   "2021年10月10日(月)"
##  [4] "2月2日(月)"         "12月22日(月)"       "2021/1/1(月)"
##  [7] "2021/10/10(月)"     "2/2(月)"            "12/22(月)"
## [10] "21年1月1日"         "2021年1月1日"       "2021年10月10日"
## [13] "2月2日"             "12月22日"           "2021/1/1"
## [16] "2021/10/10"         "2/2"                "12/22"
```