data-handling

TOPICS

はじめに

今回のチュートリアルの目的

1. jsPsychを使った心理実験・調査を作成できるようになる（7/21）済
2. jsPsych心理実験・調査で得られたデータをR, Pythonで整頓できるようになる（7/28）

なぜR, Pythonを使うのか

手作業でデータ処理をするのが大変だからです。 参加者ごとに別々に保存されたデータファイルをまとめるのも大変。 必要なデータの抽出するのも大変。

もちろん，R, Python以外のプログラミング言語でもいいと思いますが，以下の理由で今回はこの2つの言語を採用しています。

- 作成者がこの2つでしがデータ整理をしたことがない
- ウェブで日本語の資料が見つかりやすい

今回使用するライブラリ（R）・パッケージ（Py）

R

```
if(!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse, here, jsonlite)
```

使用するのは，`pacman`, `tidyverse`, `here`, `jsonlite` の4つです。 それぞれの概要は以下のとおりです。

- `pacman`: パッケージ管理ツール。`p_load` 関数が便利。基本は `library()` と同じだが， もしパッケージがインストールされてなければ `install.packages()` してくれる。`()`内にパッケージ名を並べられるのも良い。
- `tidyverse`: データ整理・分析用のツール（群）
  - 実際に使うのは，そのうちの `dplyr`, `tidyr`, `readr`, `purrr`, `fs`
  - それぞれの具体的な棲み分けは このサイト などを参照してください
- `here`: `.Rproj` ファイルを起点にパスを指定できるようにする
- `jsonlite`: json形式のデータ（`survey-likert`が吐き出すデータ）を扱いやすくする。

Python

```
import pathlib
import json
import pandas as pd
```

3つのパッケージを使用します。

- `pathlib`: ファイルパスをいい感じに指定する（組み込みパッケージ）
- `json`: json形式のデータをいい感じに扱う（組み込みパッケージ）
- `pandas`: データ整理・分析用（要インストール）

データの読み込み

今回は以下のように `exp/data/exp1/` フォルダに参加者ごとにデータが保存されています。

```
exp/data/
└── exp1
    ├── 0pu18xqt8ws6_flanker-BB_exp1.csv
    ├── 1f96wckn5emk_flanker-BB_exp1.csv
    # 省略
    ├── pe7l2asv0cml_flanker-BB_exp1.csv
    └── qf7aafx8dmog_flanker-BB_exp1.csv
```

まずは1つ

まずはcsvファイルを一つ読み込んでみて，少し離れたフォルダに保存されたファイルを読み込む方法に触れてみましょう。

R

`readr::read_csv()` を使ってcsvファイルを読み込みます[1]。 引数に読み込みたいファイルのパスを指定して実行するとデータフレームとして読み込まれます。

```
readr::read_csv(
  here::here("exp/data/exp1/0pu18xqt8ws6_flanker-BB_exp1.csv"),
  na = c("", "NA", "null")
)
```

ファイルパスの指定に `here::here()` を使用しています。 `here::here()`は（たぶん）最寄りの`.Rproj`ファイルがあるディレクトリを返します。 引数にさらにパスを指定すると，`.Rproj` ファイルがあるディレクトリを起点としてパスにすることができます。 そのため，`.Rproj` ファイルさえ作っておけば，`here::here()` を組み合わせることで，作業ディレクトリに依存せずファイルパスを指定できるようになります。

2つ目に指定している `na = c("", "NA", "null")` という引数は，csvファイル内の何を `NA` （欠損値）として認識するかを指定しています。 jsPsychが吐き出すファイルにそれなりに含まれている `null` を`read_csv` はデフォルトで， `NA` として認識しないので，このような指定をしています。

ちなみに， `readr::read_csv()` とか `here::here()` は `ライブラリ名::関数名` というライブラリ名を明示できる記法になっています。 この記法を使うと，プログラムを実行する前にライブラリを読み込む必要がなくなったり，同名の関数が複数のライブラリに存在する場合に生じる誤動作を防ぐことができたり，といった恩恵があります。 今回はそういうケースに当てはまらないので，`read_csv()`, `here()` だけでも同じ動作をします。 ただし，今回の資料では，前者の書き方をします。 プログラム実行上のメリットではないのですが，シンプルにライブラリ名がわかりやすくて検索しやすくなったりするという利点があります。

Python

`pandas.read_csv()` を使ってcsvファイルを読み込みます。 引数に読み込みたいファイルパスを指定します。

```
dir_root = pathlib.Path(__file__).parent.parent.parent # 分析ファイルの保存先によって変わる
dir_data_exp1 = dir_root / 'exp' / 'data' / 'exp1'
pd.read_csv(dir_data_exp1 / '0pu18xqt8ws6_flanker-BB_exp1.csv')
```

`pathlib` は `/` で直感的に，かつOSに依存せずパスを指定できるので便利です。 `.Rproj` + `here` のようなものはないので，最初に，今開いているファイルのパス（`pathlib.Path(__file__)`）を取得し，そこからたどってプロジェクトのルートディレクトリを取得し（今回であれば，`parent.parent.parent` と3つ上のディレクトリ）変数に格納しています。

今回は，`exp1`のデータが保存されたディレクトリのパスも変数に格納しています。

全部まとめて読み込む

上のコードを参加者分だけコピペして読み込むのはさすがに非効率的です。 第1回でBIS/BASの質問項目の設定を `for` 文でまとめて処理したのと同様に,

- まとめてファイルパスを取得し
- `for` 文などで読み込む処理を繰り返し
- それぞれのデータフレームを1つに結合する

とすると，うまく関数や組み込みの構文を使えば，Rでは2行，Pythonでは1行ですべてのデータを一つのデータフレームとして読み込むことができます。

R

- まとめてファイルパスを取得する

`fs::dir_ls()` を使います。 引数に，データが保存されたフォルダのパスとファイルの拡張子をそれぞれ指定すると，データが保存された一連のcsvファイルのパスがベクトル形式が得られます。

```
fs::dir_ls(path = here::here("exp/data/exp1"), glob = "*.csv")

# Macなら以下のような出力（3つ目以降省略
# /..プロジェクトフォルダまでのパス../jspsych-tutorial-20210721-
28/exp/data/exp1/0pu18xqt8ws6_flanker-BB_exp1.csv
# /..プロジェクトフォルダまでのパス../jspsych-tutorial-20210721-
28/exp/data/exp1/1f96wckn5emk_flanker-BB_exp1.csv
# /..プロジェクトフォルダまでのパス../jspsych-tutorial-20210721-
28/exp/data/exp1/31h414f13x6x_flanker-BB_exp1.csv
```

- `for` 文などで読み込む処理を繰り返す

得られたファイルパスのベクトルの要素それぞれに対して，`readr::read_csv` を実行します。 `for` 文を使ってもいいのですが，ここでは `purrr::map()` という関数を使います[2]。

`purrr::map` はベクトルやリストを第1引数にとり，第2引数に関数を指定します。 第3引数以降は関数の第1引数以外の引数を指定します。 実行結果はリストが返されます。

```
purrr::map(1:3, log, base = 2) # 1,2,3の底2のlogを計算
# [[1]]
# [1] 0

# [[2]]
# [1] 1

# [[3]]
# [1] 1.584963
```

これをファイルパスのベクトルに応用します。

```
fs::dir_ls(path = here::here("exp/data/exp1"), glob = "*.csv") %>%
  purrr::map(readr::read_csv, na = c("", "NA", "null"))
```

これを実行すると，各csvを読み込んだデータフレームのリストが得られます。

なお，この `%>%` はパイプ演算子と呼ばれるもので，左の項（改行している場合は上の項）の計算結果を右の項（同じく下の項）の第1引数に代入するという演算子です。 詳しくは このサイト を参照してください。

- データフレームのリストを1つのデータフレームに結合する

データフレームのリストは`dplyr::bind_rows()` で縦に連結できます。 以下のようにするだけでOK。

```
fs::dir_ls(path = here::here("exp/data/exp1"), glob = "*.csv") %>%
  purrr::map(readr::read_csv, na = c("", "NA", "null")) %>%
  dplyr::bind_rows()
```

なお，それぞれのファイルを読み込んでデータフレームを結合するという処理は `purrr::map_dfr` という関数でまとめて実行できる。

```
fs::dir_ls(path = here::here("exp/data/exp1"), glob = "*.csv") %>%
  purrr::map_dfr(readr::read_csv, na = c("", "NA", "null"))
```

こういう関数が用意されているのは，同じフォーマットのファイルが多数あって，それをそれぞれ読み込んでまとめるというケースが多いということなのかもしれません。

Python

Pythonならリスト内包表記という文法があるおかげで一行で書けます。 やっていることはRと同じです。

```
df_e1 = pd.concat([pd.read_csv(f) for f in dir_data_exp1.glob("*.csv")])
```

リスト内包表記を使わなかったら以下のようになります。

```
list_df = []
for f in dir_data_exp1.glob('*.csv'):
    list_df.append(pd.read_csv(f))

df_e1 = pd.concat(list_df)
```

- まとめてファイルパスを取得する

`.glob()`は`Path`オブジェクトのメソッドで，そのオブジェクトがディレクトリパスを指している場合，フォルダ内のファイルの一覧をリストとして取得する[3]。 引数に`'*.csv'`と指定すればcsvファイ

ルだけの一覧を作る[4]。

- `for` 文などで読み込む処理を繰り返す

まさにこの通りで，for文を使って，ファイルパスのリストから要素を一つずつ取り出して，csvファイルの読み込み `pd.read_csv()` → 別のリストに格納 `list_df.append()` という処理を繰り返します。

- 1つのデータフレームに結合する

`pd.concat()` でリストに格納されたデータフレームを縦方向に結合できます。

データの抽出・整形

すべての参加者のデータを一つのデータフレームにまとめることができました。 あとは，分析の対象となるデータを必要に応じて取り出して，分析すればよいです。

参加者情報

デモ実験では，参加者ID，年齢，性別を `jsPsych.addProperties()` という関数で追加しています。 `tempID`, `age`, `sex` という名前の列にそれぞれ保存されているので，それらの列を抽出すればよいです。 ただ，`jsPsych.addProperties()` はデータのすべての行に同じ情報を保存するので，同じ情報がかなり重複しています。 そのため，重複をなくす処理も合わせて実行する必要があります。

R

`dplyr::distinct` という関数が便利です。。 引数に列名を指定すると，列の選択と重複の削除を同時にしてくれます。

```
df_subj_e1 <- df_e1 %>%
  dplyr::distinct(tempID, age, sex)
```

Python

上記の処理を`.loc` で列を選択し，`.drop_duplicates` で重複を削除します。

```
df_subj_e1 = (
    df_e1
    .loc[:, ["tempID", "age", "sex"]]
    .drop_duplicates()
)
```

ちなみに，以下のように書くこともできますが，Rでの場合と同じように改行したほうが（個人的には）見やすいし，各行にコメントも挿入できるので，上のように書くのが好きです。 メソッドチェーンを改行する場合は，全体を`()` に入れましょう。

```
df_subj_e1 = df_e1.loc[:, ["tempID", "age", "sex"]].drop_duplicates()
```

フランカー課題

フランカー課題のメインの試行を分析時に取り出せるように，デモ実験では，`task`という列に`"flanker_main"`という文字列が保存されるようになっています。 したがって，`task` の列が `flanker_main` 行を取り出せば，フランカー課題本番のデータだけになります。

それに合わせて，分析に必要な列だけを残すようにしておくと，計算途中の結果を表示するときなどに見やすくなります。 今回のフランカー課題で，分析に使いそうなデータが入っている列は `condition`, `rt`, `correct` です。

R

`dplyr::filter()` を使って，フランカー課題の行を取り出します。 引数にある列に関する条件式を指定すると，条件式が `TRUE` となる行を返してくれます。 2つ以上の条件を指定することも可能で，その場合，複数の列それぞれに対して別々の条件を指定することも可能です。

`dplyr::select()` を使えば，一部の列だけを取り出せます。 取り出したい列を引数に指定します。

```
df_flanker_e1 <- df_e1 %>%
  dplyr::filter(task == "flanker_main") %>%
  dplyr::select(tempID, condition, rt, correct)
```

Python

`.query()` メソッドを使えば，特定の行を取り出せます。 `dplyr::filter()` と同じように，引数にある列に関する条件式を指定すると，条件式が `True` となる行を返してくれます。 ただし，条件式は文字列で指定する必要があります。

```
df_flanker_e1 = (
    df_e1
    .query('task == "flanker_main"')
    .loc[:, ["tempID", "condition", "rt", "correct"]]
)
```

BIS/BAS

BIS/BASのデータの抽出はフランカー課題のときと同じ処理をすればよいです。

しかし，`survey-lickert` で収集されたデータはjson形式の文字列になっているので，分析するために，ひと手間加える必要があります。 また，吐き出されたデータには各項目がどの因子に属するのかについての情報は含まれていないので，もし因子ごとに得点を算出する場合には，それぞれの項目が属する因子が識別できるような列を新たに追加する必要があります。

R

まずは，BIS/BASのデータを抽出してみましょう。

```
df_e1 %>%
  dplyr::filter(task == 'bis_bas') %>%
  dplyr::select(tempID, response)
```

```
#   tempID       response
#   <chr>        <chr>
# 1 0pu18xqt8ws6 "{\"Q4\":3,\"Q5\":3,\"Q11\":3,\"Q8\":1,\"Q17\":0,\"Q20…
# 2 1f96wckn5emk "{\"Q3\":3,\"Q6\":3,\"Q13\":2,\"Q2\":2,\"Q19\":3,\"Q11…
# 3 31h414f13x6x "{\"Q18\":1,\"Q2\":3,\"Q17\":2,\"Q8\":0,\"Q12\":0,\"Q6…
# 4 3nfdasky1p0s "{\"Q1\":3,\"Q7\":2,\"Q20\":0,\"Q11\":2,\"Q13\":0,\"Q18…
```

この `response` 列の各セルに入っているのが，ある参加者のBIS/BAS全項目に対する反応です。 この不思議な文字列が json 形式です。

これを，以下のように縦長のデータフレームにするのがここでの目標です。 `q_id` が質問項目番号で，`value` がその項目に対する反応です。 縦長のデータフレームのほうが，何かと便利なのでこれの作成を目指します。

```
# A tibble: 300 x 3
#    tempID        q_id value
#    <chr>        <dbl> <int>
#  1 0pu18xqt8ws6     4     3
#  2 0pu18xqt8ws6     5     3
#  3 0pu18xqt8ws6    11     3
#  4 0pu18xqt8ws6     8     1
```

ただし，場合によっては横長のほうがいいこともあるので，目的に応じてどちらも作れるようになるといいでしょう。 今回の手順では，途中で横長のデータフレームも作ります。

```
# 一部の列は省略
# A tibble: 15 x 21
#    tempID         Q4    Q5   Q11    Q8   Q17   Q20   Q19
#    <chr>       <int> <int> <int> <int> <int> <int> <int>
#  1 0pu18xqt8w…     3     3     3     1     0     2     2
#  2 1f96wckn5e…     3     3     3     3     2     0     3
#  3 31h414f13x…     2     1     0     0     2     1     0
#  4 3nfdasky1p…     3     2     2     3     2     0     2
#  5 5a4h8j0g1x…     2     2     2     2     1     2     2
```

今回は以下の手順でBIS/BASの得点を分析するための縦長のデータフレームを作成します。

- json形式の文字列をRで扱えるようなデータ型にする
- 横長のデータフレームを作る
- それを縦長のデータフレームに変換する

jsonを変換する

`jsonlite::fromJSON()` という関数を使って，その引数にJSON形式の文字列を渡すと，リストに変換してくれます。

まずは，ある参加者ひとりのBIS/BASのデータに対して `fromJSON()` を使って，その挙動を確かめてみましょう。

```
df_ques_e1 <- df_e1 %>%
  dplyr::filter(task == 'bis_bas') %>%
  dplyr::select(tempID, response)

df_ques_e1$response[1] %>% jsonlite::fromJSON()

# 返り値はRのリスト型
# $Q4
# [1] 3
#
# $Q5
# [1] 3
#
# $Q11
# [1] 3
```

横長のデータフレームを作る

リスト型の変数は `as.data.frame()` でデータフレームに変換できます。

```
df_ques_e1$response[1] %>%
  jsonlite::fromJSON() %>%
  as.data.frame()

# 一部の列は省略
#   Q4 Q5 Q11 Q8 Q17 Q20 Q19 Q3 Q1 Q10
# 1  3  3   3  1   0   2   2  2  0   3
```

ということは，

- `purrr::map` を使って，response列の各要素に対してリストへの変換とデータフレーム化を行い
- それらをデータフレームとして結合

すれば，各行にある参加者の回答が並べられた横長のデータフレームができるだろうと予想できます。

ここで，ファイルの読み込みで使った処理を応用しますが，どうやら `as.data.frame` を使わなくても `bind_rows()` だけでデータフレームにしてくれるみたいです。

```
df_ques_e1$response %>%
  purrr::map(jsonlite::fromJSON) %>%
```

```
  dplyr::bind_rows()

# A tibble: 15 x 20
#       Q4    Q5   Q11    Q8   Q17   Q20   Q19
#    <int> <int> <int> <int> <int> <int> <int>
#  1     3     3     3     1     0     2     2
#  2     3     3     3     3     2     0     3
#  3     2     1     0     0     2     1     0
#  4     3     2     2     3     2     0     2
```

ただし，このままだと参加者IDがありません。`dplyr::mutate()` を使って，これまでの処理のフローに組み込みつつ，`tempID` の列を残した横長のデータフレームを作成します。

`dplyr::mutate()` は列を増やす関数です。`dplyr::mutate(新しい列名 = ベクトル)` で列を追加できます。 この `ベクトル` には既存の列名を活用することが多いです。

```
data.frame(a = c(1,2,3)) %>%
  dplyr::mutate(b = a + 1) # a列に1を足した列をbという名前で追加
#   a b
# 1 1 2
# 2 2 3
# 3 3 4
```

`mutate()` の引数にデータフレームを指定するとそのまま連結します。

```
data.frame(a = c(1,2,3)) %>%
  dplyr::mutate(data.frame(b = c("x", "y", "z")))
#   a b
# 1 1 x
# 2 2 y
# 3 3 z
```

この`mutate` の性質を利用して，`purrr::map` で作る横長のデータフレームをもとのデータフレームに連結します。

```
df_ques_e1_wide <- df_ques_e1 %>%
  dplyr::mutate(purrr::map_dfr(response, jsonlite::fromJSON)) %>%
  dplyr::select(!response) # response 列は不要なので，削除

# A tibble: 15 x 21
#    tempID          Q4    Q5   Q11    Q8   Q17   Q20   Q19
#    <chr>        <int> <int> <int> <int> <int> <int> <int>
#  1 0pu18xqt8w…     3     3     3     1     0     2     2
#  2 1f96wckn5e…     3     3     3     3     2     0     3
```

```
#  3 31h414f13x…     2     1     0     0     2     1     0
#  4 3nfdasky1p…     3     2     2     3     2     0     2
#  5 5a4h8j0g1x…     2     2     2     2     1     2     2
```

これで，参加者の反応が行ごとに並んだデータフレームが完成しました。

横長のデータフレームを縦長に変換する

横長のデータフレームを縦長に変換します。`tidyr::pivot_longer()` を使います。引数には，最低限，縦長にしたい列を指定します。実行すると，もともとの列名とそれに対応する値がいい感じに縦長になります。

```
data.frame(a = c("x","y"), b = c(1,2), c = c(3,4)) %>%
  tidyr::pivot_longer(cols = c(b,c))
# A tibble: 4 x 3
#   a     name  value
#   <chr> <chr> <dbl>
# 1 x     b         1
# 2 x     c         3
# 3 y     b         2
# 4 y     c         4
```

もともと 2 x 3 だったデータフレームが 4 x 3 のデータフレームになりました。

今回は，`Q`から始まる列が縦長にしたい列です。`starts_with("Q")` でまとめて指定できます。また，わかりやすさのために，もともとの列名が収納される列の名前が `"q_id"` となるように，`names_to` 引数に指定します。

```
df_ques_e1_wide %>%
  tidyr::pivot_longer(cols = starts_with("Q"), names_to = 'q_id')

# # A tibble: 300 x 3
#    tempID       q_id  value
#    <chr>        <chr> <int>
#  1 0pu18xqt8ws6 Q4        3
#  2 0pu18xqt8ws6 Q5        3
#  3 0pu18xqt8ws6 Q11       3
#  4 0pu18xqt8ws6 Q8        1
```

なお，質問項目の因子を識別するための列を後で追加することを考慮して，`q_id` が数字になるように `names_pattern` と `names_transform` という引数にそれぞれ必要な情報を指定している。

```
df_ques_e1_long <- df_ques_e1_wide %>%
  tidyr::pivot_longer(cols = starts_with("Q"),
                      names_to = 'q_id',
```

```
                        names_pattern = "Q([0-9]+)",
                        names_transform = list(q_id = as.numeric))
```

`names_pattern` には正規表現で数字だけ取り出すようにしています。 取り出された数字は文字列なので，`names_transform` で 数値に変換しています。

因子を識別するための列を追加する

まず，項目番号とそれが属する因子名が保存されたデータフレームを作成します。 実行するコードは以下の通り。

```
df_items_e1 <- list(bis = c(13, 10, 15, 6, 20, 18, 1),
                    drive = c(7, 2, 9, 17),
                    responsive = c(11, 19, 5, 14, 3),
                    seek = c(8, 12, 4, 16)) %>%
  purrr::map(~data.frame(q_id = .x)) %>%
  dplyr::bind_rows(.id = "fct_nm")
```

まず，原著論文を見て，因子とそれに含まれる項目番号のリストを作成します。

次に，`purrr::map` を使って，リストの各ベクトルに対して，`data.frame()` を実行して，`q_id` という列からなるデータフレームを4つ作ります。 このとき，`~` を使って，`~data.frame(q_id = .x)` というように引数を指定できるようにしています。 `.x` には，ここではリストの各ベクトルが順番に代入されます（例えば，`c(13, 10, 15, 6, 20, 18, 1)`）。 これで，`q_id` という列に項目番号が入っているデータフレームのリストが生成される。

最後にそれを`dplyr::bind_rows()` で縦に連結します。 このとき，`.id = "fct_nm"` という引数を指定することで，リストのID（`bis`, `drive`, など）を値に持つ`fct_nm` という列が追加されたデータフレームができます。

```
#         fct_nm q_id
# 1          bis   13
# ...
# 7          bis    1
# 8        drive    7
# ...
# 11       drive   17
# 12 responsive   11
# ...
# 16 responsive    3
# 17        seek    8
# ...
```

ちなみに，これまで同様に，`purrr::map` と`dplyr::bind_rows` は `purrr::map_dfr` にまとめられます。

```
purrr::map_dfr(~data.frame(q_id = .x), .id = "fct_nm")
```

これを`q_id`で紐づけて，BIS/BASの縦長データフレームに横に連結させれば，目的のデータフレームの完成です。 ある列で紐付けてデータフレームを連結するには，`dplyr::left_join()`を使います。

```
a <- data.frame(x = c("a", "b", "a"))
b <- data.frame(x = c("a", "b"), y = c(1, 4))
dplyr::left_join(a, b, by = "x")

#   x y
# 1 a 1
# 2 b 4
# 3 a 1
```

今回のBIS/BASの例では，以下のようにします。

```
df_bisbas_e1 <- dplyr::left_join(df_ques_e1_long, df_items_e1, by =
"q_id")
```

Python

Rでの処理と同じことをしている。 ただし，一旦横長のデータフレームを作るということはせずに，最初から縦長のデータフレームを作っています。 また，因子名を追加する方法もRとは違う発想でしています。

Rの処理の説明で力尽きてしまったので，そのうち加筆します。

```
df_ques_e1 = (
    df_e1
    .query('task == "bis_bas"')
    .loc[:, ['tempID', 'response']]
)


def json_to_df(tempID, resp):
    dict_x = json.loads(resp)
    new_df = pd.DataFrame(dict_x.values(), index=dict_x.keys(), columns=
['value'])
    new_df['tempID'] = tempID
    return new_df


df_bisbas_e1 = pd.concat([json_to_df(id, r) for _, id, r in
df_ques_e1.itertuples()])
```

```
# add factor names corresponding to the question number
dict_items = {'bis': [13, 10, 15, 6, 20, 18, 1],
              'drive': [7, 2, 9, 17],
              'responsive': [11, 19, 5, 14, 3],
              'seek': [8, 12, 4, 16]}

dict_q_fct = {}
for key, val in dict_items.items():
    for v in val:
        dict_q_fct[f'Q{v}'] = key

df_bisbas_e1['fct_nm'] = df_bisbas_e1.index.map(lambda x: dict_q_fct[x])
```

付録

今後の発展に参考になりそうな書籍・サイト

- 改訂2版 RユーザのためのRStudio[実践]入門〜tidyverseによるモダンな分析フローの世界
- https://note.nkmk.me （Python）
- https://heavywatal.github.io （主にR）

Rを使っているときに，ある関数の使い方がわからなくなったらその関数のヘルプを確認することが多いです `?関数名`。 ドキュメントがよくわからなくても，サンプルコードを回すと動作の雰囲気は掴めます。

あとは，Twitter をやっているのであれば，RやPythonに強そうな人をフォローしておくといいと思います。 便利な情報をよく発信してくれます。

---

1. 組み込み関数の `read.csv()` を使っても構いません。それほど変わりません。 ↩
2. 必ずしも実行速度が上がるわけではない（はず）ですが，`purrr::map` の場合は パイプ演算子 `%>%` を使ってコード全体をスッキリさせられます。 ↩
3. 厳密にはリストではなくイテレータ。for文で使う分にはどちらの型かを気にする必要はありません。 ↩
4. 引数で指定しているのは正規表現ではありません。 ↩