R にる作業の効率化・自動化 -パッケージ活用術-

Toshikazu Masumura

2023-06-28

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

この文書は，以下の人を読者として想定している．

1 R を使って PC 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 エラーが出た時に自分で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 つ目は言うまでもないが，R を使って自動化をしたい人を対象としている．Python に慣れている人は，Python を使ったほうが早いかもしれない．あまり無いかもしれないが，Python では実行できず R だけで実行できるものがあるかもしれない．その場合は，その部分だけ R 本体で実行するとか，rpy2 ライブラリを使えば良いかもしれない．Python をふだん使っているが，新たに R を使ってみたいという人も，もちろん対象である．

2 つ目の R の基礎知識があれば良いが，既に仕事や研究などで R を多少使っていれば問題ないだろう．大学生や大学院生，あるいはこれまでの研究の中で，R をデータ解析に使用してきた研究者であれば，この文書の多くのコードをほぼそのまま利用できるはずだ．R を使ったことはなくても，他のプログラミング言語を使っていれば大丈夫である．R やパッケージのインストールについての説明をもとに R を導入して，magrittr や stringr などの部分を読めばある程度の理解ができるだろう．ただし，十分ではないと思われるので，しっかり勉強しようという人は，別の書籍を 1 冊読むことをおすすめする．

- R ではじめるデータサイエンス
- 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だが，OS の違いやその他の環境の違いによって，この文書の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．その場合，自分のまわりに R 上級者がいれば，質問できるだろう．しかし，そうでないときには，自分でネットで検索して解決しなければならない．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欲しい．できれば，日本語よりも英語で検索するのが良い．圧倒的に情報量が違うからだ．

エラーが発生して，検索しても解決しなければ質問を送って頂いても，もちろん構わない．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頂きたい．

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ある．特に作業を自動化するということは，危険な場合がある．この文書や改良したコードを実行して何らかのデータが消失したり，損害を与えたりする可能性がある．そのためコードの実行には注意をするとともに，自己責任で実行してほしい．

ファイルの上書き保存や削除の際には，細心の注意が必要だろう．ファイルの保存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だろう．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い．

特徴

この文書の特徴は，以下の 2 つがある．

- R を使った自動化の解説である
- コードをほぼそのまま実行できる (GitHub)

自動化について R(Python ではなく) を使った解説に焦点を当てていることである．Python を使った自動化については多くの書籍やネット上にも多くの文書がある．Python 自体がプログラミング言語で最も使われているものの一つで，自動化のためのライブラリも充実しているからだ．R は Python と比べられることがあるものの，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る．実際にそうではあるが，R にも結構多くの機能があり，自動化のためのパッケージも充実している．

Python は勉強したい言語や使っている言語の上位にあげられることが多い．一方，R は使いたくない言語としてあげられることがある．R は他の言語と比べるとちょっと独特なところがあるからだ．それでも使ったことがない人は，一度使ってみてほしい．独特なところが結構クセになる．

この文書のコードはコピペすれば，ほぼそのまま実行できる．GitHub にもコードがあるので，使ってほしい．読者の環境に合わせて，入力するべきとこももちろんある．その部分は，`your_directory` などのように表記している．

ただし，自動化の方法は 1 つだけではない．1 つのことをするにしても，`system()` でのコマンドの実行やパッケージの関数の利用などのように複数の方法がある．プログラムをする場合は，後で見直して自分で修正できるようにすることが重要だ．まずは，この文書のコードをコピペして使い，分かりにくい部分は自分なりの書き方に変えてほしい．その上でさらに改善を重ね，コードを組み合わせると独自の自動化手法ができる．

読み方

??? 章の R やパッケージのインストールについてを除いて，基本的には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読めば良いだろう．各章では，パッケージごとに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る．ただし，magirittr，stringr，fs パッケージは他の章でもよく使っているので，まずはここから読むと理解が早いと思われる．また，これら以外にも他の章で解説するパッケージを利用していることがあるので，必要に応じて関連部分を読んでほしい．

R とは

この文章では，プログラミング言語として 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 を使うライブラリとして 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 の総本山の 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．なお，パッケージの名前は分かっているが，内容がよくわからない場合は，`PackageName cran` で検索すると CRAN のページがひっかかることが多い．

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．

アーカイブされたパッケージ

zip ファイルとしてアーカイブ化されている場合は，devtools::install_local() でインストールできる．例えば，過去に CRAN に登録されたていたが削除されたパッケージやパッケージの古いバージョンである．CRAN の一覧からは削除されても，アーカイブ化されたものが保存されているため，そこから zip ファイルをダウンロードできる．パッケージの古いバージョンも同様である．devtools::install_local() はネットのものは直接インストールできないので，一旦ダウンロードしてからインストールする．

```
zip <-
  "http://cran.nexr.com/bin/windows/contrib/3.5/rMouse_0.1.zip" %>%
  curl::curl_download(fs::file_temp(ext = "zip"))
devtools::install_local(zip)
```

CRAN から Package 一覧を取得する

CRAN に登録されているパッケージは，2023 年 5 月現在で 2 万近くになっている．たくさんあることは嬉しい反面，目的とするパッケージを検索するのは困難である．パッケージ一覧のページで検索しても良いが，ブラウザでは正規表現が使えないことが多い．そこで，パッケージの一覧を取得して，自分のパソコンの中に一覧を保存して，その後で R やエディタの正規表現を用いて検索できるようにする．

```
library(tidyverse)
library(magrittr)
library(rvest)
  # wd <- "your_directory"
  # wd <- "D:/matu/work/tmp"
  # setwd(wd)
url <- "https://cran.r-project.org/web/packages/available_packages_by_name.html"
html <- rvest::read_html(url)  # rvest は第??? 章を参照
pkgs <-
  html %>%
  rvest::html_table(header = TRUE) %>%
  `[[`(1) %>% # .[[1]] と同じ
  magrittr::set_colnames(c("pkg", "description")) %>% # magrittr は第??? 章を参照
  stats::na.omit() %>%
  dplyr::mutate( # dplyr は第??? 章を参照，stringr は第??? 章を参照
    description = stringr::str_replace_all(description, "\n", " "))

readr::write_tsv(pkgs, "pkgs.txt") # readr は第??? 章を参照

dplyr::filter(pkgs, stringr::str_detect(description, "Image|image"))
```

詳しい説明は省略するが，以下を実行すると pkgs.txt というテキストファイルが作業ディレクトリに保存される．また，「Image」か「image」が description に含まれるものが出力される．すべてを画面に出力したい場合は，最後にコメントアウトした 3 行を実行する．検索結果を write_tsv() でテキストファイルとして保存するのも良いだろう．

パッケージとその説明の一覧を分析する方法は，tidyverse の章を参考にしてほしい．

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 remotes::install_github() を使う．本書で使用するパッケージ automater をインストールしてみる．

まずは，CRAN から remotes をインストールしておく．その後，install_github() の引数で，GitHub のリポジトリを指定する．

```
install.packages("remotes")
remotes::install_github("matutosi/automater")
## Downloading GitHub repo matutosi/automater@HEAD
## Installing 15 packages: rJava, SnowballC, semver, assertthat, binman, bitops, xlsxjars, magick, ...
## trying URL 'https://ftp.yz.yamagata-u.ac.jp/pub/cran/bin/windows/contrib/4.3/rJava_1.0-6.zip'
## Content type 'application/zip' length 1299141 bytes (1.2 MB)
## downloaded 1.2 MB
## # (中略)
## package 'rJava' successfully unpacked and MD5 sums checked
## # (中略)
##
## The downloaded binary packages are in
##         C:\Users\matu\AppData\Local\Temp\Rtmpkt9har\downloaded_packages
## # (中略)
## ** testing if installed package keeps a record of temporary installation path
## * DONE (automater)
```

automater では多くのパッケージに依存している．それらのパッケージがない場合は，まずは依存しているパッケージがダウンロード・インストールされる．その後，automater がインストールされるので，少し時間がかかるかもしれない．なお，`install_github()` でインストールする場合は，ソースコードからビルドする (要は自分のパソコン内でパッケージを作り上げる) ので，CRAN からのインストールよりは時間がかかることがある．

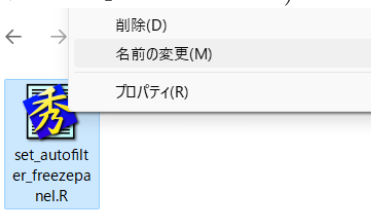
スクリプトの関連付け

Windows

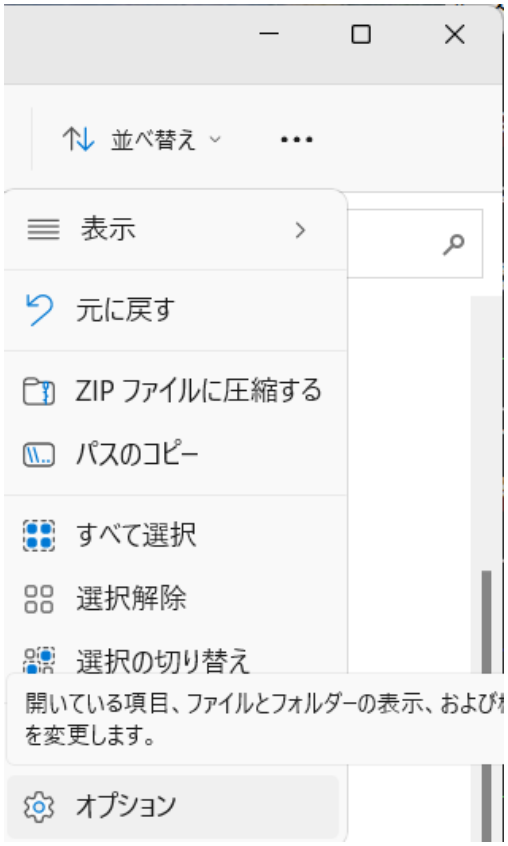
R のプログラムのファイルは拡張子「script.R」のように「R」という拡張子を付けて保存することが多い．拡張子「docx」をワードで，「xlsx」をエクセルで開くのと同様に，私は「R」をテキストエディタで開くするように設定している．その後，開いたファイルを R のコンソールに貼り付けて，プログラムを実行する．

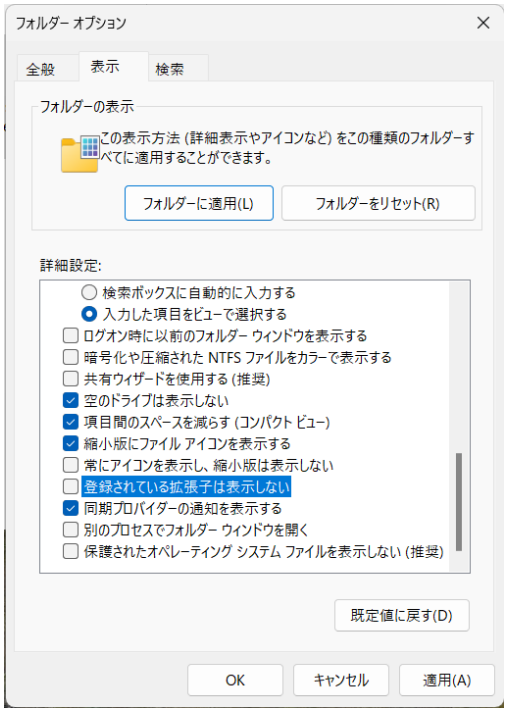
このような使い方でももちろん良いのだが，コードの内容を変更しないのであれば，いちいち R を起動してコードを貼り付けるのは面倒臭い．ファイルをクリックするだけで，プログラムが実行されれば便利である．プログラムのファイルを R に関連付けることで，これが実現できる．

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)．

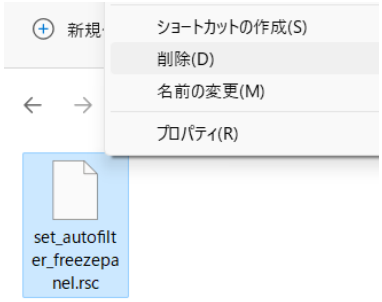


2. 拡張子が表示されていない場合は，エクスプローラの表示のオプションで，「登録されている拡張子は表示しない」のチェックを外して (チェックしないで)，「OK」を選択してから，名前を変更する．



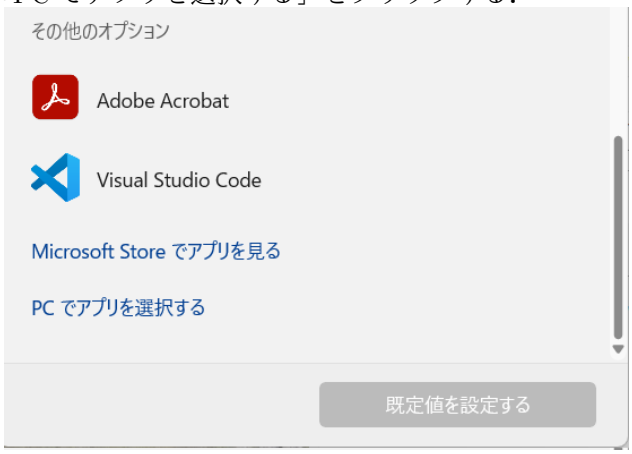


3. ファイルを右クリックして，「プロパティ」を選択する．

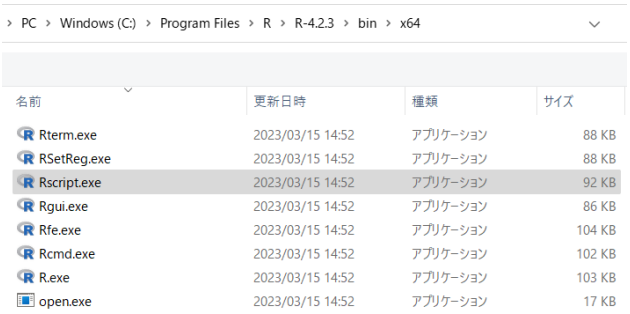


4.「全般」タブのやや上にあるプログラムの「変更」を選択する．

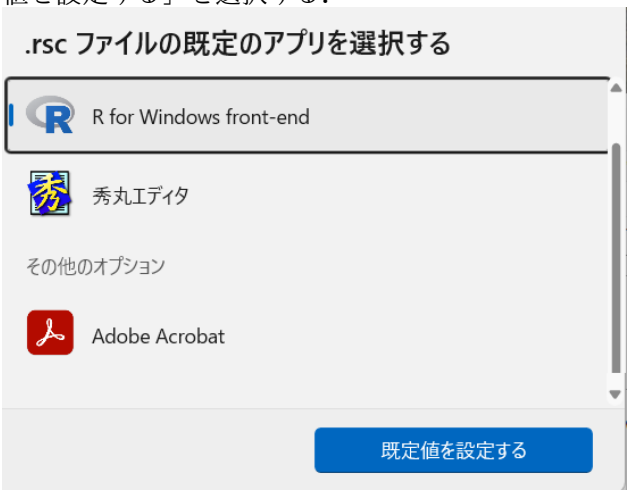
5.「PC でアプリを選択する」をクリックする．



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)．

Mac と Linux

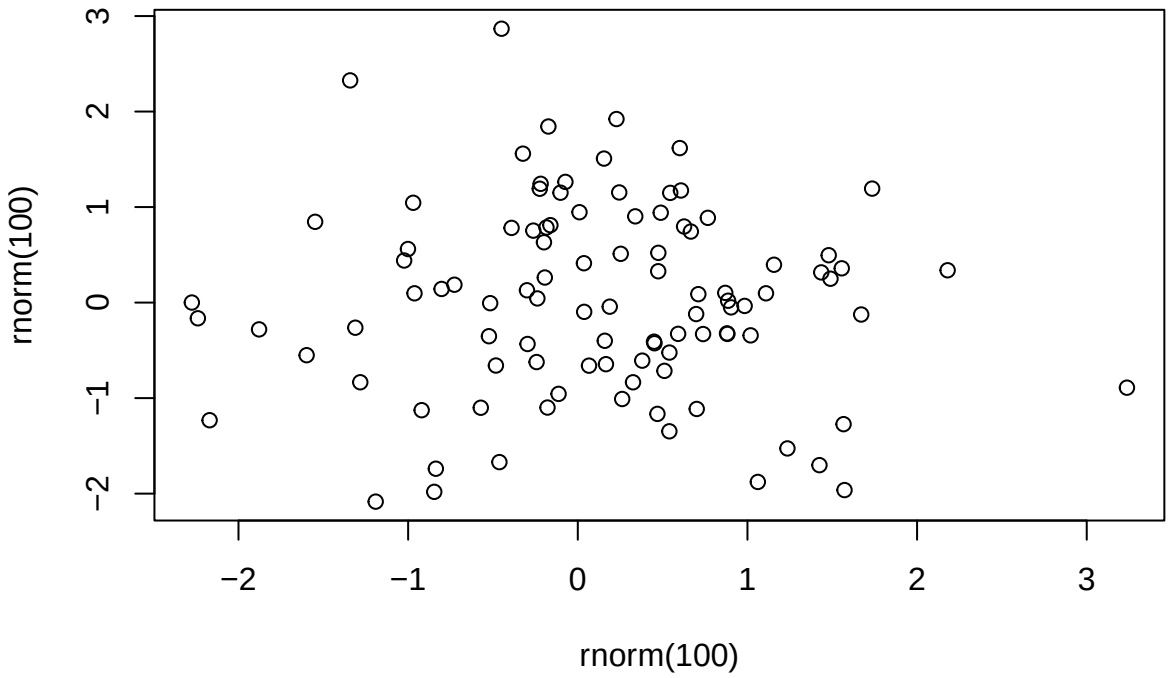
Mac で R のスクリプトファイルを簡単に実行するには以下の方法がある．

- plot.R などの名前で R のコードを保存
- シェルスクリプトのファイル (例：plot.sh) を用意

- plot.sh をクリックして実行

簡単な例として散布図を pdf に保存するコードを取り上げる．次のコードを R で実行すると，乱数 (x 軸：一様分布，y 軸：正規分布) に基づく散布図が pdf として保存される．

```
# コード例
  # ユーザディレクトリに test.pdf というファイルを生成
pdf("~/test.pdf")
  plot(1, 1)
dev.off()
```



このコードをユーザ・ディレクトリ (~/) に保存する．ファイル名は何でも良いが，ここでは plot.R とする．

次に，plot.R を実行するためのシェルスクリプトを作成する．このファイル名も何でも良いが，ここでは plot.sh とする．もちろん，R と別のファイル名でも構わない．plot.sh の中身は次のようにする．

```
#! /bin/bash
/usr/local/bin/Rscript ~/plot.R
```

#! ではじまる 1 行目はシバンとよばれるもので，ある種のおまじないである．/bin/bash というシェルで実行せよという意味である．2 行目の/usr/local/bin/Rscript は Rscript というプログラムを実行する部分である．R をインストールすると，/usr/local/bin/か/usr/bin/のどちらかに Rscript というプログラムが保存されている．どちらに保存されているかは，そのときの設定によると思われるが，読者の環境に合わせる．

~/plot.R の部分は実行する R のコードのファイルを指定する．他のファイルを実行する場合は，この部分を変更する．

これで準備が完了したので，plot.sh をクリックすれば plot.R が Rscript で実行され，plot.pdf を生成す

るはずである．うまくいかなければ，plot.sh に実行権限が設定されていないことが考えられる．Terminal から plot.sh のあるディレクトリで以下のように実行すれば，実行権限を設定する．

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

Linux でも同様の方法で実行できるようになるはずである．

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，パイプの手入力は手間がかかるので，ショートカットを利用する．RStudio では，Ctrl + Shift + M である．テキストエディタを使っているときは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)
```

コラム：magrittr の不思議な関数たち

magrittr は，そもそも変わったパッケージだが，不思議な名前の関数もある．

```
add          # `+`
subtract     # `-`
multiply_by  # `*`
divide_by    # `/`
```

?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．特に四則演算をする関数の場合は，パイプと一緒に使うと若干だが読みやすい．+ と同じ機能である add であれば，以下のようになる．

```
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
```

パイプで使わない場合は，普通の四則演算の方がわかりやすい．

```
1:10 + 100
```

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

```
add(1:10, 100)
```

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

論理演算で使う関数もパイプで使うと読みやすくなりそう．

```
and           # `&`
or            # `|`
equals        # `==`
not           # `!`, `n'est pas`(フランス語)も同じ
```

& や | で，条件が多くなってくると読みにくくなるので，改行するかパイプで繋いで 1 つ 1 つの条件を短くするとコードが読みやすくなる．

```
rep(c(TRUE, FALSE), times = 5) %>%
  and(rep(c(TRUE, FALSE), each = 5))
```

```
##  [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
```

```
rep(c(TRUE, FALSE), times = 5) %>%
  or(rep(c(TRUE, FALSE), each = 5))
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE
```

tidyverse を使う

準備

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

```
library(tidyverse)
```

tidyverse とは

tidyverse は単一のパッケージではなく，9 つのパッケージを含むパッケージ群である．

- dplyr：データフレーム操作
- forcats：ファクター (因子) 操作
- ggplot2：作図
- lubridate：日付・時間データ
- purrr：繰り返し処理
- readr：ファイル読み込み・保存
- stringr：文字列
- tibble：データフレームの拡張型
- tidyr：整然 (tidy) データのツール

このうち，この文書ではファクター (因子) をほとんど扱っていないため，forcats については説明しない．ggplot2 と lubridate はそれぞれ第? 章と第? 章で説明し，その他はこの章で説明する．以下では全体に関係するものから説明するため，上記の順序とは異なる (上記はアルファベット順)．

tibble でデータフレームを使いやすくする

データフレームと tibble

tidyvrse では，データフレーム (data.frame) をさらに拡張して使いやすくした tbl_df や tbl という形式を基本的に使う．ふつうに使っている限りはそれほど大きな違いはない．違いないというよりも，オブジェクト名そのものや `print()` で内容を表示させたときに見やすいという利点がある．例えば，各列のデータのタイプが表示される．int(整数) や chr(文字列) のように省略して表示されるので，若干の慣れが必要である．

tibble の気の利いた表示

画面の幅に表示を合わせてくれるのも良い点である．例えば，次のように画面の幅に収まりきらない場合でも，データフレームではダラダラと表示を続ける．一方，tibble では画面の幅に入る範囲に「良い感じ」にまとめて表示してくれる．それでも表示できなかった場合は，最後に列名とその形式を出力する．

```
n <- 3
ncol <- 10
cnames <- letters[seq(ncol)]
df <-
  matrix(rnorm(n * ncol), ncol = ncol, dimnames = list(seq(n), cnames)) %>%
  as.data.frame() %>%
  print()
```

```
##           a          b          c          d          e          f          g
## 1  1.237954 -1.3381755 -1.8180794 -0.6175788 -0.2404364 -0.4038622 -1.9291173
## 2  1.054952  0.1400425 -0.9664028 -0.8213629  3.0631890 -0.7440541  0.1435108
## 3 -2.183718 -0.9671053 -1.4852031 -0.6130506 -0.9884549  1.6607063  0.3464834
##            h           i          j
## 1  1.4732450  1.15784062  0.5739506
## 2 -0.4310592  0.51047016  2.1792001
## 3  0.9366664 -0.08246675 -0.4332046
```

```
tibble::as_tibble(df)
```

```
## # A tibble: 3 x 10
##       a      b      c      d      e      f      g      h       i      j
##   <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>   <dbl>  <dbl>
## 1  1.24 -1.34  -1.82  -0.618 -0.240 -0.404 -1.93   1.47   1.16    0.574
## 2  1.05  0.140 -0.966 -0.821  3.06  -0.744  0.144 -0.431  0.510   2.18
## 3 -2.18 -0.967 -1.49  -0.613 -0.988  1.66   0.346  0.937 -0.0825 -0.433
```

変なたとえかもしれないが，データフレームは「言われたことをそのままやりました」という何も考えずに与えられたことをするような感じで，tibble は「見やすく修正しておきました」という非常に気の利いた対応をする感じである．

さらに，tibble では行数が多いときは最初の 10 行だけ表示する．データフレームではすべてのデータを表示させるので，巨大なデータのときに，画面がやたらスクロールして困った経験があるかもしれない．tibble ではそれがない．なお，行数と列数は最初に書かれている．

```
n <- 21
ncol <- 3
cnames <- letters[seq(ncol)]
df <-
  matrix(rnorm(n * ncol), ncol = ncol, dimnames = list(seq(n), cnames)) %>%
  as.data.frame() %>%
  print()
```

```
##              a           b           c
## 1  -0.80220762 -1.591265423  0.037048759
## 2  -1.05711458 -0.249391158 -0.468637646
## 3   0.56136585  0.839355388  0.459678057
## 4   0.20836842 -1.045308032 -1.010863139
## 5   1.74778496  1.318956028 -0.087482662
## 6   0.28585502 -0.210633800  0.195402987
## 7  -0.07061353 -0.429689409  0.468140067
## 8   0.07092612 -0.430224566  0.620863896
## 9  -2.72652767  0.908615503 -0.440907172
```

```
## 10  0.48153629 -0.644694214  0.238347119
## 11 -1.73852902 -0.348119567 -0.448807514
## 12 -0.31722572 -1.270668001  1.485915351
## 13  1.52218592  0.663212106 -0.846873475
## 14 -1.40874589  1.873864833  0.002542226
## 15 -0.52970184  0.562416554  0.524452673
## 16 -0.85477170 -0.567118599  0.195060002
## 17  0.04822759 -0.561849040  0.504198882
## 18 -0.43113892  0.217106294 -1.561969769
## 19  1.59937288  0.110544687  0.527208398
## 20  0.17881449  1.084427216  1.017769475
## 21  1.44097577 -0.005455252 -0.270548917
```

```
tibble::as_tibble(df)
```

```
## # A tibble: 21 x 3
##          a      b       c
##      <dbl>  <dbl>   <dbl>
##  1 -0.802  -1.59   0.0370
##  2 -1.06   -0.249 -0.469
##  3  0.561   0.839  0.460
##  4  0.208  -1.05  -1.01
##  5  1.75    1.32  -0.0875
##  6  0.286  -0.211  0.195
##  7 -0.0706 -0.430  0.468
##  8  0.0709 -0.430  0.621
##  9 -2.73    0.909 -0.441
## 10  0.482  -0.645  0.238
## # i 11 more rows
```

表示したい行数を指定するには，引数 n で指定する．以下のコードのうち，3 つ目を実行すると全行が表示されるので以下では表示を省略している．

```
mpg                   # 通常表示
print(mpg, n = 30)    # 30 行を表示
print(n = nrow(mpg)) # 全行を表示
```

tibble への変換，tibble の生成

すでにデータフレームがある場合は，as_tibble() で tibble への変換ができる．

```
head(mtcars)
```

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

```
tibble::as_tibble(mtcars)
```

```
## # A tibble: 32 x 11
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  21       6  160    110  3.9   2.62  16.5     0     1     4     4
```

```
##  2  21       6  160    110  3.9   2.88  17.0     0     1     4     4
##  3  22.8     4  108     93  3.85  2.32  18.6     1     1     4     1
##  4  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
##  5  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2
##  6  18.1     6  225    105  2.76  3.46  20.2     1     0     3     1
##  7  14.3     8  360    245  3.21  3.57  15.8     0     0     3     4
##  8  24.4     4  147.    62  3.69  3.19  20       1     0     4     2
##  9  22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2
## 10  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
## # i 22 more rows
```

新たに tibble を生成するには，データフレームの場合と同様である．

```
n <- 10
data.frame(x = runif(n), y = rnorm(n))
```

```
##              x          y
## 1  0.13491130 -1.5750830
## 2  0.30525859  0.8789261
## 3  0.22594227 -2.0733670
## 4  0.65321344 -0.5281563
## 5  0.94372924  0.5533367
## 6  0.06285072  0.6827684
## 7  0.09707375 -1.8007587
## 8  0.36063390  1.7918703
## 9  0.60974899  0.6101272
## 10 0.73546566 -0.1888498
```

```
tibble::tibble(x = runif(n), y = rnorm(n))
```

```
## # A tibble: 10 x 2
##         x      y
##     <dbl>  <dbl>
##  1 0.814   0.788
##  2 0.645  -0.507
##  3 0.399   0.968
##  4 0.726   0.185
##  5 0.0465  0.288
##  6 0.842  -0.659
##  7 0.687   0.740
##  8 0.560   0.377
##  9 0.576  -0.516
## 10 0.454   0.316
```

tidyr でデータを整形する

tidyverse の tidy とは，整然としたという意味である．データフレームや tibble なっていれば tidy かといえば，そうとは限らない．エクセルなどのスプレッド形式にデータが保存されていても tidy ではないのと同様である．

tidy データとは，次の 4 つを満たすもので，データベースの第 3 正規と同様のものである．

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

例えば，次のデータ `household` は非整然 (messy，tidy の対義語) データの典型的な例である．

```
relig_income
```

```
## # A tibble: 18 x 11
##    religion `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k` `$50-75k` `$75-100k`
##    <chr>      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>      <dbl>
##  1 Agnostic      27        34        60        81        76       137        122
##  2 Atheist       12        27        37        52        35        70         73
##  3 Buddhist      27        21        30        34        33        58         62
##  4 Catholic     418       617       732       670       638      1116        949
##  5 Don't t ~     15        14        15        11        10        35         21
##  6 Evangel~     575       869      1064       982       881      1486        949
##  7 Hindu          1         9         7         9        11        34         47
##  8 Histori~     228       244       236       238       197       223        131
##  9 Jehovah~      20        27        24        24        21        30         15
## 10 Jewish        19        19        25        25        30        95         69
## 11 Mainlin~     289       495       619       655       651      1107        939
## 12 Mormon        29        40        48        51        56       112         85
## 13 Muslim         6         7         9        10         9        23         16
## 14 Orthodox      13        17        23        32        32        47         38
## 15 Other C~       9         7        11        13        13        14         18
## 16 Other F~      20        33        40        46        49        63         46
## 17 Other W~       5         2         3         4         2         7          3
## 18 Unaffil~     217       299       374       365       341       528        407
## # i 3 more variables: `$100-150k` <dbl>, `>150k` <dbl>,
## #   `Don't know/refused` <dbl>
```

宗教と所得についてのデータである．1 列目の宗教は問題ないが，所得という変数が 1 つの列ではなく 2 列目以降の複数の列に広がっている．これを整理するには，`pivot_longer()` を使う．

1 つ目の引数にはデータフレームを指定する．ここでは `relig_income` だが，パイプが用いられているので省略されている `cols` には整形する列を指定する．ここでは，`!religion` で `religion` 以外を指定している．`names_to` には列名をもとに作る新たな変数 (列) 名を，`values_to` にはデータの値をもとに作る新たな変数 (列) 名を文字列で指定する．

```
relig_income %>%
  tidyr::pivot_longer(
    cols = !religion,
    names_to = "income",
    values_to = "count"
  )
```

```
## # A tibble: 180 x 3
##    religion income             count
##    <chr>    <chr>              <dbl>
##  1 Agnostic <$10k                 27
##  2 Agnostic $10-20k               34
##  3 Agnostic $20-30k               60
##  4 Agnostic $30-40k               81
##  5 Agnostic $40-50k               76
##  6 Agnostic $50-75k              137
##  7 Agnostic $75-100k             122
##  8 Agnostic $100-150k            109
##  9 Agnostic >150k                 84
## 10 Agnostic Don't know/refused    96
## # i 170 more rows
```

pivot_longer() で出力されたデータフレームは，それぞれの変数が 1 列に，それぞれの観測が 1 行になった．3 つ目と 4 つ目の条件にもがっていしており，これで，tidy データの条件を満たすことができた．基本的にはプログラミングの際には tidy な状態にしておくのが便利である．

ただし，入手データが tidy ではないことはよくある．また，関数によっては tidy ではないオブジェクトを入力する必要やプログラム途中で tidy ではない方が都合が良いこともある．tidy ではないものから tidy なものへの変換，あるいはその逆をするのに便利な関数が tidyr には多くある．次のコードでパッケージ内の関数一覧が取得できる．

```
ls("packages:tidyr")
```

以下も参考にしてほしい．

https://tidyr.tidyverse.org/

なお，手前味噌で申し訳ないが，ピボットテーブルをより簡単に作成するためのパッケージ pivotea を作成したので，興味があれば使ってみてほしい．

https://cran.r-project.org/web/packages/pivotea/

readr でファイルの保存と読み込み

tidy な状態に整理したデータをファイルとして保存や読み込みをするには，readr の関数を使うのが良い．base の write.table() や read.table() よりも使いやすく，読み込んだデータを tibble 形式にしてくれるのが良い．

保存するときは，write_csv(カンマ区切り) や write_tsv(タブ区切り) を使う．

```
readr::write_csv(mpg, file = "file_path")
readr::write_tsv(mpg, file = "file_path")
```

読み込むときは，read_csv(カンマ区切り) や read_tsv(タブ区切り) を使う．各列のデータ形式を指定したい場合は，col_types を使う．

```
readr::read_csv("file_path")
readr::read_tsv("file_path")
```

ここでは，第? 章で保存したデータを読み込む．

```
pkgs <- readr::read_tsv("pkgs.txt", show_col_types = FALSE)
pkgs
```

```
## # A tibble: 19,530 x 2
##    pkg           description
##    <chr>         <chr>
##  1 A3            Accurate, Adaptable, and Accessible Error Metrics for Predicti~
##  2 AalenJohansen Conditional Aalen-Johansen Estimation
##  3 AATtools      Reliability and Scoring Routines for the Approach-Avoidance Ta~
##  4 ABACUS        Apps Based Activities for Communicating and Understanding Stat~
##  5 abbreviate    Readable String Abbreviation
##  6 abbyyR        Access to Abbyy Optical Character Recognition (OCR) API
##  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 19,520 more rows
```

col_types でタイプを指定していないと色々と表示されて面倒なので，show_col_types = FALSE で表示を抑制している．他にも読み込み時のオプションがあるので，?read_tsv で確認してほしい．

なお，エクセルのファイルを読み込むパッケージには，readxl がある．

dplyr でデータフレームを操作する

dplyr はデータフレームを操作するためのパッケージである．列の追加や選択，行の抽出や並べ替え，グループ化，集計などができる．なお，dplyr の関数名の多くは SQL を参考にしていると思われるので，SQL を使ったことがあればコードの内容を理解しやすいだろう．

CRAN のパッケージを分類する

ここでは，読み込んだデータフレーム pkgs の CRAN に登録されているパッケージを分類する．mutate() を使って，description に特定の文字列が含まれるかを判定して，その結果を真偽値の新しい列として追加する．まずは str_detect() で description の列に”ocr”を含むかどうかを TRUE または FALSE で返す．その結果を ocr という列として新たに追加する．

```
pkgs %>%
  dplyr::mutate(ocr = stringr::str_detect(description, "ocr"))
```

```
## # A tibble: 19,530 x 3
##    pkg           description                                              ocr
##    <chr>         <chr>                                                    <lgl>
##  1 A3            Accurate, Adaptable, and Accessible Error Metrics for Pr~ FALSE
##  2 AalenJohansen Conditional Aalen-Johansen Estimation                    FALSE
##  3 AATtools      Reliability and Scoring Routines for the Approach-Avoida~ FALSE
##  4 ABACUS        Apps Based Activities for Communicating and Understandin~ FALSE
##  5 abbreviate    Readable String Abbreviation                             FALSE
##  6 abbyyR        Access to Abbyy Optical Character Recognition (OCR) API  FALSE
##  7 abc           Tools for Approximate Bayesian Computation (ABC)         FALSE
##  8 abc.data      Data Only: Tools for Approximate Bayesian Computation (A~ FALSE
##  9 ABC.RAP       Array Based CpG Region Analysis Pipeline                 FALSE
## 10 ABCanalysis   Computed ABC Analysis                                    FALSE
## # i 19,520 more rows
```

mutate() での新しい列名は，「“」を使わずに指定する．対話的に使っているときには，このように列名を直接入力するのが楽である．なお，「”」を使わずに列名を変数名のように指定 (評価) する方法を NSE(Non Standard Evaluation，非標準評価) という．ただし，プログラミングの途中で使っているときには列名が一義的に決まらず，入力したデータの列名を使いたいことがある．そのような時には，all_of(“colnames”) のように文字列として列名を指定したほうが良いことがある．

また，mutate() に似た関数として transmute() がある．transmute() は，関数内で指定した列以外は削除する点が mutate() とは異なる．

上のコードでは，1 つの文字列に対して列を追加した．ここでは複数の文字列に対して同じことをするために，関数を作成する．mutate() の内部がややトリッキーな事になっているが，実行内容は上と同じである．意味としては，str_detect() で出力した真偽値を kwd の文字列の中身を列名として新たな列を追加している．関数内では，新しい列名を NSE ではなく，文字列として標準評価として用いるために，列名の文字列を {{}} で囲っている．同様の理由で = ではなく := という rlang パッケージの関数を使っている．また，str_detect() 内で.data[[col]] としている．これは，文字列で列を選択するときに使用する手法である．

```
  # キーワードを列名として追加する関数，合致するときは TRUE
add_kwd <- function(df, col, kwd){
  pattern <- stringr::regex(kwd, ignore_case = TRUE)
  df %>%
    dplyr::mutate(`:=`({{kwd}}, stringr::str_detect(.data[[col]], pattern)))
}
  # キーワードの一覧
```

```
kwds <-
  c("database", "excel", "file", "ggplot", "image|magick", "keyboards|mouse",
    "ocr", "office", "pdf", "python", "scrape|scraping|selenium", "shell")
  # キーワード列の追加
for(i in seq_along(kwds)){
  pkgs <- add_kwd(pkgs, "description", kwds[i])
}
pkgs
```

```
## # A tibble: 19,530 x 14
##    pkg           description          database excel file  ggplot `image|magick`
##    <chr>         <chr>                <lgl>    <lgl> <lgl> <lgl>  <lgl>
##  1 A3            Accurate, Adaptable~ FALSE    FALSE FALSE FALSE  FALSE
##  2 AalenJohansen Conditional Aalen-J~ FALSE    FALSE FALSE FALSE  FALSE
##  3 AATtools      Reliability and Sco~ FALSE    FALSE FALSE FALSE  FALSE
##  4 ABACUS        Apps Based Activiti~ FALSE    FALSE FALSE FALSE  FALSE
##  5 abbreviate    Readable String Abb~ FALSE    FALSE FALSE FALSE  FALSE
##  6 abbyyR        Access to Abbyy Opt~ FALSE    FALSE FALSE FALSE  FALSE
##  7 abc           Tools for Approxima~ FALSE    FALSE FALSE FALSE  FALSE
##  8 abc.data      Data Only: Tools fo~ FALSE    FALSE FALSE FALSE  FALSE
##  9 ABC.RAP       Array Based CpG Reg~ FALSE    FALSE FALSE FALSE  FALSE
## 10 ABCanalysis   Computed ABC Analys~ FALSE    FALSE FALSE FALSE  FALSE
## # i 19,520 more rows
## # i 7 more variables: `keyboards|mouse` <lgl>, ocr <lgl>, office <lgl>,
## #   pdf <lgl>, python <lgl>, `scrape|scraping|selenium` <lgl>, shell <lgl>
```

列がたくさんできたので，そのうちのいくつかを選択および抽出してみよう．

データフレームの特定の列を選択するには，`select()` を使う．`select()` でも NSE が使えるので，列名をそのまま入力する．特定の列を除きたいときは，`-`あるいは`!` を使う．なお，文字列で列を指定するには，`all_of()` と `any_of()` を使うことができる．さらに，`starts_with()` や `ends_with()`，`contains()` などもある．

`filter()` は，データフレームから条件に合致した行を抽出する．最も単純な抽出方法は `ocr == TRUE` のように真偽値 (logical，TRUE か FALSE) として判定されるものであれば，どのようなものでも構わない．次のコードの場合は，そもそも `ocr` が真偽値なので，`ocr == TRUE` とせずに `ocr` としても同じ結果を得られる．

```
pkgs %>%
  dplyr::select(pkg, description, ocr) %>%
  dplyr::filter(ocr == TRUE)
```

```
## # A tibble: 10 x 3
##    pkg                description                                          ocr
##    <chr>              <chr>                                                <lgl>
##  1 abbyyR             "Access to Abbyy Optical Character Recognition (OCR~ TRUE
##  2 coalitions         "Bayesian \"\"Now-Cast\"\" Estimation of Event Prob~ TRUE
##  3 elastes            "Elastic Full Procrustes Means for Sparse and Irreg~ TRUE
##  4 googleCloudVisionR "Access to the 'Google Cloud Vision' API for Image ~ TRUE
##  5 paco               "Procrustes Application to Cophylogenetic Analysis"  TRUE
##  6 pcv                "Procrustes Cross-Validation"                        TRUE
##  7 ProcMod            "Informative Procrustean Matrix Correlation"         TRUE
##  8 RSocrata           "Download or Upload 'Socrata' Data Sets"             TRUE
##  9 soql               "Helps Make Socrata Open Data API Calls"             TRUE
## 10 tesseract          "Open Source OCR Engine"                             TRUE
```

```
dplyr::select(pkgs, -description) # dplyr::select(pkgs, !description) も同じ
```

```
## # A tibble: 19,530 x 13
##    pkg        database excel file  ggplot `image|magick` `keyboards|mouse` ocr
##    <chr>      <lgl>    <lgl> <lgl> <lgl>  <lgl>          <lgl>             <lgl>
##  1 A3         FALSE    FALSE FALSE FALSE  FALSE          FALSE             FALSE
##  2 AalenJoha~ FALSE    FALSE FALSE FALSE  FALSE          FALSE             FALSE
##  3 AATtools   FALSE    FALSE FALSE FALSE  FALSE          FALSE             FALSE
##  4 ABACUS     FALSE    FALSE FALSE FALSE  FALSE          FALSE             FALSE
##  5 abbreviate FALSE    FALSE FALSE FALSE  FALSE          FALSE             FALSE
##  6 abbyyR     FALSE    FALSE FALSE FALSE  FALSE          FALSE             TRUE
##  7 abc        FALSE    FALSE FALSE FALSE  FALSE          FALSE             FALSE
##  8 abc.data   FALSE    FALSE FALSE FALSE  FALSE          FALSE             FALSE
##  9 ABC.RAP    FALSE    FALSE FALSE FALSE  FALSE          FALSE             FALSE
## 10 ABCanalys~ FALSE    FALSE FALSE FALSE  FALSE          FALSE             FALSE
## # i 19,520 more rows
## # i 5 more variables: office <lgl>, pdf <lgl>, python <lgl>,
## #   `scrape|scraping|selenium` <lgl>, shell <lgl>
```

お気づきかもしれないが pkgs は tidy ではないので，ここで整形しておく．

```
pkgs <-
  tidyr::pivot_longer(pkgs,
    cols = -c(pkg, description), names_to = "kwd", values_to = "val") %>%
    dplyr::filter(val) %>%
    print()
```

```
## # A tibble: 958 x 4
##    pkg              description                                      kwd   val
##    <chr>            <chr>                                            <chr> <lgl>
##  1 abbyyR           Access to Abbyy Optical Character Recognition (~ ocr   TRUE
##  2 abjData          Databases Used Routinely by the Brazilian Jurim~ data~ TRUE
##  3 ABPS             The Abnormal Blood Profile Score to Detect Bloo~ file  TRUE
##  4 ace2fastq        ACE File to FASTQ Converter                      file  TRUE
##  5 AcousticNDLCodeR Coding Sound Files for Use with NDL              file  TRUE
##  6 activPAL         Advanced Processing and Chart Generation from a~ file  TRUE
##  7 ADAPTS           Automated Deconvolution Augmentation of Profile~ file  TRUE
##  8 add2ggplot       Add to 'ggplot2'                                 ggpl~ TRUE
##  9 adepro           A 'shiny' Application for the (Audio-)Visualiza~ file  TRUE
## 10 adfExplorer      Import from and Export to Amiga Disk Files       file  TRUE
## # i 948 more rows
```

tidy になったデータフレームを集計する．group_by() は，指定した列のカテゴリーに従って，グループ化する．ここでは，kwd の文字列でグループ化している．print() で表示させると Groups: kwd [12] となっており，kwd によって 12 のグループになっていることがわかる．なお，グループ化を解除するには，ungroup() を用いる．

その後，summarise() でグループごとの集計ができる．n() でグループごとの行数を計算する．なお，tally() は summarise(n = n()) のショートカットである．

arrange() は，指定した列のデータの順序に従って並べ替えをする．何も指定しなければ昇順，desc() で列名を指定すれば降順で並べ替えができる．

```
pkgs %>%
  dplyr::filter(val) %>%
  dplyr::group_by(kwd) %>%
  print() %>%
```

```
  dplyr::summarise(n = n()) %>% # tally() も同じ
  print() %>%
  dplyr::arrange(desc(n))
```

```
## # A tibble: 958 x 4
## # Groups:   kwd [12]
##    pkg              description                                          kwd   val
##    <chr>            <chr>                                                <chr> <lgl>
##  1 abbyyR           Access to Abbyy Optical Character Recognition (~ ocr   TRUE
##  2 abjData          Databases Used Routinely by the Brazilian Jurim~ data~ TRUE
##  3 ABPS             The Abnormal Blood Profile Score to Detect Bloo~ file  TRUE
##  4 ace2fastq        ACE File to FASTQ Converter                          file  TRUE
##  5 AcousticNDLCodeR Coding Sound Files for Use with NDL                  file  TRUE
##  6 activPAL         Advanced Processing and Chart Generation from a~ file  TRUE
##  7 ADAPTS           Automated Deconvolution Augmentation of Profile~ file  TRUE
##  8 add2ggplot       Add to 'ggplot2'                                     ggpl~ TRUE
##  9 adepro           A 'shiny' Application for the (Audio-)Visualiza~ file  TRUE
## 10 adfExplorer      Import from and Export to Amiga Disk Files           file  TRUE
## # i 948 more rows
## # A tibble: 12 x 2
##    kwd                         n
##    <chr>                   <int>
##  1 database                  155
##  2 excel                      25
##  3 file                      364
##  4 ggplot                    165
##  5 image|magick              153
##  6 keyboards|mouse             6
##  7 ocr                        10
##  8 office                     19
##  9 pdf                        24
## 10 python                     18
## 11 scrape|scraping|selenium   15
## 12 shell                       4
```

```
## # A tibble: 12 x 2
##    kwd                         n
##    <chr>                   <int>
##  1 file                      364
##  2 ggplot                    165
##  3 database                  155
##  4 image|magick              153
##  5 excel                      25
##  6 pdf                        24
##  7 office                     19
##  8 python                     18
##  9 scrape|scraping|selenium   15
## 10 ocr                        10
## 11 keyboards|mouse             6
## 12 shell                       4
```

ここでは紹介しなかったが他の有用な関数として，2 つのデータフレームを列の内容に合わせて結合 (マージ) する `left_join()` がある．これににた関数として，右側の引数 (つまり第 2 引数) をもとにして結合する `right_join()`，全結合する `full_join()`，マッチしなかった行を返す `anti_join()` などがある．詳しい使い方は，ヘルプを参照してほしい．

また，重複行を除去する `distinct()` もよく使う関数である．

stringr で文字列操作

はじめに

stringr は stringi パッケージのラッパーである．stringi は文字列操作のパッケージで，文字コードの変換などを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だろう．

stringr には，

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．しかし，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く，見た目が良く，何よりもバグが入りにくい (入らないわけではない)．

準備

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

```
library(tidyverse)
library(stringr)  # 本来は不要
library(fs)
```

stringr の主な関数

stringr では主な関数は `str_` で始まるようになっている．これらの一覧は次のとおりである．ちなみに，`str_subset()` で正規表現に合致した要素だけを返している．

```
ls("package:stringr") %>%
  stringr::str_subset("^str_")
```

```
##  [1] "str_c"             "str_conv"          "str_count"
##  [4] "str_detect"        "str_dup"           "str_ends"
##  [7] "str_equal"         "str_escape"        "str_extract"
## [10] "str_extract_all"   "str_flatten"       "str_flatten_comma"
## [13] "str_glue"          "str_glue_data"     "str_interp"
## [16] "str_length"        "str_like"          "str_locate"
## [19] "str_locate_all"    "str_match"         "str_match_all"
## [22] "str_order"         "str_pad"           "str_rank"
## [25] "str_remove"        "str_remove_all"    "str_replace"
## [28] "str_replace_all"   "str_replace_na"    "str_sort"
## [31] "str_split"         "str_split_1"       "str_split_fixed"
## [34] "str_split_i"       "str_squish"        "str_starts"
## [37] "str_sub"           "str_sub_all"       "str_sub<-"
## [40] "str_subset"        "str_to_lower"      "str_to_sentence"
## [43] "str_to_title"      "str_to_upper"      "str_trim"
## [46] "str_trunc"         "str_unique"        "str_view"
## [49] "str_view_all"      "str_which"         "str_width"
## [52] "str_wrap"
```

また，合致した以外のものを返すには，`negate = TRUE` とする．

```
ls("package:stringr") %>%
  stringr::str_subset("^str_", negate = TRUE)
```

```
##  [1] "%>%"          "boundary"     "coll"         "fixed"        "fruit"
##  [6] "invert_match" "regex"        "sentences"    "word"         "words"
```

このうち fruits，words，sentences は，それぞれ果物の名前，一般的な単語，文のサンプルデータである．`fixed()` は正規表現を使わずに，文字列を正規表現ではなくそのままの文字列として使う関数である．これら以外のものについて知りたい場合は，ヘルプを参照して欲しい．

```
  # stringr::str_detect()
  # stringr::str_replace_all()
  # stringr::str_c()
  # stringr::str_detect()
  # stringr::str_extract_all()
  # stringr::str_length()
  # stringr::str_split()
  # stringr::str_subset()
  # stringr::str_sub()

  # stringr::str_dup()
  # stringr::str_extract()
  # stringr::str_pad()
  # stringr::str_trunc()

  # stringr::str_replace_all()
  # stringr::str_to_lower()
```

活用例

```
  # fs::dir_ls()

  # パッケージ名の一覧を検索するという活用例
  # fs::dir_ls(regexp = "Rmd")
  # install.packages("")
  # library()
  # pkg::fun()


  # stringr::fixed()
  # stringr::regex()

  # fs.Rmd
  # fs::path_dir("path")          # パスからディレクトリ名抽出
  # fs::path_file("path")         # パスからファイル名抽出
  # fs::path_ext("path")          # パスから拡張子抽出
  # fs::path_ext_remove("path")   # パスから拡張子を削除
  # path_ext_set("path", "new_ext")    # 拡張子変更
```

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] "2023-06-28" "2024-06-28" "2025-06-28" "2026-06-28" "2027-06-28"
```

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

```
## [1] "2023-06-28" "2024-06-27" "2025-06-27" "2026-06-27" "2027-06-27"
```

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

```
## [1] "2023-06-28" "2023-07-28" "2023-08-28" "2023-09-28" "2023-10-28"
```

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

```
## [1] "2023-06-28" "2023-07-28" "2023-08-28" "2023-09-28" "2023-10-28"
```

なお，`today()` はそのときの本日の日付を返す関数のため，上記の表示と読者が実行したときの表示は異なるはずである．また，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")` とする．

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] 4
```

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

```
## [1] 水
## 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 <- ymd("2023-05-01")
x + years(1)
```

```
## [1] "2024-05-01"
```

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．

```
testthat::expect_equal(real, expect)
```

念のため，パッケージ testthat で確認する．testthat::expect_equal() の結果として，何も出力されなければテストを通過したことが分かる．

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 date as ", x[i], "!")
    }
  }
  return(same_pos)
}
```

実際の日付で確認してみる．

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

```
## Warning in same_pos_next_yr(days): No same date as 2023-05-29!
```

```
## Warning in same_pos_next_yr(days): No same date as 2023-05-30!
```

```
days
```

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

```
days_n
```

```
##  [1] "2024-05-06" "2024-05-07" "2024-05-01" "2024-05-02" "2024-05-03"
##  [6] "2024-05-04" "2024-05-05" "2024-05-13" "2024-05-14" "2024-05-08"
```

```
## [11] "2024-05-09" "2024-05-10" "2024-05-11" "2024-05-12" "2024-05-20"
## [16] "2024-05-21" "2024-05-15" "2024-05-16" "2024-05-17" "2024-05-18"
## [21] "2024-05-19" "2024-05-27" "2024-05-28" "2024-05-22" "2024-05-23"
## [26] "2024-05-24" "2024-05-25" "2024-05-26" NA           NA
## [31] "2024-05-29"
```

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] 1 1 1 1 1 1 1 2 2 2 2 2 2 2 3 3 3 3 3 3 3 4 4 4 4 4 4 4 5 5 5
mweek(days_n)
## [1]  1  1  1  1  1  1  1  2  2  2  2  2  2  2  3  3  3  3  3  3  3  4  4  4  4  4  4  4 NA NA  5
sum(mweek(days) != mweek(days_n), na.rm = TRUE)
## [1] 0
testthat::expect_equal(mweek(days), mweek(days_n))
## Error: mweek(days) not equal to mweek(days_n).
## 2/31 mismatches (average diff: NaN)
## [29] 5 - NA == NA
## [30] 5 - NA == NA
```

少しだけ NA のところがあるが，それ以外はうまくいっているようだ．testthat でのテストでも 29 日目と 30 日目がうまく行っていないことがわかる．

さらに，曜日も確認してみる．

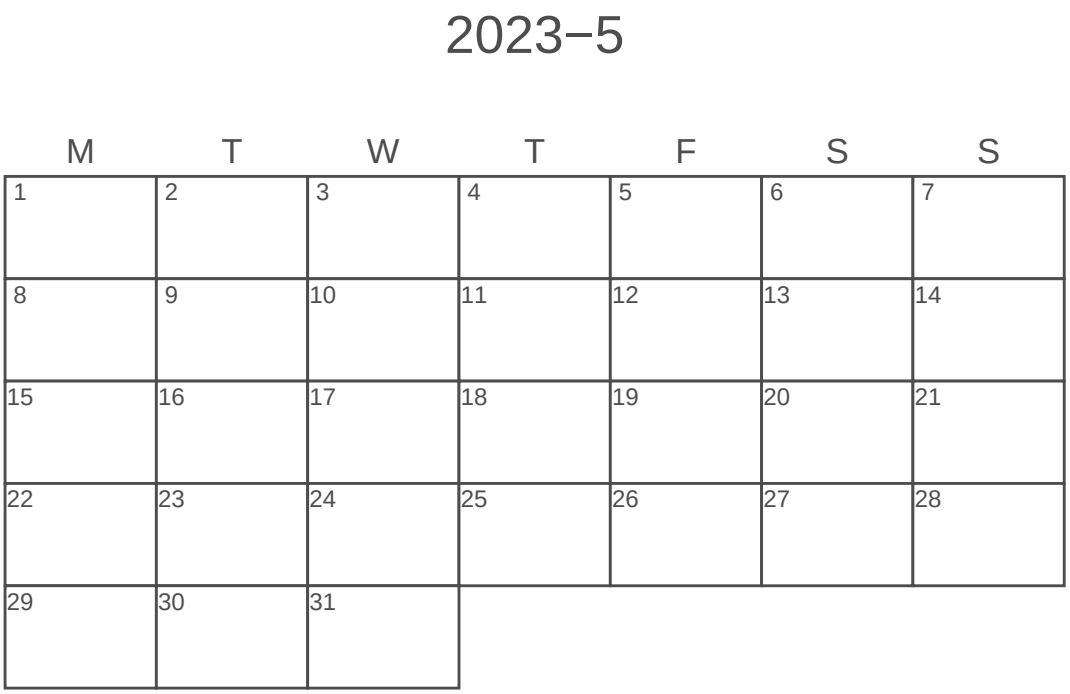
```
wday(days, label = TRUE)   # 曜日
##  [1] 月 火 水 木 金 土 日 月 火 水 木 金 土 日 月 火 水 木 金 土 日 月 火 水 木 金 土 日 月 火 水
## Levels: 日 < 月 < 火 < 水 < 木 < 金 < 土
wday(days_n, label = TRUE)
##  [1] 月   火   水   木   金   土   日   月   火   水   木   金   土   日   月   火   水   木   金   土
## [21] 日   月   火   水   木   金   土   日   <NA> <NA> 水
## 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))
## Error: wday(days) not equal to wday(days_n).
## 2/31 mismatches (average diff: NaN)
## [29] 2 - NA == NA
## [30] 3 - NA == NA
```

こちらも，最後の方でエラーが出ている．なお，このエラーは第 5 週でずれるためにでてくる部分である．ただし，28 日しかない 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．

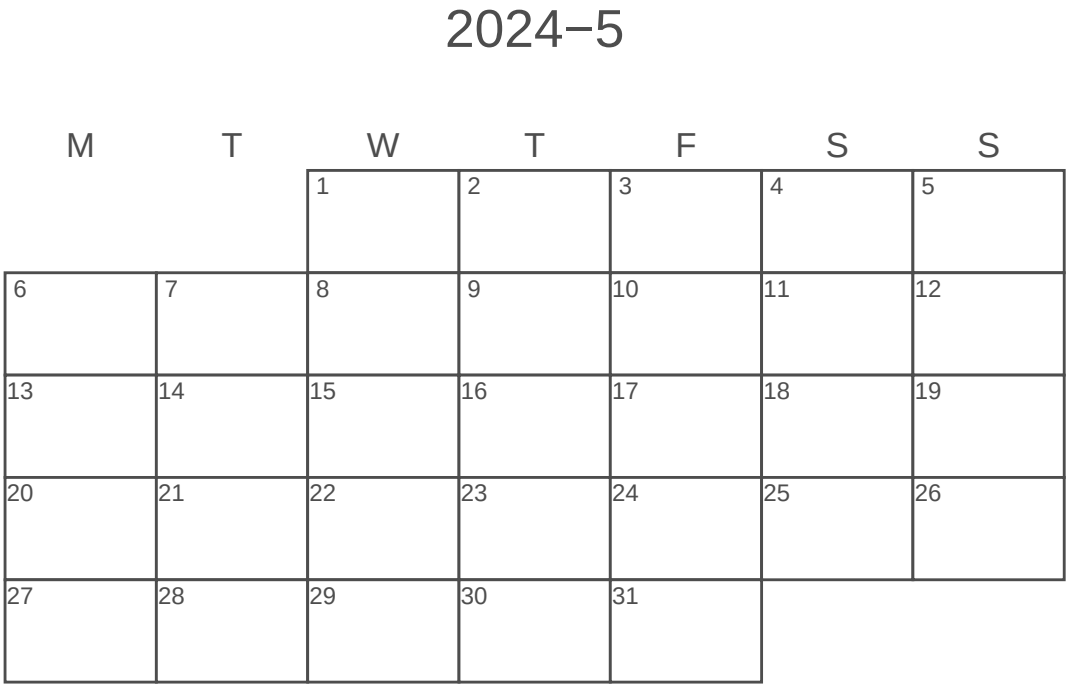
テストもだいたいあっていそうだが，分かりにくいため，カレンダーで表示してみる．

```
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)
```



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，名前の変更やファイル名によるディレクトリの振り分けを大量にするなら，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 (`system()` や `shell()`) 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，`system()` 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とることができる．

なお，fs，base，シェルの詳細な比較が，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)
```

シェル，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．

シェルを使う

シェルなら，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．テキストエディタ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．

```
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 やシェル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，シェルの比較は次の 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

パス操作は fs パッケージの関数を使うと簡単にできる．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．パス操作の関数を自作するには，自作してもある程度はできるが，stringr を駆使しなければならない．自作した関数にはバグが入っている可能性がある．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

シェルや 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

ファイル操作はシェルや 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．

コラム：GUI での作業ディレクトリの指定

GUI(Graphical User Interface)，つまりマウス操作による作業ディレクトリを指定するには，tcltk パッケージを使うと良い．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ない．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()
library(tcltk)
wd <- tcltk::tkchooseDirectory()
setwd(fs::path(wd))
getwd()
```

tcltk::tkchooseDirectory() を実行すると，ディレクトリを選択する画面が表示されるので，使いたいディレクトリを指定する．

system() や shell() で OS コマンドの実行

R から `system()` で OS のコマンドを実行できる (Windows では `shell()` も使える)．例えば，以下のようなことだ．

- ファイルの移動
- PDF ファイルの結合
- png から PDF へ変換
- Python スクリプトの実行

見れば分かるように，これらはこの文書で紹介するパッケージで実行できることとほぼ同じである．この文書の執筆前は，便利なパッケージの存在自体を知らなかったため，かなり `system()` や `shell()` を使っていた．しかし，便利なパッケージがあるのが分かれば，コマンドを自分で書く必要はない．便利なものを使うのが良い．わざわざ `system()` で実行するならば，自作のバッチファイル (Windows) やシェルスクリプト (Mac や Linux) などだろうか．あるいは自分で関数やパッケージを開発する際に使ってみることがあるかもしれない．一応，こんなこともできるんだという程度に知っておけばよいだろう．

以下では，R から Python とそのライブラリをインストールする．普通ならシェルで直接実行するものだが，ここでは R をシェルの代わりとして使う．利点は，R を起動していれば新たなシェルを起動せずにそのまま実行できること，返り値を R のオブジェクトに代入して使えることである．欠点は，キー入力が若干増えることである．

準備

`system()` や `shell()` などは R の base パッケージに含まれている．ここでは，組み合わせて使用する curl パッケージをインストールしておく．

```
install.packages("curl")
```

```
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
```

```
library(curl)
```

```
## Using libcurl 7.84.0 with Schannel
##
## Attaching package: 'curl'
##
## The following object is masked from 'package:readr':
##
##     parse_date
```

活用例：Python とそのライブラリのインストール

Python とそのライブラリのインストールを R から実行してみよう．Python を直接使わなくても，インストールしておいて損はない．R にはないライブラリが Python にはあり，reticulate パッケージを使えば R から Python を実行できるためだ．ここではさらに，キーボードとマウス操作の自動化で使用する Pillow と PyAutoGUI というライブラリをインストールする．

Python のイントール

まずは，Python のインストーラをダウンロードする．Windows の場合は，以下を実行するとデフォルトのブラウザで Python のインストラのダウンロードページが開く．なお，ここで利用している `shell.exec()` は Windows で使える関数で，拡張子に関連付けられたファイルを実行するものである．

```
shell.exec("https://pythonlinks.python.jp/ja/index.html")
```

Mac や Linux には，通常はデフォルトで Python3 がインストールされている．必要に応じてバージョンアップして欲しい．また，Python2 がインストールされていても，使わない場合はアンインストールしておくことをお勧めする．複数のバージョンがインストールされていると，`Python` というコマンドで Python2 が起動して混乱を招くおそれがあるためだ．

2023 年 5 月現在での Python の最新版は 3.11.1 なので，以下ではそのファイルの URL を入力している．最新版が異なる場合は，適宜 URL を変更してほしい．curl_download で一時ファイルとしてインストーラをダウンロードする．なお，`py_installer` にはダウンロードした一時ファイル名を保存している．

```
py_installer <-
  "https://www.python.org/ftp/python/3.11.1/python-3.11.1-amd64.exe" %>%
  curl::curl_download(fs::file_temp(ext = "exe"))
```

ダウンロードしたファイルを実行するために，再度 `shell.exec()` を使う．これを実行すると，Python のインストーラが起動するので，その指示に従って Python をインストールする．最初の `Add python.exe to PATH` にチェックが入っていなければ，追加でチェックをすれば良いだろう．それ以外は，デフォルトのボタンをクリックしていけば OK である．

内容がわかっていて，自分流でインストールしたい場合はカスタマイズすると良い．カスタマイズする場合でも，インストールする項目で `pip` のチェックは入れたままにしておく．Pillow と PyAutoGUI のインストールに使用するからだ．

```
shell.exec(py_installer)
```

これで Python がインストールできたはずだが，念のため確認しておく．インストールしたバージョンによって表示が若干異なるが，以下のコードを実行してバージョンが表示されれば，Python のインストールは成功である．

```
system("python -V", intern = TRUE)
```

```
## [1] "Python 3.11.1"
```

Python のインストールがうまくいかない場合は，以下を参考にして欲しい．

https://www.python.jp/install/windows/install.html

Pillow と PyAutoGUI のインストール

次に，Pillow と PyAutoGUI をインストールする．インストールする前に，インストール済のライブラリを確認する．`pip list` は，Python のライブラリ管理ソフトである pip を使ってインストール済みライブラリ一覧を出力するコマンドである．初めて Python をインストールしたのであれば，一覧に何も表示されないはずなので，Pillow と PyAutoGUI をインストールする．もし，一覧の中に”Pillow”と”PyAutoGUI”という文字列があれば，インストールは不要である．

```
cmd <- "pip list"
res <- system(cmd, intern = TRUE)
tibble::as_tibble(res) %>%
  dplyr::filter(stringr::str_detect(value, "Pillow|PyAutoGUI"))
## # A tibble: 0 × 1
##   1 variable: value <chr>
>
```

```
cmd <- "pip install Pillow PyAutoGUI"
res <- system(cmd, intern = TRUE)
head(res)
## [1] "Collecting Pillow"
## [2] "  Using cached Pillow-9.5.0-cp311-cp311-win_amd64.whl (2.5 MB)"
## [3] "Collecting PyAutoGUI"
## [4] "  Using cached PyAutoGUI-0.9.53-py3-none-any.whl"
## [5] "Requirement already satisfied: pymsgbox in c:\\users\\username\\appdata\\local\\programs\\pytho
## [6] "Requirement already satisfied: PyTweening>=1.0.1 in c:\\users\\username\\appdata\\local\\progra
tail(res)
## [1] "Requirement already satisfied: pyperclip in c:\\users\\username\\appdata\\local\\programs\\pyth
## [2] "Installing collected packages: Pillow, PyAutoGUI"
## [3] "Successfully installed Pillow-9.5.0 PyAutoGUI-0.9.53"
## [4] ""
## [5] "[notice] A new release of pip available: 22.3.1 -> 23.1.2"
## [6] "[notice] To update, run: python.exe -m pip install --upgrade pip"
```

最後の方に”Successfully installed Pillow-9.5.0 PyAutoGUI-0.9.53”のような表示があればインストールができているはずだ．念のため pip で確認する．以下では書き方をわざと変えているが，ライブラリのインストール前と実行内容は同じである．一覧の中に”Pillow”と”PyAutoGUI”という文字列があれば，インストールされていることが確認できる．

```
"pip list" %>%
  system(intern = TRUE) %>%
  tibble::as_tibble() %>%
  dplyr::filter(stringr::str_detect(value, "Pillow|PyAutoGUI"))
## # A tibble: 2 × 1
##   value
##   <chr>
## 1 Pillow          9.5.0
## 2 PyAutoGUI       0.9.53
```

ところで，2 つ前のコードのところで，“[notice]”として pip の最新版がある．これは，pip をバージョンアップしてはどうかという提案である．せっかくなので，バージョンアップしておこう．

```
res <- system("python.exe -m pip install --upgrade pip", intern = TRUE)
res
##  [1] "Requirement already satisfied: pip in c:\\users\\username\\appdata\\local\\programs\\python\\p
##  [2] "Collecting pip"
##  [3] "  Using cached pip-23.1.2-py3-none-any.whl (2.1 MB)"
##  [4] "Installing collected packages: pip"
##  [5] "  Attempting uninstall: pip"
##  [6] "    Found existing installation: pip 22.3.1"
##  [7] "    Uninstalling pip-22.3.1:"
##  [8] "      Successfully uninstalled pip-22.3.1"
##  [9] "  WARNING: The scripts pip.exe, pip3.11.exe and pip3.exe are installed in 'C:\\Users\\username
## [10] "  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-
```

```
## [11] "Successfully installed pip-23.1.2"
```

最後に”Successfully installed pip-23.1.2”のような表記があれば，バージョンアップ成功である．これで Python およびライブラリ Pillow と PyAutoGUI のインストールさらに pip のバージョンアップが完了した．

Windows でのアプリ・ディレクトリの瞬間起動 {#run_in a_second}

ここでは，Windows での便利な機能を紹介する．Path の通ったところにショートカットを保存し，アプリやディレクトリを一発で起動できるというものである．これは非常に便利な機能である．デスクトップが多くのアプリのショートカットがあったり，スタートメニューからアプリを探す人にはぜひ使ってほしい．これを覚えておくだけでもかなりの時間が節約できるだろう．

- ショートカットをパスの通ったディレクトリに保存
- [Win] + [R] で「ファイル名を指定して実行」を起動
- ショートカットの名称を入力して [Enter]
- アプリやディレクトリが起動する
  なお，Mac でも Linux でもパスの通ったところにあるファイルやディレクトリであれば，シェルから起動できる．Windows の「ファイル名を指定して実行」の代わりとして，Mac は Spotlight([Command] + [Space]) が使える．

例えば，RStudio のショートカットをパスの通った `C:\Windows\System32`(システムファイルがあるので，パスが通っているはず) に”rst”という名前で保存すると (管理者権限が必要)，[Win] + [R] に続けて”rst”と入力すると RStudio を起動できる．他のディレクトリにショートカットを保存する場合は，そのディレクトリにパスを通しておく．

上記のショートカットの作成やパスを通す作業はそれぞれ 1 回だけなのでマウスでの手作業でも全く問題ない．でもせっかくなので，ここでは R からショートカットを作成する．もともとパスの通っている `C:\Windows\System32` にショートカットを作っても良いのだが，管理者権限が必要なため R からの操作が難しい．そこで，`C:\Windows\USERNAME\shortcut`(USERNAME は環境によって異なる) というディレクトリを作成し，ここにショートカットを保存するとともに，パスを通すことにする．以下のコードでは，`shell()` を利用した automater の関数で，RStudio と R のショートカットを作成するとともに，ショートカットを作成したディレクトリにパスを通している．

```
  # RStudioのショートカット作成
  # 実際のパスと異なる場合は変更する
exe <- fs::path("C:/Program Files/RStudio/rstudio.exe")
shortcut <- "rst"
wd <- Sys.getenv("R_USER")
size = 3
res <- automater::make_shortcut(exe, shortcut = shortcut, size = size, wd = wd)

  # Rのショートカット作成
  # 実際のパスと異なる場合は変更する
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"
wd <- Sys.getenv("R_USER")
size = 3
res <- automater::make_shortcut(exe, shortcut = shortcut, arg = arg, size = size, wd = wd)
```

```
  # ショートカットを作成したディレクトリにパスを通す
new_path <-
  fs::path_dir(res$shortcut) %>%
  automater::add_path()
```

これで，[Win] + [R] で出てきた「ファイル名を指定して実行」に”rst”を入力すればRStudio，“r”を入力すればRが起動する．

なお，automater::make_shortcut と automater::add_path の中身は以下のとおりである．独自で関数を作成する際の参考になれば幸いである．

```
automater::make_shortcut
```

```
## function (exe, shortcut = NULL, dir = NULL, arg = NULL, size = 1,
##     wd = NULL)
## {
##     exe <- double_quote(exe)
##     if (is.null(dir)) {
##         dir <- fs::path(Sys.getenv("USERPROFILE"), "shortcut")
##         if (!fs::dir_exists(dir)) {
##             fs::dir_create(dir)
##         }
##     }
##     else {
##         if (!fs::dir_exists(dir)) {
##             stop("directory ", dir, " not found!")
##         }
##     }
##     if (is.null(shortcut)) {
##         shortcut <- fs::path_file(exe)
##     }
##     shortcut <- fs::path(dir, shortcut) %>% fs::path_ext_set("lnk") %>%
##         double_quote()
##     wsh <- paste0("$WsShell = New-Object -ComObject WScript.Shell;")
##     create <- paste0("$Shortcut = $WsShell.CreateShortcut(",
##         shortcut, ");")
##     target <- paste0("$Shortcut.TargetPath = ", exe, ";")
##     icon <- paste0("$Shortcut.IconLocation = ", exe, ";")
##     size <- paste0("$ShortCut.WindowStyle = ", size, ";")
##     if (!is.null(arg)) {
##         arg <- double_quote(arg)
##         arg <- paste0("$ShortCut.Arguments = ", arg, ";")
##     }
##     if (!is.null(wd)) {
##         wd <- double_quote(wd)
##         wd <- paste0("$ShortCut.WorkingDirectory = ", wd, ";")
##     }
##     finish <- "$Shortcut.Save()"
##     input <- paste0(wsh, create, target, icon, size, arg, wd,
##         finish)
##     cmd <- "powershell"
##     res <- shell(cmd, input = input, intern = TRUE)
##     shortcut <- stringr::str_remove_all_all(shortcut, "\"")
##     return(list(shortcut = shortcut, res = res))
## }
## <bytecode: 0x00000293e297ff60>
```

```
## <environment: namespace:automater>
```

```
automater::add_path
```

```
## function (new_path)
## {
##     cmd <- "reg query \"HKEY_CURRENT_USER\\Environment\" /v \"path\""
##     path <- shell(cmd, intern = TRUE)[3] %>% stringr::str_remove(" *path *REG_[A-z]* *") %>%
##         double_quote()
##     cmd <- paste0("setx path ", normalizePath(new_path), ";",
##         path)
##     shell(cmd, intern = TRUE)
## }
## <bytecode: 0x00000293e29fc5e0>
## <environment: namespace:automater>
```

Deeplr で翻訳する

DeepL は高機能な翻訳ツールで，結構自然な文章を提案してくれる．Google Translate もそれなりに素晴らしい翻訳をしてくれるが，それよりも良さそうな文章になることが多い．DeepL のページに文章を入力・貼り付けするだけで簡単に使える．しかし，ネットでのフリー版だと 1 回あたり 5000 文字という制約があるため，文章量が増えると手作業が面倒になってくる．

DeepL の API を使えば，1 ヶ月あたり 50 万文字までは無料で翻訳が可能である．R から直接操作するのはちょっと面倒だが，CRAN に登録されている deeplr パッケージのお世話になれば簡単に DeepL を使いこなせる．

準備

```
install.packages("deeplr")
```

```
library(tidyverse)
library(deeplr)
```

DeepL の API

DeepL の API を使うには，事前に認証キーを取得しなければならない．

deeplr

有料版の Pro を契約しているときは `translate()` を，無料版の free のときは `translate2()` を使う．この文章の冒頭部分を英語に翻訳してみる．`str_detect()` の正規表現で`"^[^\x01-\x7E]"` として全角文字ではじまる文字列を抽出している．正規表現は若干ややこしいが，`[^\x01-\x7E]"` の `^` と`\x01-\x7E` はそれぞれ”以外”と”1 バイト文字”の意味，最初の `^` は”行頭”の意味である．つまり，1 バイト文字以外が行頭のものということだ．

```
text <-
  readr::read_tsv("index.Rmd", col_names = "jp", show_col_types = FALSE) %>%
  dplyr::filter(stringr::str_detect(jp, "^[^\x01-\x7E]")) %>%
  head()
en <-
  text %>%
  `$`("jp") %>%
  print() %>%
  purrr::map_chr(deeplr::translate2,
```

```
    target_lang = "EN", source_lang = "JP", auth_key = "your_api_key") %>%
  print()

## [1] "誰でもそうだろうが，面倒くさい仕事はしたくない．"
## [2] "というか，したくないことが面倒くさいのだろう．"
## [3] "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．"
## [4] "でも，しなければならないのなら自動化したい．"
## [5] "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
## [6] "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．
## [1] "I don't want to do a tedious job, as anyone would."
## [2] "Or perhaps it is the hassle of not wanting to."
## [3] "Aside from the chicken and egg argument, I don't want to go through the hassle if I can help it
## [4] "But if I have to do it, I want to automate it."
## [5] "Of course, not all tasks can be automated, and some tasks, such as writing, should not be autom
## [6] "Programming languages are often used to automate tasks."
```

無事に翻訳できたが，自分の文章を読むよりも英語に翻訳したものを読む方がなぜか恥ずかしい感じがする．上のように英語だけをベクトルにしてもよいが，できれば英語と日本語を並べて見たほうがわかりやすい．あとから `tibble::tibble(en = en, jp = jp)` としても同じであるが，以下のようにすることもできる．なお，`relocate()` は列の表示順を変更しているだけであまり大した意味はない．

```
translated <-
  dplyr::mutate(text, en = deeplr::translate2(jp,
    target_lang = "EN", source_lang = "JP", auth_key = "your_api_key") %>%
  dplyr::relocate(en, jp) %>%
  print()
## # A tibble: 6 × 2
##   en                                                    jp
##   <chr>                                                 <chr>
## 1 I don't want to do a tedious job, as anyone would.    誰でもそうだろ…
## 2 Or perhaps it is the hassle of not wanting to.        というか，した…
## 3 Aside from the chicken and egg argument, I don't want… ニワトリとタマ…
## 4 But if I have to do it, I want to automate it.        でも，しなけれ…
## 5 Of course, not all tasks can be automated, and some t… もちろんすべて…
## 6 Programming languages are often used to automate task 作業の自動化に…
```

reticulate で Python を使う

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)
library(automater)
```

Python と PyAutoGUI のインストール

Python とそのライブラリのインストールを参照して，Python と PyAutoGUI をインストールしておく．

ライブラリの一覧の中に，あとで使う numpy，matplotlib，art がなければインストールしておく．なお，この文書では Python をパッケージ版でインストールしするとともに，pip でパッケージを管理している．Anaconda や miniconda で仮想環境を使用している場合は，`retuculate::conda_install()` でパッケージをインストールする．

```
system("pip list", intern = TRUE) %>%
  tibble::as_tibble() %>%
  dplyr::filter(stringr::str_detect(value, "numpy|matplotlib|art"))
## A tibble: 0 × 1
##   1 variable: value <chr>
```

一覧に”numpy”，“matplotlib”，“art”などのライブラリ名がなければ，以下のコードでライブラリをインストールする．

```
system("pip install numpy", intern = TRUE)
system("pip install matplotlib", intern = TRUE)
system("pip install art", intern = TRUE)
```

ライブラリがインストールできたか念のため確認したい場合は，以下のようにする．

```
system("pip list", intern = TRUE) %>%
  tibble::as_tibble() %>%
  dplyr::filter(stringr::str_detect(value, "numpy|matplotlib|art"))
## # A tibble: 3 × 1
##   value
##   <chr>
## 1 art             5.9
## 2 matplotlib      3.7.1
## 3 numpy           1.24.2
```

使用する Python の指定

1 つのパソコンにバージョンや形態の異なる Python を複数インストールすることが可能である．Python とそのライブラリのインストールで説明したような Python パッケージのインストーラを用いたもの (パッケージ版 Python) もあれば，Microsoft Store 版の Python もある．さらに，Anaconda をインストールしてその中で Python を使うこともできる．色々とあってややこしいが，この文章ではパッケージ版 Python を使うことにする．reticulate で使用する Python としてインストールしたパッケージ版 Python を指定する．なお，Anaconda などにある Python を使う場合は，別途インストールして，`use_condaenv()` で指定する．

ここでは，パッケージ版 Python を使用する．Python のパスが分かってれば，`use_python()` でそのパスを指定する．パスが分からなければ，以下のように `find_python()` を使うことができる．Python が 1 つだけのときは，パスがそのまま取得できる．

複数の Python がインストールされている場合は，メニューが表示される．例えば，以下のような文字列が入っている可能性がある．C:/Users/your_user_name/AppData/Local/r-miniconda/envs/r-reticulate/python.exe C:/Users/your_user_name/AppData/Local/Microsoft/WindowsApps/python.exe

メニューが表示されれば，使用する Python のパスの番号を入力する．どれを使っていいのかわからない場合は，“0”でとりあえず全部のパスを出力する．その後，それぞれのパスの Python がどのようなものか確認してからどれを使うか決める．

なお，Windows でパスに”WindowsApps/python.exe”があるときは，このファイルはアプリ実行エイリアスという機能で Python をインストールための実行ファイルである．以下を参考にアプリ実行エイリアスの設定でチェックを外すとパスの一覧からは出なくなる．

参考：https://hrkworks.com/it/programming/python/py-4421/

```
find_python <- function(select_memu = TRUE){
  os <- get_os()
  python_path <-
    ifelse(os == "win", "where python", "which python") %>%
    system(intern = TRUE) %>%
    fs::path()
  if(length(python_path) > 1){
    if(select_memu){
      choice <- utils::menu(python_path, title = "Select Python path. 0: exit and return all.")
      if(choice == 0){ return(python_path) }
    }else{
      return(python_path)
    }
  }else{
    choice <- 1
  }
  return(python_path[choice])
}
path <- find_python()
path
```

```
## C:/Users/matutosi/AppData/Local/Programs/Python/Python311/python.exe
```

```
reticulate::use_python(path)
```

Python を使ってみる

Python の呼び出し

repl_python() を実行すると R から Python を呼び出すことができる．Python 実行時にはコンソールが > から >>> に変化する．

古典的なことだが，まずは Hellow World! を実行する．“Hellow World!”が出力されたら成功である．コンソールが >>> のときに exit か quit とすれば R に戻る．

```
repl_python()
## Python 3.11.1 (C:/Users/username/AppData/Local/Programs/Python/Python311/python.exe)
## Reticulate 1.28 REPL -- A Python interpreter in R.
## Enter 'exit' or 'quit' to exit the REPL and return to R.
>>> print("Hello World!")
## Hellow World!
>>> exit
```

他にも Python で実行したいことがあれば，>>> のときに入力する．

R から Python の Hellow World!

Python にあらかじめ用意されているビルトイン関数を使うには，import_builtins() を使う．import_builtins() で生成したオブジェクトに $ と Python の関数名をつければ，R の関数として使うことができる．

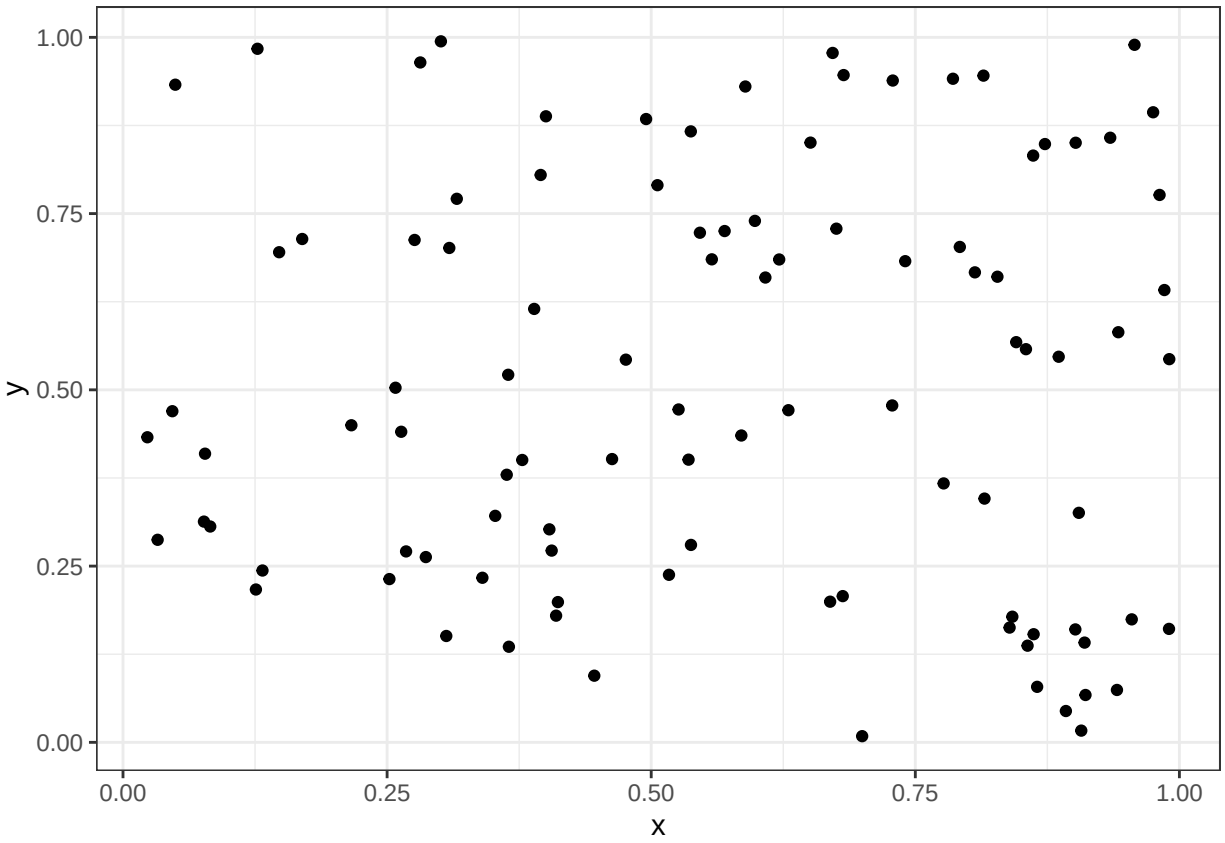
```
builtins <- reticulate::import_builtins()
builtins$print('Hellow World!')
```

ふつうに Hellow World! をしていても面白くないので，ちょっと変わった Hellow World! をしてみる．ライブラリ art を呼び出して，Hellow World! をするとちょっと面白い．

```
art <- reticulate::import("art")
art$tprint("Hellow World!")
##  _   _          _  _                    __          __                  _       _  _
## | | | |  ___  | || |  ___  __      __ \ \        / /  ___   _ __  | |  __| || |
## | |_| | / _ \| || | / _ \ \ \ /\ / /  \ \ /\ / /  / _ \ | '__|| | / _` || |
## |  _  ||  __/| || || (_) | \ V  V /    \ V  V /  | (_) || |   | || (_| ||_|
## |_| |_| \___||_||_| \___/   \_/\_/      \_/\_/    \___/ |_|   |_| \__,_|(_)
art$tprint("FUN", font = "block", chr_ignore = TRUE)
##  .----------------.  .----------------.  .----------------.
## | .--------------. || .--------------. || .--------------. |
## | |  _________   | || | _____  _____ | || | ____  _____  | |
## | | |_   ___  |  | || ||_   _||_   _|| || ||_   \|_   _| | |
## | |   | |_  \_|  | || |  | |    | |  | || |  |   \ | |   | |
## | |   |  _|      | || |  | '    ' |  | || |  | |\ \| |   | |
## | |  _| |_       | || |   \ `--' /   | || | _| |_\   |_  | |
## | | |_____|      | || |    `.__.'    | || ||_____|\____| | |
## | |              | || |              | || |              | |
## | '--------------' || '--------------' || '--------------' |
##  '----------------'  '----------------'  '----------------'
```

遊んでばかりいても仕方ないので，ちょっと真面目に使ってみよう．Python で，`np.random.rand` と表記するものを R で使うには，`np$random$rand` とする．R のオブジェクトに代入してしまえば，あとは慣れたもので簡単に散布図が作成できる．

```
np <- reticulate::import("numpy") # import numpy as np と同じ
x <- np$random$rand(as.integer(100))
y <- np$random$rand(as.integer(100))
tibble::tibble(x, y) %>%
  ggplot2::ggplot(aes(x, y)) +
    ggplot2::geom_point() +
    ggplot2::theme_bw()
```



Pytho と R との変数のやり取り

行ったり来たりつつ R と Python を使いたいことがあるかもしれない．つまり `repl_python()` を使って Python に入って，`exit` で R でに戻って，また Python に入るなどである．そのときに，R と Python の変数のやりとりができる．

Python で R の `variable` という変数を取り出したいときは `r.variable`，R で Python の `variable` という変数を取り出したいときには `py$variable` とする．

```
r.variable  # R から Python へ (Python で取り出し)，variable は変数名
py$variable # Python から R へ (R で取り出し)
```

これで R と Python を対話的に行きつ戻りつしながら実行できる．

```
a <- "r_val"
repl_python(quiet = TRUE)
>>>r.a
## 'r_val'
>>>a = "python_val"
>>>exit
py$a
## [1] "python_val"
```

Python のコードを実行

既に Python の関数やコードのファイルがある場合は，`source_python()` でファイルを読み込んで使うことができる．

```
reticulate::source_python()
```

source_python() でのコード内の関数や変数をそのまま R で使うことができる．

```
reticulate::py_run_file("script.py")
reticulate::py_run_string("x = 10")
```

py_run_file() や py_run_string() でのコード内の変数や関数を R で使う場合は，py$val のように py$ の後ろに変数や関数の名前を付ける必要がある．

コラム：R と Pytho での用語の違い

Python でのモジュールとはファイル (*.py) のことで，モジュールをまとめたものがパッケージ，パッケージをまとめたものがライブラリである．このライブラリを pip install でインストール，import() でインポートしている．つまり，R でのパッケージにあたるのが，Python でのライブラリである．

```
  # USB 取り出し用の画像のトリミング (作業済)
  #   トリミングをしたほうが画像認識がうまくいくかとおもったが，逆にダメだった
  # library(tidyverse)
  # path <-
  #   "D:/matu/work/ToDo/automater/inst/img" %>%
  #   fs::dir_ls(regexp = "png")
  # img <-
  #   path %>%
  #   purrr::map(magick::image_read) %>%
  #   purrr::map(magick::image_trim)
  # purrr::map2(img, path, magick::image_write)
  # path
```

```
  # 画面サイズの取得だが，実際は拡大をしていたりするので，ちょっと数字が違う
  # system("wmic path Win32_VideoController get VideoModeDescription,CurrentVerticalResolution,CurrentH
```

KeyboardSimulator でマウス・キーボードの自動化

KeyboardSimulator を使うと，R から直接操作しづらいくても，マウスやキーボードを用いて決まった操作をするものは自動化可能である．ただし，このパッケージは Windows 専用である．

準備

キーボードとマウスの自動操作のパッケージ KeyboardSimulator をインストールする．

```
install.packages("KeyboardSimulator")
```

```
library(tidyverse)
library(KeyboardSimulator)
library(automater)
```

KeyboardSimulator では，マウスやキーボードの操作とマウス位置の取得は可能であるものの，アイコンの画像をもとに画面上での位置の取得ができない．画像からの位置取得のために，Python とそのライブラリ Pillow と PyAutoGUI を使用する．これらのインストールがまだの場合は，[Python とそのライブラリのインストール](#install_python} を参考にして，Python，Pillow，PyAutoGUI をインストールする．

また，R から Python を使うために reticulate を呼び出しておく．

```
library(reticulate)
```

さらに，使用する Python の指定を参考に reticulate で使用する Python を指定する．automater パッケージの `find_python()` を使用すると，楽に指定ができる．インストール済の Python を探して 1 つだけのときはそのパスを出力する．複数ある場合は選択肢からユーザが選んだパスが出力されるので，そこから選ぶ．

```
automater::find_python() %>%
  reticulate::use_python()
```

なお，find_python の中身は以下のとおりである．

```
find_python
```

```
## function (select_memu = TRUE)
## {
##     os <- get_os()
##     python_path <- ifelse(os == "win", "where python", "which python") %>%
##         system(intern = TRUE) %>% fs::path()
##     if (length(python_path) > 1) {
##         if (select_memu) {
##             choice <- utils::menu(python_path, title = "Select Python path. 0: exit and return all.")
##             if (choice == 0) {
##                 return(python_path)
##             }
##         }
##         else {
##             return(python_path)
##         }
##     }
##     else {
##         choice <- 1
##     }
##     return(python_path[choice])
## }
## <bytecode: 0x00000269a7e2afc8>
## <environment: namespace:automater>
```

もちろん，以下のように自分で Python のパスを指定しても良い．

```
reticulate::use_python("your_python_path")
```

マウスの移動

マウスの移動は次のようにする．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る．マウスの位置がにもよるが，徐々にマウスが左上 (10,10 の場所) に向かって移動するのが見て取れる．`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る．`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細かな動きになる．

```
KeyboardSimulator::mouse.move(10, 10, duration = 1, step_ratio = 0.1)
KeyboardSimulator::mouse.move(100, 100, duration = 0.5)
```

マウスの位置取得

マウス操作で重要なのは画面上での位置を取得することである．

KeyboardSimulator には，マウス位置を取得する関数 `mouse.get_cursor()` がある．1 箇所だけならこの関数で十分だが，何度かマウスをクリックする作業の場合は何度も実行する必要があってちょっとめんどくさい．

```
KeyboardSimulator::mouse.get_cursor()
 ## [1] -3297  1306
```

USB の取り外しの関数を作成するときには，タスクバーと R ウインドウとの行き来が必要だが，作業中にタスクバーの操作がもとの状態に戻ってしまうことがある．そうなると，正しいマウス位置の取得が難しくなる．そこで，一定時間の間隔でマウス位置を取得する automater の関数 `mouse_record()` を利用する．関数の内容は以下のとおりである．

```
automater::mouse_record
```

```
## function (n = 5, interval = 1)
## {
##     x <- list()
##     y <- list()
##     for (i in seq(n)) {
##         if (interval < 0) {
##             user_input("Press any keys on R console")
##         }
##         else {
##             automater::sleep(interval)
##         }
##         x[[i]] <- KeyboardSimulator::mouse.get_cursor()[1]
##         y[[i]] <- KeyboardSimulator::mouse.get_cursor()[2]
##         position <- paste0(i, ": x = ", x[[i]], ", y = ", y[[i]],
##             "\n")
##         cat(position)
##     }
##     return(list(x = unlist(x), y = unlist(y)))
## }
## <bytecode: 0x00000269a73158e8>
## <environment: namespace:automater>
```

sleep() を挟んで，n 回分の位置を取得するだけの関数である．実行すると以下のような結果が得られる．位置を表示させているのが煩わしければ，関数の for ループ中の `cat(position)` を削除すれば良い．

```
position <- automater::mouse_record()
position
 ## 1: x = 680, y = 587
 ## 2: x = 162, y = 482
 ## 3: x = 94, y = 1250
 ## 4: x = 816, y = 1352
 ## 5: x = 773, y = 511
 ## $x
 ## [1] 680 162  94 816 773
 ##
 ## $y
 ## [1]  587  482 1250 1352  511
```

USB メモリの安全な取り出しの自動化

ここでは「USB の安全な取り出し」を自動化する．タスクバーにある USB のアイコンを何度かクリックして，USB を安全に取り出せるようにするものである．この作業をしなくても，USB ディスクが壊れることはほとんど無いだろうが，作業するに越したことは無い．単純な操作だが，何度もやっていると面倒くさいし，しかも老眼の人間には避けたい作業である．

マウスをどこでクリックしているのか，あらかじめ位置を取得しておく必要がある．`mouse_record()` を

使えば，マウス位置を簡単に取得できる．

```
position <- automater::mouse_record(n = 4, interval = -1)
position
 ## $x
 ## [1]  770  730  700 1040
 ## $y
 ## [1] 1900 1800 1760 1750
```

今回の場合は，以下の画像ののようなマウスの位置を予め取得しておく．

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，あらかじめ位置を取得しておき (ここは pos に保存)，最後にその位置に戻す．あとは，上で取得した位置にマウスを移動させ，順次クリックさせるだけだ．実際に USB メモリをパソコンに取り付けて，動作を確認する必要がある．マウスの移動速度が早すぎて，パソコンの描画・動作速度よりも先にクリックをしてしまう場合があるかもしれない．その場合は，sleep_sec = 1(デフォルトは 0.5 秒) のようにマウスの移動後に時間を挟む必要がある．

```
pos <- KeyboardSimulator::mouse.get_cursor()
automater::mouse_move_click( 770,1900)
automater::mouse_move_click( 730,1800)
automater::mouse_move_click( 700,1760)
automater::mouse_move_click(1040,1750)
automater::mouse_move_click(pos[1], pos[2])
```

また，アプリ・ディレクトリの瞬間起動を参考に，以下のようにすれば，上記のコードを瞬間起動ができる．

- R のコードをテキストファイル remove_usb.rsc として保存
- remove_usb.rsc をクリックして，USB メモリの安全な取り出しが実行できるか確認しておく
- パスの通ったディレクトリに，remove_usb.rsc 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 と入力すると，remove_usb.rsc が実行される (USB メモリの安全な取り出しが起動)

一旦ショートカットを登録してしまえば，あとは [Win] + [R] の「ファイル名を指定して実行」から起動できる．

頻繁に行うが他に代替するのが難しいマウス操作があれば，同様に自動化しておくと便利である．筆者は，USB メモリの取り出しの他に，wifi への接続やタスクバーに固定されているアプリの操作などを自動化している．

画像をもとにマウスを動かしてクリックする

アイコンの位置が固定されている,つまりマウスのクリック位置が一定であれば,automater::mouse_move_click(1040,175 のような定位置で良い．しかし，ウィンドウの位置が異なったり，タスクバー上での位置が変化したりして，クリックするべき位置が一定でないことがある．

クリックする対象を画像ファイルとして用意できれば，その画像をもとにクリックの位置を決定できる．R のパッケージではこれを実現できなさそうなので，Python の PyAutoGUI を利用する．

```
  # reticulate::use_python("your_python_path")
  # reticulate::use_python(automater::find_python())
pag <- reticulate::import("pyautogui")  # 呼び出し時は小文字で
```

Python に慣れていない筆者は，このコードを実行する前にライブラリの指定で大文字を使ったため何度か失敗してしまった．Python でライブラリが大文字を含んでいても，呼び出すときは小文字で指定しないといけないようだ．

以下ではディスプレイの左下部部のスクリーンショットをとって，一時フォルダに png 保存する．スクリーンショットの範囲は，ディスプレイの左下の 100*100 ピクセルである．この範囲を png として保存する．

```
region <- automater::display_corner(corner = "bottom_left", width = 100, height = 100)
screenshot <- pag$screenshot(region = region)
  # ls("package:fs")
png <- fs::path_temp("screenshot.png")
screenshot$save(png)
png
```

```
## C:/Users/matutosi/AppData/Local/Temp/RtmpkvzPrY/screenshot.png
```

コードを実行すると，一時ディレクトリに png 形式の画像が保存される．筆者の環境では次の画像が保存されていた．

以下のように pag$locateOnScreen() を使うと，画像のパスを指定することでその画像の位置を取得できる．位置は，画像の 4 つの頂点の位置なので，その中心を automater::center() で求めて，その位置へマウスを移動する．以下ではマウスの移動は pag$moveTo() で実行しているが，もちろん KeyboardSimulator::mouse.move() でも構わない．コードを実行して画像が正しく認識されると，画像の中心位置 (画面の左下から左から 50，上から 50 の位置) に一旦移動した後で，左上 (左から 10，上から 10 の位置) に移動するはずである．

```
position <- pag$locateOnScreen(png)
position <- automater::center(position)
pag$moveTo(position$x, position$y)
  # duration の指定が可能
pag$moveTo(10,10, duration = 5)
```

マウスが動かない場合は，画像認識がうまくいっていない可能性が高い．PyAutoGUI の画像認識は，ディープラーニングのような似たものを抽出するのではなく，ピクセル単位で合致するもの，つまり全く同じ画像を探し出すものである．そのため，取得したスクリーンショットの状態から変化があると認識できない．これを改善するためには，pag$locateOnScreen() で confidence = 0.6 のように画像認識の精度を設定する．完全に一致しなくても結果を返してくれるが，下げすぎると誤認識の可能性が出てくる．また，confidence を使うには，Python のライブラリである OpenCV-Python をインストールしなければならないので，以下のコードを実行しておく．

```
shell("pip install opencv-python")
position <- pag$locateOnScreen(png, confidence = 0.6)
```

さらに，grayscale = TRUE を指定すると白黒画像として認識するため若干であるが高速化する．

```
position <- pag$locateOnScreen(png, confidence = 0.6, grayscale = TRUE)
```

画像から位置を取得するとき，画面全体を検索すると時間がかかる．設定によるが，USB メモリの取り出しや wifi への接続なら画面の右下，スタートメニューなら画面の中央か左下のように，ある程度の位置が決まっている．そのため，特定の領域のみ検索すれば動作が早くなる．そこで，検索する領域を画面サイズから位置を指定する．パソコンの設定を見て手入力しても良いが，できれば自動的に取得したい．画面サイズを取得するには，display_size() を使いう．画面の 4 隅の位置を指定するには，display_size() を含んでいる display_corner() を使う．関数の実行例と中身は以下を見てほしい．

```
automater::display_size()
automater::display_size

automater::display_corner()
automater::display_corner
```

これまでの内容をもとにした画像から位置をクリックする関数として,automater には recog_image_click() がある．以下の引数が利用可能である．

- wait：TRUE では，画像が見つかるまで画像認識を 1 秒間隔で繰り返す．パソコン描画がマウスの自動化に追いつかないときには便利．
- button：クリックの左右を指定．c(“left”, “right”)
- hold：クリックの保持を指定．TRUE, FALSE

- …：オプションの引数で `pag$locateOnScreen()` に渡される
  - region：画像の検索範囲を指定．`display_corner()` で指定可能
  - grayscale：TRUE で白黒認識
  - confidence：認識精度を数値で指定 (0-1)

使い方と関数の内容は以下を見てほしい．

```
region <- automater::display_corner()
img <- "your_image_path"
automater::recog_image_click(img, pag, region = region)
automater::recog_image_click
```

まとめ

固定位置への操作は，クリックするべき位置をあらかじめ取得すれば簡単に自動化できる．クリックする位置が一定でない場合は，画像をもとにしてマウスを動かすことができる．そうすれば，まるで人間が作業しているようなこともできる．ただし，画像が完全に一致しないときには認識できないことがあるので，`grayscale` や `confidence` でうまく調整しなければならない．また，描画が追いつかないときへの対応も必要である．

(コラム) 画面サイズ取得の試行錯誤

本文では，画面サイズの取得方法する関数 `display_size()` をさらっと紹介したが，関数完成には試行錯誤があった．スマートな解決方法をすぐに思い付けば良いのだが，なかなかそうは行かない．

まず試したのは，rJava を使う方法である．rJava で画面サイズの取得は可能であるが，あまりおすすめしない．「おすすめしない」というのは理由がある．rJava パッケージはその名のとおり R から Java を利用するものだが，うまく動作しないことがあるからだ．通常のパッケージならインストールして呼び出せば，そのまますぐに使えるはずだが，rJava はうまくいかないことがある．OS にインストールされている Java のバージョンと R との関係や，パスの設定の関係でエラーが出て動かないことがある．

うまくいくか分からないが，次のコードが動けば使って画面サイズを取得できる．

```
  # C:\Users\matu>java -version
  # java version "1.8.0_361"
  # Java(TM) SE Runtime Environment (build 1.8.0_361-b09)
  # Java HotSpot(TM) 64-Bit Server VM (build 25.361-b09, mixed mode)
  # C:\Users\matu>path
  # PATH=C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Program Files (x86)\Java\jre1.8.0_
  # ユーザの環境変数
  #   C:\Program Files (x86)\Java\jre1.8.0_361\bin\client
install.packages("rJava")
library(rJava)
.jinit()
toolkit <- J("java.awt.Toolkit")
default_toolkit <- .jrcall(toolkit, "getDefaultToolkit")
dim <- .jrcall(default_toolkit, "getScreenSize")
width <- .jcall(dim, "D", "getWidth")
height <- .jcall(dim, "D", "getHeight")
```

いくつかの環境で試したが，どうも設定がよくわからない場合があった．rJava があまり良くなさそうなので，別の方法を考えた．Windows であれば，次のコードのように system() を使ってコマンドを入力して情報を得ることができる．ただし，この場合は注意が必要で，ここで取得したのは画面の解像度であって，KeyboardSimulator で指定するマウスの位置ではないことがある．高解像度の画面の場合は，画面を 125% に拡大していることがあるためだ．その場合は，ここで得た値を 1.25 で割る必要がある．

```
  #   https://stackoverflow.com/questions/7305244/how-can-i-get-the-screen-resolution-in-r
library(tidyverse)
  # cmd <- "wmic path Win32_VideoController get CurrentHorizontalResolution,CurrentVerticalResolution /
item <- c("CurrentHorizontalResolution", "CurrentVerticalResolution")
cmd <- paste0("wmic path Win32_VideoController get ", item, " /format:value")
resol <-
  cmd %>%
  purrr::map(system, intern = TRUE) %>%
  purrr::map(paste0, collapse = "") %>%
  purrr::map_chr(stringr::str_replace_all, "[A-z\\r=]+", "") %>%
  as.double()
```

rJava もダメで，system() を使っても注意が必要であり，自動化もいろいろと大変である．自動化すれば手を抜けるが，手を抜くための努力は必要だ．著者も色々と試行錯誤したが，結局たどり着いたのは簡単な方法であった．

次のように mouse.move() でマウスをありえないぐらい右下に移動して，その位置を取得する方法だ．幸いなことに，mouse.move() はありえない位置を指定してもエラーにはならず，最大限可能なところまで移動してくれる．最大限に移動した位置を取得すれば完了だ．画面の最大値とマウスの位置が 1 つずれているのは，画面の左上が [1,1] ではなく [0,0] のためだ．

```
KeyboardSimulator::mouse.move(999999,999999)
KeyboardSimulator::mouse.get_cursor()
```

以上のように，関数を 1 つ作るにも試行錯誤をともなうことがある．もちろん，考えた関数がそのままうまく動けばよいが，そうでないときも多い．出来上がった関数だけを見れば非常にスマートな解決方法を実装していても，その背景には多くの試行錯誤や失敗 (成功の糧) があるのであった．

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

rvest でスクレイピング

スクレイピング

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少ないものの，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．手動でも可能だが多大な手間と時間が必要で，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．

ここではウェブでの情報収集の方法を紹介する．具体的には気象庁のページから世界各地の気象データを入手する．情報収集には R のパッケージである rvest を用いる．

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だ．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，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．静的なページの 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，ページが遷移する．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 ...
```

ところで，手作業にはなってしまうが，ニッコウキスゲを検索した結果のページをブラウザで表示させ，その後のページの内容を確認する．ウエブページには次の規則性があることに気づく．検索結果のページには表 (table) としてデータが含まれており，その表の中の目録 No である「5259」が指定状況や地図のページの URL に含まれている．つまり，目録 No を入手すれば指定状況や地図ページの URL を生成できそうだ．

必要な情報の取得

read_html() で取得した 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．

HTML の全体を表示させたい場合は，以下のコードを実行する．とてもではないが，これを自分で解析したいとは思わないだろう．

```
 # 全体を表示させたい場合
as.character(html) %>%
  cat()
```

最初と最後の部分だけを示すと次のようになる．

```
as.character(html) %>%
  stringr::str_split("\\n") %>%
  `[[`(1) %>%
  head()
```

```
## [1] "<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.0 Transitional//EN\" \"http://www.w3.org/TR/REC-html4
## [2] "<html><body>"
## [3] "<em></em>"
## [4] ""
## [5] "<title>日本のレッドデータ検索システム</title>"
## [6] "<meta http-equiv=\"content-type\" content=\"text/html;charset=utf-8\">"
```

```
as.character(html) %>%
  stringr::str_split("\\n") %>%
  `[[`(1) %>%
  tail()
```

```
## [1] "}"               ""                "//-->"           "</script>"
## [5] "</body></html>" ""
```

幸いにして rvest には html から要素を取得するための便利な関数が用意されている．ウエブページを見ると，table にデータが入っていそうなので，html_table() でデータを抽出する．ここではスペースの節

約のため purrr::map(colnames) として列名だけ表示するとともに不要なスペースを除去している．なお，rvest::html_table(html) とすると table 全体を表示できる．

```
rvest::html_table(html) %>%
  purrr::map(colnames) %>%
  purrr::map(stringr::str_remove, " ")
```

```
## [[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"
```

列名から 6 個目の table が目録 No を含んでいそうだと考えられるので，6 つ目の表を表示させる．

```
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．ほぼ同じものなので，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．

途中にちょっと面倒な点があるので，その点だけやや詳しく補足する．separate() の sep(区切り文字) として，stringi::stri_unescape_unicode("\\u00a0") 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 データの列名を確認する．なお，str_remove_all() で不要な文字の除去をしている．

```
html %>%
  rvest::html_table() %>%
  purrr::map(colnames) %>%
  purrr::map(stringr::str_remove_all, "\\n|\\t|\\r")
```

```
## [[1]]
## [1] "X1"
##
## [[2]]
## [1] "X1"
##
## [[3]]
## [1] "X1" "X2" "X3" "X4" "X5" "X6"
##
## [[4]]
## [1] "都道府県名 " "和名"          "学名"          "RDBカテゴリ名"
## [5] "統一カテゴリ"
##
## [[5]]
```

```
##  [1] "ニッコウキスゲ学名：Hemerocallis dumortieri var. esculenta分類：  単子葉類     ユリ科 登録別名：
##  [2] "ニッコウキスゲ"
##  [3] "ニッコウキスゲ"
##  [4] "学名："
##  [5] "Hemerocallis dumortieri var. esculenta"
##  [6] "分類："
##  [7] "単子葉類     ユリ科"
##  [8] "登録別名："
##  [9] ""
## [10] "環境省カテゴリ："
## [11] "なし"
## [12] "都道府県のRDB指定状況："
## [13] ""
##
## [[6]]
## [1] "X1" "X2"
```

果たして，table の 4 番目に欲しいデータがあった．あとは，filter で不要な都道府県データを除去する．さらに，データを分析するならば整形・変換・保存などをするが，スクレイピングとしての作業はここまでとする．

```
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せて便利である．

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 `<id>` や `<class>` を含めたすべてのタグのことで，他にも `<p>`，`<div>`，`<table>` 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 と矢印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．これは，curl パッケージの `curl_download()` で簡単にできる．引数 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．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("img/", 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 を確認する．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，tag，class，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．

気象庁ではなく他のサイトの事例ではあるが，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，複数種に対しても応用可能である．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/%22 %22robots.txt%22 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．れらは polite パッケージが一般的な注意として示しているに過ぎない．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

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

参考：magick で画像編集

Rselenium でスクレイピング

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)
    ver3 の最終版は以下からダウンロード可能
    https://github.com/SeleniumHQ/selenium/releases/tag/selenium-3.150.0
    https://github.com/SeleniumHQ/selenium/releases/download/selenium-3.150.0/IEDriverServer_x64_3.150.2.zip

- ブラウザの Driver
  GoogleChrome の場合
  https://chromedriver.chromium.org/downloads
  - 注意: 自身の利用しているブラウザのドライバが必要 (バージョンも合致させる)
    ブラウザが自動 update していることがあるので，バージョンは要確認

  - 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

- R からシェルのコマンドを使う
  - Seleniumu の起動・終了

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
- 種名リストの列数の変更

スクレイピングの実行時には，適切な間隔を空ける．

- 通常は 5 秒以上を求めていることが多い

- 最低でも 1 秒は空ける

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，実務的な意味合いもある．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．

magick で画像編集

パッケージ magick は，画像編集ができる．magick は，画像編集ソフトの ImageMagick を R から使えるようにしたものである．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 は，コマンドラインで使えるので，関数 `system()` を使えば R から直接操作できる．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 もインストールする．

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

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

使い方

ImageMagick 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，白黒画像にできる．

```
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．`format` を指定しなければもとの画像形式で保存されるため，形式を変換したい場合は `format` で形式を指定する．

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

切り出し

画像を切り出しする前に，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, paste0("img/", 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 ([width]x[height]+[xpos]+[y]) 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．

```
  # automater::magick_crop() と同じ
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．それぞれ正しい設定だけを使っても良いが，面倒なので 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 パッケージの関数は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(paste0("img/", pngs)) %>%
  automater::magick_crop(c(100, 0), c(890, 560)) %>%
  as.list() %>%
  purrr::map2_chr(., paste0("img/crop1_", pngs), magick::image_write)
crop_2 <-
  magick::image_read(paste0("img/", pngs)) %>%
  automater::magick_crop(c(100, 0), c(890, 980)) %>%
```

```
  as.list() %>%
  purrr::map2_chr(., paste0("img/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 = "40" は左右に，geometry = "x40" は上下に 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)

```
  # automater::web_screenshot() と同じ
web_screenshot <- function(url, trim = TRUE, border_size = "x40", annotate_size = 20, format = "png", re
  png <- fs::file_temp(ext = "png")
  webshot::webshot(url, png)
  img <- magick::image_read(png)
  fs::file_delete(png)
  if(trim){ img <- magick::image_trim(img) }
    # add top margin (and 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 = resize) }
  return(img)
}
```

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)
```

本来は DB に接続するが，解説のための一時データを使用する．通常は DB 接続のためのユーザ ID とパスワードが必要だろう．

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

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

```
  # https://docs.aws.amazon.com/ja_jp/AmazonRDS/latest/UserGuide/CHAP_CommonTasks.Connect.html
  # con <-
  #   dbConnect(RSQLite::SQLite(),
  #   username = "matutosi@gmail.com",
  #   password = "yPXIHeGWXGmPabSiVx2LCopy",
  #   host = "database-1.cd3bluovykft.ap-southeast-2.rds.amazonaws.com",
  #   port = 5432, # PostgreSQL
  #   dbname = "database-1")
```

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

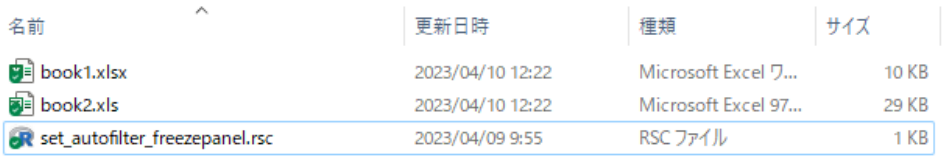
```
library(tidyverse)
```

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．

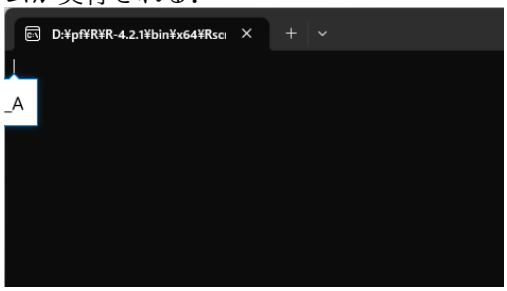
使用方法

- 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，コマンドのウィンドウで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
}
```

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，remotes::install_github() を使ってインストールする．

```
install.packages("qpdf")
install.packages("devtools")
remotes::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

余談だが，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 のコードをもとにして改良可能であり，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",
    fs::file_temp(ext = "pdf")
    )
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-4.3.1/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，combine_qpdf()(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．

```
  # automater::pdf_overlay_stamp() と同じ
pdf_overlay_stamp
```

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．

```
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)
##     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)
## }
## <bytecode: 0x000001fd495a4f68>
## <environment: namespace:automater>
```

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 <- fs::file_temp(ext = "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
```

RDCOMClient で MS Word や Excel と他形式と相互変換する

RDCOMClient は Windows に特化したパッケージであるが，これを使うと MS Word や Excel の操作が可能である．ここでは，Word および Excel を他のファイル形式に変換したり，その逆の方法を紹介する．また，操作対象の Word や Excel がインストールされている必要がある．

準備

CRAN には登録されておらず，ウェブページか GitHub からインストールする．install.packages() でインストールする場合は，バイナリなので比較的時間が早いが，R のバージョンによってはうまくインストールできないかもしれない．install_github() の場合は，コンパイルするのに少し時間がかかる．

```
  # どちらか一方でうまくいけば OK
utils::install.packages("RDCOMClient", repos = "http://www.omegahat.net/R", type = "win.binary")
remotes::install_github("omegahat/RDCOMClient")
remotes::install_github("matutosi/automater")
```

```
library(tidyverse)
library(RDCOMClient)
library(automater)
```

ドキュメントの形式変換

```
convert_docs
```

```
## function (path, format)
## {
##     if (fs::path_ext(path) == format) {
##         return(invisible(path))
##     }
##     no <- switch(format, docx = 11, pdf = 17, xps = 19, html = 20,
##         rtf = 23, txt = 25)
##     path <- normalizePath(path)
##     suppressWarnings({
##         converted <- normalizePath(path_convert(path, pre = "converted_",
##             ext = format))
```

```
##     })
##     wordApp <- RDCOMClient::COMCreate("Word.Application")
##     wordApp[["Visible"]] <- TRUE
##     wordApp[["DisplayAlerts"]] <- FALSE
##     doc <- wordApp[["Documents"]]$Open(path, ConfirmConversions = FALSE)
##     doc$SaveAs2(converted, FileFormat = no)
##     doc$close()
##     return(invisible(converted))
## }
## <bytecode: 0x000002a5a7c24800>
## <environment: namespace:automater>
```

convert_docs() の中でファイルの読み込み・保存でファイル名を指定する．その際に normalizePath() を使う必要がある．この部分を別の関数に置き換えても大丈夫かと考えて，fs::path_norm() を使ってみたところエラーになった．このように，コードを改善しようとする場合は，作業の結果としてうまく動作しないことがよくり，注意が必要である．

doc$SaveAs2() の FileFormat = no で保存形式をそれに対応する数値で指定している．このあたりは，試行錯誤の結果である．もしかしたら他の形式での保存が可能なのかもしれないが，確実に変換できるのは以下の 5 つである．

- pdf：PDF
- xps：XML Paper Specification(xml ベースファイル形式)
- html：HTML
- rtf：リッチテキスト
- txt：テキスト

なお，convert_docs() の実行時には，MS Word を起動してその機能としてファイルの読み込み・保存をする．そのため，MS Word がインストールされていないと，この関数は使えない．

複数の docx ファイルを圧縮した zip ファイルがあり，ファイル解凍，PDF への変換，1 つの PDF ファイルへの結合をするようなコードは以下のとおりである．結合時の順序はファイル名の順序に従う．そのため，PDF ファイルでのファイル順を踏まえて，docx の命名規則を決める必要がある．

- unzip() で解凍
- fs::dir_ls() でファイル名取得
- convert_docs() で形式変換
- qpdf_combine() で結合

スプレッドシートの形式変換

Microsoft365R でメールを一斉送信する

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，「田中様」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，カンマでパス (ファイル名) を区切る．絶対パス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)
```

officer で Word や Escel の内容を編集する

準備

例によってまずはパッケージをインストールする．日付のデータを扱うための lubridate とさらにそのラッパーを使うので，automater を読み込む．

```
install.packages("officer")
```

```
## -- 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
```

officer と officeverse

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．ノートパソコンでは 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，仕事上は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

```
  # コードの動作確認

  # https://ardata-fr.github.io/officeverse/index.html
  # pkg <- "D:/matu/work/ToDo/automater/R"
  # devtools::load_all(pkg)

  # fs::path_package("")
  # replacement <- readr::read_tsv("replacement.txt", header = TRUE)
  # ls("package:tibble")

library(tidyverse)
library(officer)
library(lubridate)
  # library(automater)
a <-
  read_docx() %>%
  officer::body_add_par("置換前，変換前，へんかん，ヘンカン") %>%
  print(fs::path_temp("a.docx"))
b <-
  read_docx() %>%
  officer::body_add_par("置換前，変換前，へんかん，ヘンカン") %>%
  print(fs::path_temp("b.docx"))
fs::path_dir(a) %>%
  shell.exec()
```

```
replacement <-
  tibble::tribble(
    ~file          , ~old_value, ~new_value,
    a              , "置換前"  , "置換後"  ,
    b              , "変換前"  , "変換後"  ,
    paste0(a,",",b), "へんかん", "変換"    )

replacement <-
  replacement %>%
  expand_file() %>%
  dplyr::filter(!stringr::str_detect(file, "replaced\\_"))

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，作業の手順が決まっていれば，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．日付っぽい表記のすべてを含もうとするとややこしいが，よく使う日付表記であれば網羅できるだろう．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．

```
  # Word ファイルを開く
  # 文字列の取得
  # 日付の一覧抽出
  # 日付の置換
  # Word の保存

  # library(automater)
  # library(moranajp)
library(tidyverse)
```

```r
library(officer)
library(lubridate)

  # 準備
x <-
  "23 年 1 月 1 日 (月)，2023 年 1 月 1 日 (月)，2023 年 10 月 10 日 (月)，
  2 月 2 日 (月)，12 月 22 日 (月)，2023/1/1(月)，2023/10/10(月)，
  2/2(月)，12/22(月)，23 年 1 月 1 日，2023 年 1 月 1 日，
  2023 年 10 月 10 日，2 月 2 日，12 月 22 日，2023/1/1，2023/10/10，
  2/2，12/22"
date_x <- extract_date(x)
date_doc <- print(fs::path_temp("date.docx"))

doc <- read_docx()
doc <- body_add_par(doc, " 日付置換の例")
for(d in date_x){
  doc <- body_add_par(doc, d)
  #   doc <- body_add_fpar(doc, fpar(ftext(d, fp_text(color = "red"))))
}
write_docx(doc, date_doc)
fs::path_dir(date_doc) %>%
  shell.exec()

  # 読み込み
doc <- read_docx(date_doc)


automater::format_ymd(x)

  # ls("package:officer")
  # ls("package:moranajp")
```