

R による作業の自動化・効率化 -各種パッケージの活用方法-

Toshikazu Masumura

2023-04-27

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。ニワトリかタマゴのような話は別として、できることなら、面倒な作業は自動化したい。もちろんすべての仕事を自動化できるわけでもないし、文章執筆のように作業内容によっては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としては、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方法を紹介する。

基本的に独学でここまで来たので、我流のスクリプト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

R とは

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。

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 (「%>%」や「|>」(R-4.1 以降)) を多用することが挙げられる。

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はなく、tidyverse の利用も大きい。コードが簡潔になって、コードの使い回しもし易い。パイプにはこのような多くの利点がある。

他にも「::」がやたらと出てくことや、実行速度が遅いなど欠点もそれなりにある。そもそも完璧なプログラミング言語など存在せず、それぞれが利点・欠点を持っている。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。私はこれまでちょっとだけでも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。

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はできる。例えば、R から Python を使うパッケージとして `reticulate` があり、Python から R を使うパッケージとして `rpy2` がある。つまり、1 つのプログラミング言語でしか実行できないものはほとんどなく、使いたい言語を使って勉強したい言語を勉強すれば良い。

汎用的なプログラミング言語としては、Python, C, C++, Java が、Web 関連では JavaScript が広く使われている。R の総本山である CRAN には、他にも様々なパッケージがあり、これらの言語に関連した多くの道具が揃っている。そのため、R を通してこれらの言語やそれに含まれるパッケージを利用することは可能である。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、そこから使えるものは使うのも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。例えば、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。正直なところは C で書いたというよりも、ネットで元になるコードを探して、多少アレンジしただけである。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。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 を使っているけど、他の言語のお世話になっていることは多い。R だけを勉強して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 にかぎ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。

充実したヘルプ・ドキュメント

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き、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圧力はあることは事実である。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。

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) をダウンロード。

<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。

- 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 OK. 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、英語のみの表示。

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、エラーメッセージなどを日本語で表示させることができる。「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しかも英語で表示されたら、わけがわからないからです。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のが結構ある。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だと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。

```
# 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。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 ですが、好みの問題ですので正直どちらでも大丈夫。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作り。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、それらを参照するのは便利。

- その後の設定

その他は、既定値 (そのまま) で OK。

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するには、R で簡単なコマンドを実行するだけである。

CRAN から

CRAN は、R の総本山である。

<https://cran.r-project.org/>

R 本体だけでなく、各種パッケージが公開されている。

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

CRAN に掲載されており、パッケージの名前がわかっていたら、以下のようにすればインストールできる。

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

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

GitHub から

たいていは CRAN に登録されているが、GitHub にしかないパッケージもある。その場合には、以下のようにする。

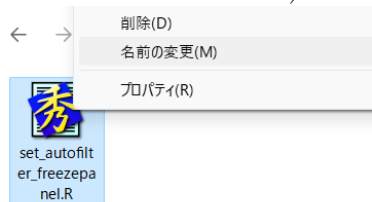
```
install.packages("devtools")
devtools::install_github("matutosi/ecan")
```

スクリプトの関連付け

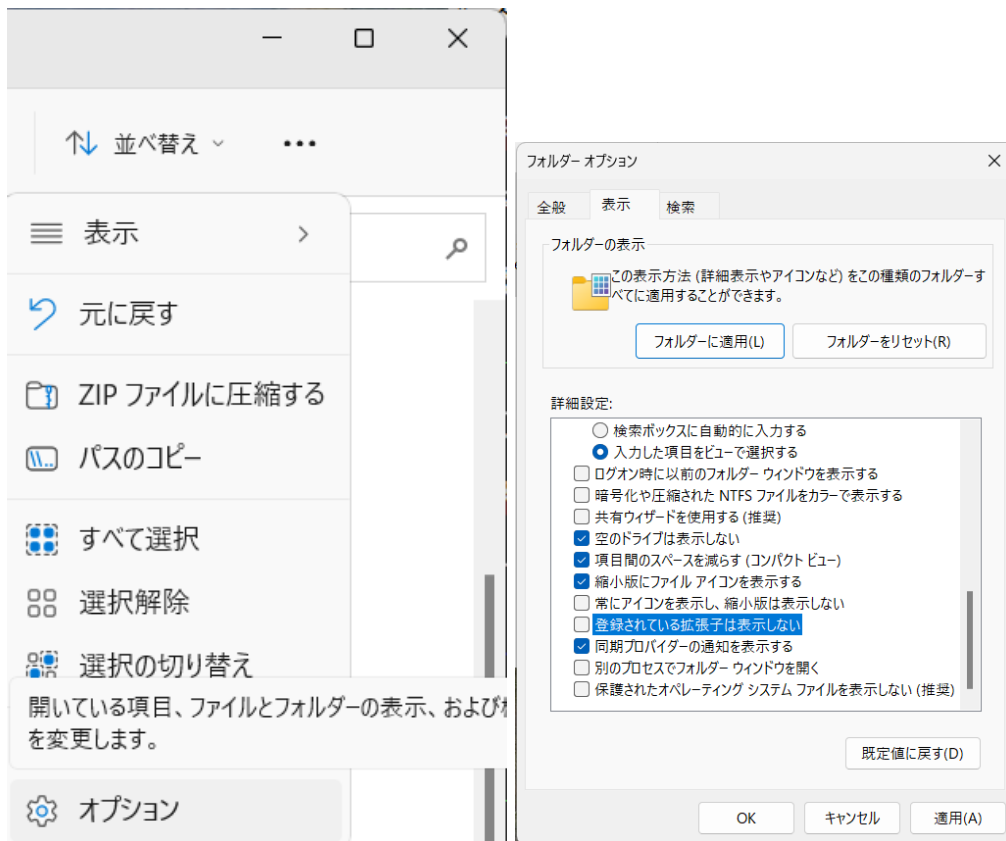
R のプログラムのファイルは拡張子「script.R」のように「R」という拡張子を付けて保存することが多い。「.docx」をワードで、「.xlsx」をエクセルで開くのに同様に、私は「*.R」をテキストエディタで開くように設定している。その後、開いたファイルを R のコンソールに貼り付けて、プログラムを実行する。

このような使い方でももちろん良いのだが、内容を変更しないのであれば面倒臭い。つまり、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。プログラムを R に関連付けれることで、これが実現できる。

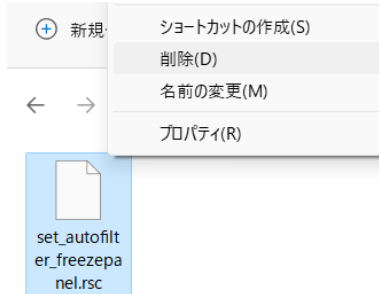
1. プログラムのファイル名を「.R」から「.scr」に変更する (「scr」は大文字小文字は関係なく、「Rsc」や「RSC」などでも OK)。



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。



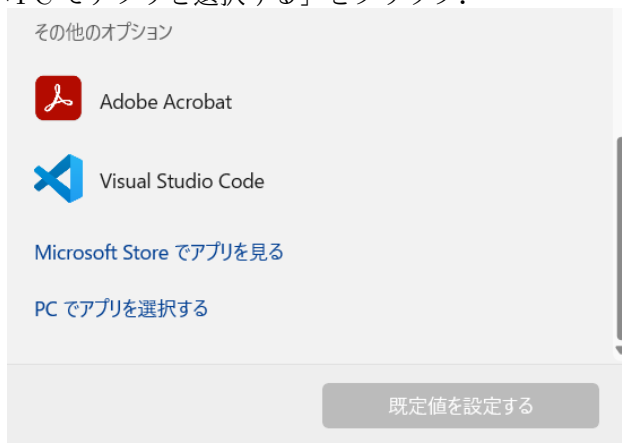
3. ファイルを右クリックして、「プロパティ」を選択。



4. 「全般」タブのやや上にあるプログラムの「変更」を選択。



5. 「PC でアプリを選択する」をクリック。

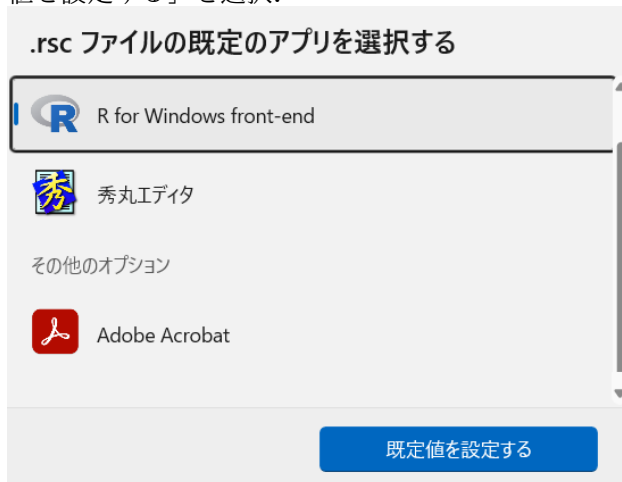


6. ファイル選択画面で、R をインストールしたフォルダまで辿っていき (「c:\Program files\R\R-4.2.3\bin\x64」など), 「Rscript.exe」を選択する。

> PC > Windows (C:) > Program Files > R > R-4.2.3 > bin > x64

名前	更新日時	種類	サイズ
Rterm.exe	2023/03/15 14:52	アプリケーション	88 KB
RSetReg.exe	2023/03/15 14:52	アプリケーション	88 KB
Rscript.exe	2023/03/15 14:52	アプリケーション	92 KB
Rgui.exe	2023/03/15 14:52	アプリケーション	86 KB
Rfe.exe	2023/03/15 14:52	アプリケーション	104 KB
Rcmd.exe	2023/03/15 14:52	アプリケーション	102 KB
R.exe	2023/03/15 14:52	アプリケーション	103 KB
open.exe	2023/03/15 14:52	アプリケーション	17 KB

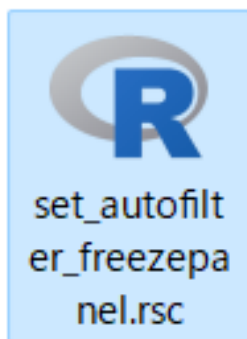
7. 「.rsc ファイルの既定のアプリを選択する」で「R for windows front-end」が表示されるので、「既定値を設定する」を選択。



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。



9. ファイルのアイコンが R のアイコンになったら OK.



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).

magrittr でコードを簡潔に

パッケージ magrittr はちょっと変わっている.

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(`%>%`)である点もちょっと変わっている。ただし、パイプ以外にも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、`set_colnames()` はデータフレームの列名を変更する時に便利だ。パイプを使ったコードの途中で列名を変更するために、`<- colnames()` でコードを区切るのは面倒である。`# <- colnames` できる? `# [<-` また、`set_colnames()` 以外にも、`dplyr` の `rename()` や `select()` で列名を変更する方法もある。

```
hoge <- colnames(c("foo", "bar"))
hoge %>%
  magrittr::set_colnames(c("foo", "bar")) %>%
  dplyr::filter(...)
```

`magrittr` に含まれる関数たちで、どんな内容か気になるものの一覧

```
export("n'est pas")
export(add)
export(and)
export(equals)
export(not)
export(or)
export(pipe_nested)
export(set_colnames)
export(use_series)
```

tidyverse と magrittr

`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。そこで、R を起動時に `tidyverse` を読み込む人は多いだろう。なお、`tidyverse` は 1 つのパッケージではなく、複数のパッケージからなるパッケージ群である。

```
library(tidyverse)
```

`tidyverse` のパッケージ群を読み込んだときや、そのうちの個別のパッケージ (`forcats`, `tibble`, `stringr`, `dplyr`, `tidyr`, `purrr` など) を読み込むと、`%>%` (パイプ) を使うことができる。私は `%>%` が `tidyverse` の独自のものと勘違いをしていた。しかし、`%>%` はもとはパッケージ `magrittr` の関数であり、そこからインポートされている。そのため、`tidyverse` を読み込むと使うことができる。`%>%` 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。

`tidyverse` の関数では、引数とするオブジェクトが統一されている。具体的には、第 1 引数のオブジェクトがデータフレームや `tibble` になっていることが多い。そのため、パイプと相性が特に良い。

`%>%` とその仲間

`%>%` の仲間としては、以下の関数もある。

- `%<>%`
- `%T>%`
- `%%$%`

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。`%>%` と同じように使うことができるが、役割がそれぞれ違う。

```
library(magrittr)
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:purrr':
##
##   set_names
## The following object is masked from 'package:tidyr':
##
##   extract
```

```
%<>%
```

%<>% 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)) %>%
  head() %>%
  print()
```

```
## # A tibble: 6 x 12
##   manufacturer model      displ  year   cyl trans1 trans2 drv   cty   hwy fl   class
##   <chr>          <chr>    <dbl> <int> <int> <chr> <chr> <chr> <int> <int> <chr>
## 1 audi          a4      1.8  1999     4 auto    15    f      18    29 p
## 2 audi          a4      1.8  1999     4 manual  m5    f      21    29 p
## 3 audi          a4      2.8  1999     6 auto    15    f      16    26 p
## 4 audi          a4      2.8  1999     6 manual  m5    f      18    26 p
## 5 audi          a4 quatt~  1.8  1999     4 manual  m5    4      18    26 p
## 6 audi          a4 quatt~  1.8  1999     4 auto    15    4      16    25 p
## # i 1 more variable: class <chr>
```

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

```
## # A tibble: 6 x 12
##   manufacturer model      displ  year   cyl trans1 trans2 drv   cty   hwy fl   class
##   <chr>          <chr>    <dbl> <int> <int> <chr> <chr> <chr> <int> <int> <chr>
## 1 audi          a4      1.8  1999     4 auto    15    f      18    29 p
## 2 audi          a4      1.8  1999     4 manual  m5    f      21    29 p
## 3 audi          a4      2.8  1999     6 auto    15    f      16    26 p
```

```
## 4 audi          a4          2.8 1999      6 manual m5      f          18      26 p
## 5 audi          a4 quatt~    1.8 1999      4 manual m5      4          18      26 p
## 6 audi          a4 quatt~    1.8 1999      4 auto    15      4          16      25 p
## # i 1 more variable: class <chr>
```

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。コードを短くできるのは1行だけで、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>%

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、「{ }」で囲う - 処理終了後に「%>%」が必要

例のコードを示す

```
# mpg %T>%
# {
#   tmp <<- dplyr::select(., )
# } %>%
```

%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。

%%

%% 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。R CMD CHECK(???) の possible problem で Warning が出力され、そのままではCRANでは受け付けてもらえない(たぶん)。automater のように 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%% や%>% をインポートしておくと、パッケージの中でこれらを使える。%>% だけなら、usethis::use_pipe() とすれば、開発パッケージの DESCRIPTION に、importFrom(magrittr, "%>%") を書いてくれる。

```
importFrom(magrittr,"%>%")
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、`.` を明示する必要がない。

```
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
```

stringr で文字列操作

はじめに

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

stringr には、

stringr と base

base パッケージ

stringr パッケージ

準備

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

```
library(stringr)
```

stringr の関数

stringr の利点

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

ggplot2 で楽に綺麗に作図

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 graphics システムがそれに当たる。すごく便利なものとは当時は考えていた。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ることで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 ggplot2 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。base の作図環境とは異なり、統一的なインターフェースを持っており、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、などなど各種の plot が可能
- 図が綺麗
- テーマの変更も簡単
- facet によるグループ分けも便利
- 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

ggplot2 の基本

iris を例にするが、できれば, vegan とか dave のデータを使う tidy data への変換が必要コードのみか, 詳しくは松村や比嘉の解説を参考に

gather() と spread() は pivot_longer() と pivot_wider() になって使いやすくなった. Hadley 自身も使い方を混乱していたらしい

aesthetics

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

facet を使おう

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()

shiny

shiny は必要? R だけでウェブアプリが作れる reactive の考え方を覚える必要あり

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
- unwind GDA
- チートシート

fs でファイル操作

はじめに

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 ならシェルを使えば、各種ファイル操作をコマンドラインで実行できる。もちろん、マウスを使った操作でも構わないが、多くのファイルでの名前の変更やファイル名によるフォルダの振り分けなら、マウス操作よりもコマンドを使った操作が早いし確実である。なお、Windows の場合は [Win] + [R] で「ファイル名を指定して実行」で「cmd」と入力すれば、

コマンドプロンプトやバッチファイル (あるいはシェルスクリプト) などでの操作に慣れていれば、それが便利である。ただ、dos コマンドの変数の扱いは、慣れていないと結構難しい (慣れていても?)。そんなときは、R の関数 (`shell()`, `system()`) を使って、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るならば、`shell()` などを使うのは良い方法である。

また、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 の関数群の中には名前が分かりにくい点や引数の一貫性が無い点などの難点がある。これは、R が発展していく中で徐々に関数が追加されたことによるようだ。

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

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ない。いちいち個別のものを覚えるよりも、fs パッケージの関数を覚えておけば、どの OS であろうが同じように動作してくれるので楽ができる。

なお、fs, base, shell の詳細な比較が、以下の URL にあるので、参照してほしい。

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

shell, base パッケージ, fs パッケージ

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

shell を使う

shell なら、以下のようなコマンドだ。dos コマンドの変数やループなどを駆使すると、もっと短く書けるのかもしれないが、残念ながらそのような知識がない。テキストファイルで書いてもそれほど時間がからないだろうが、ファイル数が多くなれば大変だ。

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

base パッケージ

基本的に tidyverse の関数群を使わず、できるだけ R の標準の関数を使った例を示す。sprintf() は使い慣れていないと、どのように指定するべきか分かりにくい。

```
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・検索などを行うことが多いので、stringr が役立つ。stringr パッケージの関数は、str_ の名前になっているため、覚

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

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

準備

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

```
library(fs)
```

fs の関数群

パス操作 (`path_`)、ディレクトリ操作 (`dir_`)、ファイル操作 (`file_*`) の関数群に分けることができる。パス操作には、base や shell にはない機能が多くあって、使いやすい。拡張子を取り除く関数を自作したことがあるが、同じような関数 (しかもおそらく、fs のほうがしっかりしている) 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。

fs, base, shell の比較は次の URL を参照して欲しい。

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

パス操作

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

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

ディレクトリ操作

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


```

dir_ls("path") # 一覧
dir_info("path") # 情報
dir_copy("path", "new-path") # 複写
dir_create("path") # 作成
dir_delete("path") # 削除
dir_exists("path") # 有無確認
dir_move() (see file_move) # 移動
dir_map("path", fun) # 複数処理
dir_tree("path") # ツリー表示

```

ファイル操作

ファイル操作は shell や base とそれほど変わらない感じがする.

```

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

```

fs を使ったファイル操作例

ごく個人的なことだが、R のバージョンアップ時には Rconsole と RProfile.site を古いバージョンから複製して、カスタマイズした設定を引き継いでいる。バージョンアップをそれほど頻繁にしないのであれば、手作業でコピーしてもそれほど問題はない。普通の R ユーザなら常に最新版を使わなくても良い。ただ、R 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
library(fs)
library(magrittr)
wd <-
  path_package("base") %>%
  path_split() %>%
  unlist() %>%
  .[-c((length(.) - 2):length(.))] %>%
  path_join()
setwd(wd)
dir <- dir_ls()
d_old <- dir[length(dir)-1]
d_new <- dir[length(dir)]
files <- c("Rconsole", "Rprofile.site")
f_old <- path(d_old, "etc", files)
f_new <- path(d_new, "etc")

```

```
file_copy(f_old, f_new, overwrite = TRUE)
```

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。

shell

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

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

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

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

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

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

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

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

ImageMagick 画像変換など

準備

Python のスクリプト実行

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

rvest でスクレイピング

スクレイピング

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

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

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

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

rvest と RSelenium

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

rvest のできること

- HTML の取得
- DOM の取得: `id`, `class`, `tagName` などを用いる
- table の取得
 - HTML 内の取得したいデータは table にあることが多いため、非常に便利そもそも、table でないデータを取得するのは非常に不便
- リンクの取得
 - ページ遷移に使用する
- `stringr` と組み合わせて使うと良い
- 文字コードの変換には `stringi` を用いる
- `tidyverse` や `magrittr` との合せ技が便利

- Form の入力・選択 radio ボタンはちょっと工夫が必要 `moranaip::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、インデックスがずれるので結構厄介 `polite` パッケージとの連携

準備

```
# install.packages("rvest")
# library(rvest)
```

HTML の取得

使い方

注意点

DOM の取得

使い方

注意点

Rselenium

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

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

準備

- RSelenium: CRAN からインストール
- Selenium: 本家サイトからインストール
 - 注意: `ver3.xxx` をインストールする
`ver4.0` 以上は `RSeleniumu` が対応していない (Python なら可)
- ChromeDriver
 - 注意: 自身の利用しているブラウザのドライバが必要 (バージョンも合致する必要あり)
GoogleChrome は自動的に update されるので、バージョンをよく確認する通常は、安定版の最新版で大丈夫である
 - Selenium と同じフォルダに保存する

ブラウザの自動化

使い方

注意点

要素の取得

id がわかるとき `document.getElementById()`

`xpath document.selectQueryAll()` 動的にサイトが作られているときには、変化する可能性があるので注意
使用されている JavaScript の関数がわかる `script <- "" rem$execute(script)`

例 BiSS の文字サイズの変更主命リストの列数の変更

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

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

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

その他

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

reticulate

RとPythonのパッケージは、相互に移植されていることが多い。例えば、Pythonのlogging(とRのfutile.logger)をもとにRのパッケージloggerは開発されている。 <https://cran.r-project.org/web/packages/logger/index.html> また、Rのggplot2やdplyrはPythonにも移植されている。

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

準備：Pythonのインストール

準備：Pythonでのモジュール(パッケージ)のインストール

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

py_install() や conda_install() でパッケージがインストールできないとき - pipでパッケージをインストール

- pipでインストールできたpythonをreticulate::use_python()で指定

準備 - Pythonのインストール

- pdf2docxのインストール

```
pip install pdf2docx
```

実行

```
# pdf2docxの読み込みでエラーになるとき
#   reticulate::use_python()でpythonを指定
#   pipでpdf2docxがインストールできたpythonを使う
library(reticulate)
# reticulate::py_install("pdf2docx") エラー
# https://anaconda.org/conda-forge/python-docx
```

```
# reticulate::conda_install(channel = "conda-forge", packages = "python-docx") # できたけど, pdf2docx
reticulate::use_python("C:/Python/Python39/python.exe")
reticulate::py_run_string("from pdf2docx import parse")
reticulate::py_run_string("pdf_file = 'D:/a.pdf'")
reticulate::py_run_string("docx_file = 'D:/a.docx'")
reticulate::py_run_string("parse(pdf_file, docx_file)")
```

Pytho と R との変数のやり取り

variable は変数名

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

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

DBI でデータ取得

データベースとの連携

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

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

```
## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
## v dplyr      1.1.1      v readr      2.1.4
## v forcats    1.0.0      v stringr   1.5.0
## v ggplot2    3.4.1      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
```

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

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

使い方

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

```
## # A tibble: 81 x 4
##   year model      displ  cyl
##   <int> <chr>    <dbl> <int>
## 1 1999 a4         1.8     4
## 2 1999 a4         1.8     4
## 3 2008 a4          2     4
## 4 2008 a4          2     4
## 5 1999 a4 quattro  1.8     4
## 6 1999 a4 quattro  1.8     4
## 7 2008 a4 quattro  2     4
## 8 2008 a4 quattro  2     4
## 9 1999 malibu    2.4     4
## 10 2008 malibu    2.4     4
## # i 71 more rows
```

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

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

```
## 8 audi      a4 quattro  1.8  1999    4 manu~ 4      18    26 p    comp~
## 9 audi      a4 quattro  1.8  1999    4 auto~ 4      16    25 p    comp~
## 10 audi     a4 quattro  2     2008    4 manu~ 4      20    28 p    comp~
## # i 224 more rows

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

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

その他、DBI パッケージの詳細は以下を参照。

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

xlsx でエクセル操作




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

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

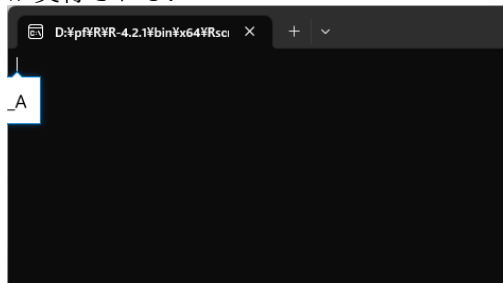
- 準備：R のインストール
- 準備：set_autofilter_freezepanel.rsc をダウンロード (右クリックして「名前を付けてリンク先を保存」) して、任意のフォルダに保存。
- 準備：スクリプトの関連付けを参考にして、「.rsc」を「Rscript.exe」に関連付けする (Windows の場合)。Mac の場合は、Mac - 拡張子に関連付けられているアプリを変更する方法などを参考にしてほしい。
- set_autofilter_freezepanel.rsc と同じフォルダに、処理したいエクセルのファイルを保存。

名前	更新日時	種類	サイズ
 book1.xlsx	2023/04/10 12:22	Microsoft Excel ワ...	10 KB
 book2.xls	2023/04/10 12:22	Microsoft Excel 97...	29 KB
 set_autofilter_freezepanel.rsc	2023/04/09 9:55	RSC ファイル	1 KB

- 実行前のエクセルのファイル



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



プログラムが自動的にエクセルのファイルの 1 行目の A 列から Z 列までにオートフィルタを設定し、1 行目と 1 列目でウィンドウ枠を固定する。複数ファイル・複数シートにも対応している。

なお、初回実行時は、xlsx パッ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)   # ワークブックの保存
}

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

```

その他の操作 (未作成)

qpdf で PDF 操作

関数一覧

```

library(qpdf)
# ページ数取得, show the number of pages in a pdf
pdf_length(input, password = "")
# 1ページごとに分割, split a single pdf into separate files, one for each page
pdf_split(input, output = NULL, password = "")
# 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 = "")
# 結合, join several pdf files into one
pdf_combine(input, output = NULL, password = "")
# 圧縮, compress or linearize a pdf file
pdf_compress(input, output = NULL, linearize = FALSE, password = "")
# ページ回転, rotate selected pages
pdf_rotate_pages(input, pages, angle = 90, relative = FALSE, output = NULL, password = "")
# 重ね合わせ
pdf_overlay_stamp(input, stamp, output = NULL, password = "")

input <- ""
pdf_split(input, output = "d:/", password = "")

```

pdf を docx に変換

RDCOMClient

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

インストール

```

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

```

変換実行

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

```

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

```

ラッパー関数

```

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

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

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

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

```

pdf2docx

Microsoft365R

<https://cran.r-project.org/web/packages/Microsoft365R> # Outlook の使い方 # <https://cran.r-project.org/web/packages/Microsoft365R/vignettes/outlook.html>

Outlook で複数メール送信を一斉送信

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

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

そのようなとき、いちいちメールを作成・編集していると面倒だし、間違いのもとになる。名前を中途半端に修正して、3箇所のうち1箇所だけ別の人の名前にしてしまっていたり、日付と曜日があっていないなどの間違いは日常茶飯事だ。このような間違いをなくするには、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(Microsoft365R)
# 会社など組織で契約している場合
Microsoft365R::get_business_outlook()
# 個人利用の場合
# Microsoft365R::get_personal_outlook()
```

とりあえず使う

まずは、試しにメールを作って送ってみる。

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

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

# メール送信
em$send()

# outlook の下書きフォルダからメールを取り出す
```

```

drafts <- outlook$get_drafts()$list_emails()
# 下書きフォルダのメール一覧
drafts
# 下書きフォルダのメールの 1 つ目を送信
drafts[[1]]$send()

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

```

メールの一斉送信

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

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

宛先が入力されていないとメールは送信できない。CC と BCC は任意。

件名と本文はなくても送信できるが, 両方とも何もないとメールの意味がない。

添付ファイルがあれば, ファイル名を指定。複数ファイルを添付するときは, カンマで path(ファイル名) を区切る。絶対 path で指定すると間違いは少ない。

```

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

# 関数の読み込み
source("https://gist.githubusercontent.com/matutosi/bed00135698c8e3d2c49ef08d12eef9c/raw/6acc2de844eeea")

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)

```