がんばらないデータ加工：R による繰り返し作業入門（前編）

やわらかクジラ

目次

はじめに

本書『**R で読む Excel ファイル**』を書こうと思ったのは「R と RStudio を使いたい！と思う人がもっと増えればいいのに」という願いからです。使う人が多くなれば、新しい知識に出会いやすくなりますし、仕事でも使う機会が増える可能性があります。

使う人を増やすためにはよい入門書や web サイトが必要ですが、それは巷にあふれていて無料でアクセスできるものも多いです。

例えば

- R for Data Science（英語）[*1]
- 日本社会心理学会　第 5 回春の方法論セミナー　R と Rstudio 入門 [*2]

そこで本書では目的を絞って、R（実際はすべて RStudio から使います）を使いたいと思わせる部分を解説することを目指します。R でどんな便利なことができるか、入門書などでもあまり深く解説されてない部分にフォーカスして紹介します。

本書の特徴

- これまでの解説で不足していること
  - 便利な関数や基本的な使い方の解説は多いが，データ加工の実務上知りたいコード例が豊富なわけではない
  - 同じ作業を大量の変数についてくり返し実行したい時のやり方の解説は少ない
- まずはよくあるモダンな R のデータラングリング解説
- 本書の強みは，くり返し同じ作業する部分を効率化したコードを併せて解説する点

[*1] https://r4ds.had.co.nz/
[*2] https://kazutan.github.io/JSSP2018_spring/index.html

- 自分の学習経験から，そのコード例が知りたかったんだ！という実用的な方法を収集
- 「がんばらない」とは，単純作業のくり返しに無駄なエネルギーを注がなくてよいようにすること
- まず基本の書き方を解説した後に，___【効率化】___ でより効率的にコードを書く解説を行う *【効率化】のタグが本書の核心になる。手作業の繰り返しをなるべく避けることが目指すべき点だからである。
- 冗長だが【別解】を示すことで様々な関数の働きを理解でき，データ加工の幅が広がる

本書の内容は、github レポジトリの ですべて公開しています。コードやサンプルデータはこちらのレポジトリをダウンロードしてお試しください。pdf 版が読みたい方は、以下のページで無料で入手可能です。自力でできる方は、`Build Book` でも作成できます。

- 技術書典マーケットの販売ページ

想定読者

色々な Excel ファイルを読み込んで分析する機会があるのであれば、全く R のことを知らない方から、少し R の経験があるけど複数のファイルを一度に読みこんだことはないというレベルの方ぐらいまでが対象となるでしょう。

本書の到達目標は、R での Excel ファイルの読み書きレベルをある程度高める、という所に定めました。その先は是非好きなように可視化なり解析なり進めていただければと思います。とはいえ、そこでお好きなように！と言われても路頭に迷う方もいるかもしれないので、データの内容把握に関して、要約値や欠損値の一覧、簡単な可視化、相関の一覧についても少しだけ解説しました。その一助として、特別付録として本書と並行してまとめた、可視化のためのggplot2 の辞書（特別付録について参照）もあります。

なお、データをきれいにする過程（例：前処理、データクリーニング、データクレンジング、データラングリングなど）については多くの説明を要するため、本書の範囲を超えます。これはまた別機会にまとめられたらなと思っています。

まず2章では、RStudio でファイルを読み書きする際に、最低限知っておいた方がよい知識について解説しておきます。とっつきにくいかもしれませんが、知っておいてよかったと後になって実感する類のものなので、使って慣れていきましょう。

4章は本書のメインである Excel ファイルの読み込みについて解説します。一つのファイ

ルの読み込みから、複数シート、複数ファイルの読み込みまで、様々なシーンに対応しました。また、読みこんだファイルを一つのデータフレーム[*3]にまとめる方法についても触れています。

章は Excel ファイルの保存についてです。ここでも、一つのファイルの保存から、複数ファイルの保存まで解説します。ここまでの内容が理解できれば、大量ファイルの読み書きにまつわる単純な繰り返し作業とはさよならできるでしょう。

章は関連知識として csv ファイルの読み込みと保存について解説します。windows ユーザーは文字コードの違いによる文字化けというつらみと対峙することになり、初学者はここで脱落していくことが多いのではないかと思います。そのために、サバイバルスキルとして知っておくことが有用だと思い書いておきました。自分が相当苦しんだので...

章は、読み込んだファイルの特徴をざっと把握する方法について解説しました。ここまでやれば、（きれいなデータであれば !）きっとデータ解析に入っていくことができるでしょう。

0.0.1 前提

- プロジェクト
- %>% tidyverse を読み込めば使える
- `dplyr::select()`

執筆環境

- 本書はbookdownにて執筆しました [*4]
- R および RStudio、パッケージのバージョン
  - R version 3.6.1
  - RStudio version 1.3.1073
  - readxl version 1.3.1
  - tidyverse version 1.3.0

[*3] 変数（列）とオブザベーション（行）が碁盤の目のようになった集まりの形のデータ。Excel であれば通常 1 行目に列名が入り、2 行目以降が個別のデータを表す。データ解析において便利で分かりやすいため、本書ではデータフレームの形で説明していく

[*4] https://bookdown.org/

注意事項など

- 本書の内容はすべて windows 環境を想定しています。
- この本に書いてある内容は、筆者が学習したことをまとめているものにすぎないため、正常な動作の保証はできません。使用する際は、自己責任でお願いします。

特別付録について

本書の執筆に先駆けて、順序が違う気がしますがまず付録の作成からはじめました。特別付録は以下でアクセス可能なオンライン付録となります。

- ggplot2 の辞書
  - https://izunyan.github.io/practice_ggplot2/

Twitter で応援してもらったら項目が増えていく仕様にしているので、もっと読みたい方はこちらのツイートに何らかのレスポンスください！

第 1 章

前提知識

-

1.1 本書に出てくるコード部分の見方

- グレーの背景部分は R のコードが書いてあり，その下の ## で始まる部分は出力結果を表す
  –

```
1 + 1
```

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

- ここでは 1 + 1 がコード部分で，## [1] 2 が出力結果部分
- [1] というのは，その次にくる値（ここでは 1 つしかないが）が何番目にあるかを示している
- たとえば，1 から 50 までの数値を出力してみる
  – コロン : で最初と最後の値をつなぐことで連番を表現できる

```
1:50
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

- コード部分に # で始まる文章がある場合は，コメントを表す。ここは実行されないので説明のために書かれる

```
# *（アスタリスク）は掛け算であることを示す

2 * 3  # ここにもコメントを入れられる
```

```
## [1] 6
```

1.2　プロジェクト

- データを加工して解析する際に、1 つのフォルダ（サブフォルダも含む）の中に関連するデータやコードなどをまとめておき、そのフォルダを **プロジェクト**と設定する
- これにより、ファイルの読み書きの際の場所指定をいちいち意識しないで作業できるようになる
- 詳しくは拙書の解説参照

1.3　パッケージ

- 様々な関数やデータなどがまとまっていて，読み込むと色々なことができる
  - 逆にいえば読み込まないと便利な作業ができないことが多い
- 例：`library(tidyverse)` または `require(tidyverse)` のように書くことで読み込める
- パッケージを読み込まなくても，パッケージ名:: 関数名 () でパッケージ内の関数が使える
  - どのパッケージの関数か明示するのにも便利なので，本書では多用する

1.4　関数

- 適切な値や変数などを指定すれば，データの処理や計算，統計解析など様々な処理を簡単に実行してくれる

- 例えば mean( ) などのように関数名 ( ) で出てくるので，( ) で囲まれてる所を見たらほぼ関数だと思えばよさそう
- ( ) の中に入る値を **引数**（ひきすう）と呼ぶ
- 引数は，でつないで追加していき，これによって実行したい処理のカスタマイズが可能　＋ 関数の ( ) の最初の位置に来るものを **第一引数**という

1.4.1 例

1.4.1.1 **複数のものを 1 つにする**: c( )

- **ベクトル**を作る（複数のものを 1 つにする）ための関数。
- 慣れてる人は当たり前に使っているので，初学者にとって理解しとくとよい最重要関数と思われる

```
c(1,2,3)
```

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

```
c("a", "b", "c") # " "で囲まれる値は文字を表す
```

```
## [1] "a" "b" "c"
```

```
# 複数あるように見えるが実は 1 つのベクトルになっている例
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

1.4.1.2 **平均値：**mean( )

- 引数にベクトルを入れることで平均値を計算する

```
mean(c(1,2,3))
```

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

```
# 欠損値（NA）があると結果が NA
mean(c(1, NA, 3))
```

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

```
# 引数に na.rm = TRUE を追加すると結果が出る。基本的に実務上は常につけておいたほうがよい
mean(c(1, NA, 3), na.rm = TRUE)
```

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

1.5　オブジェクト

- 計算の結果や，複数の数値や文字など（他にも色々）を 1 つの文字列に格納することができ，その後のコードで活用できる
- <-RStudio ではショートカット `alt + -`で出せる (Mac は `Option + -`)
- この後説明するデータフレームもオブジェクトに入れられる
  - データの少ないミニデータを作る時や，計算結果を格納するときに多用

1.5.1　例

```
res <- 1 + 1
res
```

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

```
res2 <- c(1, 2:4, 5)
res2
```

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

```
res3 <- c("a", "b")
res3
```

```
## [1] "a" "b"
```

```
rm(res, res2, res3)
```

1.6 データフレーム

1.6.1 本書で使う主なデータ

1.6.1.1 ペンギンデータ

- palmerpenguins パッケージの penguins データ

1.7 %>% （パイプ演算子）

- RStudio のショートカットは Ctrl + Shift + M(Mac は Cmd + Shift + M)
- R version 4.1 からは |> が同じ機能を持つと演算子して実装されたので，特にパッケージの読み込みをせずに使えるようになった。こちらを使う説明も今後増えていくと思われる
  - ショートカットで出るパイプを切り替えたい場合は，RStudio の Tools > Global Options > Code > Editing > use native pipe operator にチェックを入れる

第 2 章

変数（列）を選ぶ：select( )

- パッケージ dplyr の関数 select()
- tidy な世界では「列名 = 変数名」
- 変数が多い時に関心ある変数に限定したデータにしたい
- 関心ある変数の名前を取得したい
- 後々出てくる繰り返し作業で便利なヘルパー関数

2.1 データ読み込み

- データの指定を簡単にするために，penguins データを df と読み込む

```
library(tidyverse)

# パッケージが入ってなければ下記実行
# install.packages("palmerpenguins")

df <-
  palmerpenguins::penguins

df # データの表示
```

```
## # A tibble: 344 x 8
##   species island    bill_length_mm bill_depth_mm flipper_length_~
##   <fct>   <fct>              <dbl>         <dbl>             <int>
```

```
## 1 Adelie  Torgersen           39.1          18.7               181
## 2 Adelie  Torgersen           39.5          17.4               186
## 3 Adelie  Torgersen           40.3          18                 195
## # ... with 341 more rows, and 3 more variables:
## #   body_mass_g <int>, sex <fct>, year <int>
```

- 読み込みの様々な方法については拙書『R で読む Excel ファイル』参照

2.2 基本

- select( ) の中に関心のある変数名を，をつけて並べる
  - 変数は 1 つから OK

```
df %>%
  select(bill_length_mm, bill_depth_mm)
```

```
## # A tibble: 344 x 2
##   bill_length_mm bill_depth_mm
##            <dbl>         <dbl>
## 1           39.1          18.7
## 2           39.5          17.4
## 3           40.3          18
## # ... with 341 more rows
```

- 新しいデータフレームを作りたい場合は <-を使って新しいオブジェクトに格納する

```
df2 <-
  df %>% select(bill_length_mm)

df2
```

```
## # A tibble: 344 x 1
##   bill_length_mm
##            <dbl>
```

```
## 1           39.1
## 2           39.5
## 3           40.3
## # ... with 341 more rows
```

2.2.1 範囲指定

- 関心ある変数が指定された範囲に含まれていれば: でつなげて取得できる
  - 変数の連番をまとめて指定する時などに便利（例 変数 1: 変数 100）

```
df %>%
  select(bill_length_mm:flipper_length_mm)
```

```
## # A tibble: 344 x 3
##   bill_length_mm bill_depth_mm flipper_length_mm
##            <dbl>         <dbl>             <int>
## 1           39.1          18.7               181
## 2           39.5          17.4               186
## 3           40.3          18                 195
## # ... with 341 more rows
```

- 範囲に加えて追加の変数を追加できる
  - 飛び飛びの変数群を選びたいときに有用

```
df %>%
  select(bill_length_mm:flipper_length_mm, sex)
```

```
## # A tibble: 344 x 4
##   bill_length_mm bill_depth_mm flipper_length_mm sex
##            <dbl>         <dbl>             <int> <fct>
## 1           39.1          18.7               181 male
## 2           39.5          17.4               186 female
## 3           40.3          18                 195 female
## # ... with 341 more rows
```

2.2.2　中身が文字でも動く

- 変数名が" " で囲われていると，R では文字 (character) だと認識される

```
df %>%
  select("bill_length_mm", "bill_depth_mm")
```

```
## # A tibble: 344 x 2
##   bill_length_mm bill_depth_mm
##            <dbl>         <dbl>
## 1           39.1          18.7
## 2           39.5          17.4
## 3           40.3          18
## # ... with 341 more rows
```

- これは効率化を図りたいときに重要な特徴
- select( ) の中にたくさんの変数名を並べるより，事前に指定しておき代入した方が読みやすい
  - 様々なコード例でこの事前指定が多用されるので慣れるとよい

```
# あらかじめオブジェクト（ここでは vars）に引数を格納して後で使える
vars <- c("bill_length_mm", "bill_depth_mm")

df %>%
  select(all_of(vars))
```

```
## # A tibble: 344 x 2
##   bill_length_mm bill_depth_mm
##            <dbl>         <dbl>
## 1           39.1          18.7
## 2           39.5          17.4
## 3           40.3          18
## # ... with 341 more rows
```

- ここで vars は文字ベクトル（vector）のオブジェクトとなっている

- `all_of( )` の中に文字ベクトルを指定することで，それぞれの中身を変数名として認識する
  - 以前使われていた `one_of` は現在は非推奨

2.3 変数の指定に便利なヘルパー関数

- selection helper と呼ばれる `tidyselect` パッケージの関数群
- select の所で解説されることが多いが，後から出てくる `across()` と併せた活用場面が多いため，なじんでおくと後から楽になる

2.3.1 変数名の最初の文字列

- `bill` から始まる変数を選ぶ

```
df %>%
  select(starts_with("bill"))
```

```
## # A tibble: 344 x 2
##   bill_length_mm bill_depth_mm
##            <dbl>         <dbl>
## 1           39.1          18.7
## 2           39.5          17.4
## 3           40.3          18
## # ... with 341 more rows
```

2.3.2 変数名の最後の文字列

- `_mm` で終わる変数を選ぶ
  - `mm` だけだと他にも含まれる場合が出てくるので，`_` も含めた方が安全

```
df %>%
  select(ends_with("_mm"))
```

```
## # A tibble: 344 x 3
```

```
##   bill_length_mm bill_depth_mm flipper_length_mm
##            <dbl>         <dbl>             <int>
## 1           39.1          18.7               181
## 2           39.5          17.4               186
## 3           40.3          18                 195
## # ... with 341 more rows
```

2.3.3　変数名のどこかに含まれる文字列

- 指定した文字列を含んだ変数名を対象とする

```
df %>%
  select(contains("length"))
```

```
## # A tibble: 344 x 2
##   bill_length_mm flipper_length_mm
##            <dbl>             <int>
## 1           39.1               181
## 2           39.5               186
## 3           40.3               195
## # ... with 341 more rows
```

2.3.3.1　変数名のどこかに含まれる文字列：その 2

- 文字列で **正規表現**が使えるため柔軟な指定が可能
- ここでは，“length” または”depth” を含む変数名を対象
  - | が「または」を意味する

```
df %>%
  select(matches("length|depth"))
```

```
## # A tibble: 344 x 3
##   bill_length_mm bill_depth_mm flipper_length_mm
##            <dbl>         <dbl>             <int>
## 1           39.1          18.7               181
```

```
## 2           39.5          17.4               186
## 3           40.3          18                 195
## # ... with 341 more rows
```

2.3.4 上記の組み合わせ

2.3.4.1 かつ

- それぞれの条件を両方満たす

```
df %>%
  select(starts_with("bill") & contains("length"))
```

```
## # A tibble: 344 x 1
##   bill_length_mm
##            <dbl>
## 1           39.1
## 2           39.5
## 3           40.3
## # ... with 341 more rows
```

2.3.4.2 または

- それぞれの条件をいずれか満たす

```
df %>%
  select(starts_with("bill") | contains("length"))
```

```
## # A tibble: 344 x 3
##   bill_length_mm bill_depth_mm flipper_length_mm
##            <dbl>         <dbl>             <int>
## 1           39.1          18.7               181
## 2           39.5          17.4               186
## 3           40.3          18                 195
## # ... with 341 more rows
```

2.3.5 数値範囲

```
num_range
```

```
## function (prefix, range, width = NULL, vars = NULL)
## {
##     vars <- vars %||% peek_vars(fn = "num_range")
##     if (!is_null(width)) {
##         range <- sprintf(paste0("%0", width, "d"), range)
##     }
##     match_vars(paste0(prefix, range), vars)
## }
## <bytecode: 0x0000000019c66e88>
## <environment: namespace:tidyselect>
```

2.4 特定の変数を選ばない（落とす）

- 変数名の前に! をつける

```
df %>%
  select(!species)
```

```
## # A tibble: 344 x 7
##   island    bill_length_mm bill_depth_mm flipper_length_mm
##   <fct>              <dbl>         <dbl>             <int>
## 1 Torgersen           39.1          18.7               181
## 2 Torgersen           39.5          17.4               186
## 3 Torgersen           40.3          18                 195
## # ... with 341 more rows, and 3 more variables:
## #   body_mass_g <int>, sex <fct>, year <int>
```

- 複数列を落としたい場合は，!c( ) の中に対象の列名を含める

```
df %>%
  select(!c(bill_length_mm:flipper_length_mm, sex))
```

```
## # A tibble: 344 x 4
##   species island    body_mass_g  year
##   <fct>   <fct>           <int> <int>
## 1 Adelie  Torgersen        3750  2007
## 2 Adelie  Torgersen        3800  2007
## 3 Adelie  Torgersen        3250  2007
## # ... with 341 more rows
```

2.5 変数を並び変える

2.6 変数名を変更する

2.7 関心のある変数名を取得する

- データ分析の段階では，関心のある変数名を選択して，それらを代入する作業が頻出
- 変数名手打ちだと時間もかかるしミスもあるので，効率化のために必ずおさえておきたい技術

2.7.1 全ての変数名

```
df %>% names()
```

```
## [1] "species"           "island"            "bill_length_mm"
## [4] "bill_depth_mm"     "flipper_length_mm" "body_mass_g"
## [7] "sex"               "year"
```

2.7.2　選択した変数名を取得

- ベクトル

```
bill_vars <-
  df %>%
  select(starts_with("bill")) %>%
  names()

bill_vars
```

```
## [1] "bill_length_mm" "bill_depth_mm"
```

2.7.3　コピペに便利な形式に出力

- , で区切られた形式で出てくればそのまま `select()` に入れられるのに…と思った方のための便利関数 `dput()`

```
df %>%
  select(starts_with("b")) %>% # b から始まる変数名
  names() %>%
  dput()
```

```
## c("bill_length_mm", "bill_depth_mm", "body_mass_g")
```

- " " すらもいらない，という時は，新しく r script(アイコン New File または `ctrl + shift + n`) 開いて，`dput()` の出力を貼り付けてすべて置換する方法も

2.8 【応用編】複数データフレームで同時に select

2.8.1 複数データフレームで同じように select

2.8.2 複数データフレームで別々に select

https://izunyan.github.io/practice_bfi/#13_%E3%82%AF%E3%83%AD%E3%83%B3%E3%83%90%E3%83%83%E3%82%AF%E3%81%AE%E3%82%A2%E3%83%AB%E3%83%95%E3%82%A1

第 3 章

変数名を変更する：rename( )

- パッケージ dplyr の関数 rename()
- tidy な世界では「列名 = 変数名」
- 変数が多い時に関心ある変数に限定したデータにしたい
- 関心ある変数の名前を取得したい

3.1 基本

- 変更したい変数名を new = old の順に入力する
  - ここでは bill_length_mm を blmm に変更してみる
- 複数の変数名を変更する場合は，rename() の中に，でつなげていく
  - たくさんある場合に一つ一つ書いていくのは大変

```
df %>% names()
```

```
## [1] "species"           "island"            "bill_length_mm"
## [4] "bill_depth_mm"     "flipper_length_mm" "body_mass_g"
## [7] "sex"               "year"
```

```
df %>%
  rename(blmm = bill_length_mm)
```

```
## # A tibble: 344 x 8
##   species island  blmm bill_depth_mm flipper_length_~ body_mass_g
```

```
##   <fct>   <fct>  <dbl>          <dbl>             <int>       <int>
## 1 Adelie  Torge~  39.1           18.7               181        3750
## 2 Adelie  Torge~  39.5           17.4               186        3800
## 3 Adelie  Torge~  40.3           18                 195        3250
## # ... with 341 more rows, and 2 more variables: sex <fct>,
## #   year <int>
```

```
# 複数を rename する場合
df %>%
  rename(blmm = bill_length_mm,
         bdmm = bill_depth_mm)
```

```
## # A tibble: 344 x 8
##   species island    blmm  bdmm flipper_length_~ body_mass_g sex
##   <fct>   <fct>    <dbl> <dbl>            <int>       <int> <fct>
## 1 Adelie  Torgers~  39.1  18.7              181        3750 male
## 2 Adelie  Torgers~  39.5  17.4              186        3800 fema~
## 3 Adelie  Torgers~  40.3  18                195        3250 fema~
## # ... with 341 more rows, and 1 more variable: year <int>
```

- 複数変数を扱うときは rename_with() が便利。以下はそれを用いた例を示していく

3.2　同じ語を共通の語で置き換える

- 変数名の”bill” の部分を日本語の” くちばし” に変更していく
- まずは基本の知識でできる方法

```
df %>%
  rename(くちばし_length_mm = bill_length_mm,
         くちばし_depth_mm = bill_depth_mm)
```

```
## # A tibble: 344 x 8
##   species island    くちばし_length_mm くちばし_depth_mm
##   <fct>   <fct>                  <dbl>             <dbl>
```

```
## 1 Adelie  Torgersen               39.1              18.7
## 2 Adelie  Torgersen               39.5              17.4
## 3 Adelie  Torgersen               40.3              18
## # ... with 341 more rows, and 4 more variables:
## #   flipper_length_mm <int>, body_mass_g <int>, sex <fct>,
## #   year <int>
```

3.2.1 【効率化】str_replace() で一括変換 (1)

- ‘rename_with“は，まず適用したい関数を示し，そのあとに該当する変数を選ぶ
- 適用したい関数の中にある . の部分に，その後選ぶ変数が入っていく
- 語の置き換えは stringr::str_replace() を使う

```
df %>%
  rename_with(~str_replace(., "bill", "くちばし"),
              starts_with("bill"))
```

```
## # A tibble: 344 x 8
##   species island    くちばし_length_mm くちばし_depth_mm
##   <fct>   <fct>                  <dbl>             <dbl>
## 1 Adelie  Torgersen               39.1              18.7
## 2 Adelie  Torgersen               39.5              17.4
## 3 Adelie  Torgersen               40.3              18
## # ... with 341 more rows, and 4 more variables:
## #   flipper_length_mm <int>, body_mass_g <int>, sex <fct>,
## #   year <int>
```

3.2.1.1 【別解】

- select のように単に c( ) の中に変数を指定していくだけでも動く

```
df %>%
  rename_with(~str_replace(., "bill", "くちばし"),
              c(bill_length_mm, bill_depth_mm))
```

```
## # A tibble: 344 x 8
##   species island    くちばし_length_mm くちばし_depth_mm
##   <fct>   <fct>                  <dbl>             <dbl>
## 1 Adelie  Torgersen               39.1              18.7
## 2 Adelie  Torgersen               39.5              17.4
## 3 Adelie  Torgersen               40.3              18
## # ... with 341 more rows, and 4 more variables:
## #   flipper_length_mm <int>, body_mass_g <int>, sex <fct>,
## #   year <int>
```

3.3　同じ語を削除する

- “_mm” を取り除きたい場合，それを削除した変数名を指定すればよいが，たくさんあると大変

```
df %>%
  rename(bill_length = bill_length_mm,
         bill_depth  = bill_depth_mm,
         flipper_length = flipper_length_mm)
```

```
## # A tibble: 344 x 8
##   species island    bill_length bill_depth flipper_length
##   <fct>   <fct>           <dbl>      <dbl>          <int>
## 1 Adelie  Torgersen        39.1       18.7            181
## 2 Adelie  Torgersen        39.5       17.4            186
## 3 Adelie  Torgersen        40.3       18              195
## # ... with 341 more rows, and 3 more variables:
## #   body_mass_g <int>, sex <fct>, year <int>
```

3.3.1　【効率化】str_replace() で一括変換（2）

- `str_replace()` で変換先に空白`""` を指定すると削除できる

```
df %>%
  rename_with(~str_replace(., "_mm", ""),
              ends_with("mm"))
```

```
## # A tibble: 344 x 8
##   species island    bill_length bill_depth flipper_length
##   <fct>   <fct>           <dbl>      <dbl>          <int>
## 1 Adelie  Torgersen        39.1       18.7            181
## 2 Adelie  Torgersen        39.5       17.4            186
## 3 Adelie  Torgersen        40.3       18              195
## # ... with 341 more rows, and 3 more variables:
## #   body_mass_g <int>, sex <fct>, year <int>
```

3.3.1.1 【別解】

- `stringr::str_remove()` の方が直接的

```
df %>%
  rename_with(~str_remove(., "_mm"),
              ends_with("mm"))
```

```
## # A tibble: 344 x 8
##   species island    bill_length bill_depth flipper_length
##   <fct>   <fct>           <dbl>      <dbl>          <int>
## 1 Adelie  Torgersen        39.1       18.7            181
## 2 Adelie  Torgersen        39.5       17.4            186
## 3 Adelie  Torgersen        40.3       18              195
## # ... with 341 more rows, and 3 more variables:
## #   body_mass_g <int>, sex <fct>, year <int>
```

3.4 同じ接尾辞をつける

- 変数 year で 2007 年のみのデータに限定し，くちばし（bill）と翼（flipper）の変数名の末に”_2007” をつける

- rename の中に全部書いていけばできれば数が多いと大変

```
df %>%
  filter(year == 2007) %>%
  select(bill_length_mm:flipper_length_mm, year) %>%
  rename(bill_length_mm_2007 = bill_length_mm,
         bill_depth_mm_2007  = bill_depth_mm,
         flipper_length_mm_2007 = flipper_length_mm)
```

```
## # A tibble: 110 x 4
##   bill_length_mm_2007 bill_depth_mm_2007 flipper_length_mm~  year
##                 <dbl>              <dbl>               <int> <int>
## 1                39.1               18.7                 181  2007
## 2                39.5               17.4                 186  2007
## 3                40.3               18                   195  2007
## # ... with 107 more rows
```

3.4.1　【効率化】str_c() で一括指定

- 適用したい関数の中にある . の部分に，その後選ぶ変数が入っていく
- stringr::str_c() で指定した語をくっつける
- ここでは変数 year 以外なので，! をつけることで変数を指定できる

```
df %>%
  filter(year == 2007) %>%
  select(bill_length_mm:flipper_length_mm, year) %>%
  rename_with(~str_c(., "_2007"),
              !year)
```

```
## # A tibble: 110 x 4
##   bill_length_mm_2007 bill_depth_mm_2007 flipper_length_mm~  year
##                 <dbl>              <dbl>               <int> <int>
## 1                39.1               18.7                 181  2007
## 2                39.5               17.4                 186  2007
## 3                40.3               18                   195  2007
```

```
## # ... with 107 more rows
```

3.4.1.1 【別解】

```
df %>%
filter(year == 2007) %>%
  rename_with(~str_c(., "_2007"),
              matches("bill|flipper"))
```

```
## # A tibble: 110 x 8
##   species island    bill_length_mm_2007 bill_depth_mm_2007
##   <fct>   <fct>                   <dbl>              <dbl>
## 1 Adelie  Torgersen                39.1               18.7
## 2 Adelie  Torgersen                39.5               17.4
## 3 Adelie  Torgersen                40.3               18
## # ... with 107 more rows, and 4 more variables:
## #   flipper_length_mm_2007 <int>, body_mass_g <int>, sex <fct>,
## #   year <int>
```

3.5 同

-

```
df %>%
  rename(くちばし_length_mm = bill_length_mm,
         くちばし_depth_mm = bill_depth_mm)
```

```
## # A tibble: 344 x 8
##   species island    くちばし_length_mm くちばし_depth_mm
##   <fct>   <fct>                  <dbl>             <dbl>
## 1 Adelie  Torgersen                39.1              18.7
## 2 Adelie  Torgersen                39.5              17.4
## 3 Adelie  Torgersen                40.3              18
## # ... with 341 more rows, and 4 more variables:
```

```
## #   flipper_length_mm <int>, body_mass_g <int>, sex <fct>,
## #   year <int>
```

3.5.1 【効率化】

-

```
df %>%
  rename_with(~str_replace(., "bill", " くちばし"),
              starts_with("bill"))
```

```
## # A tibble: 344 x 8
##   species island    くちばし_length_mm くちばし_depth_mm
##   <fct>   <fct>                  <dbl>             <dbl>
## 1 Adelie  Torgersen               39.1              18.7
## 2 Adelie  Torgersen               39.5              17.4
## 3 Adelie  Torgersen               40.3              18
## # ... with 341 more rows, and 4 more variables:
## #   flipper_length_mm <int>, body_mass_g <int>, sex <fct>,
## #   year <int>
```

第 4 章

ケース（行）を選ぶ

- パッケージ dplyr の関数 filter()
- tidy な世界では「行 = ケース, 個人（wide 形式の場合)」
- ケースが多い時に関心あるケースに限定したデータにしたい

4.1 基本

- filter( ) の引数に論理式（TRUE or FALSE になるもの）を入れる
  - 論理式の部分について，最初の内は select( ) に入れるものと違って混乱するかもしれないが，慣れると段々分かってくると思う

```
df %>%
  filter(species == "Adelie")
```

```
## # A tibble: 152 x 8
##   species island    bill_length_mm bill_depth_mm flipper_length_~
##   <fct>   <fct>              <dbl>         <dbl>             <int>
## 1 Adelie  Torgersen           39.1          18.7               181
## 2 Adelie  Torgersen           39.5          17.4               186
## 3 Adelie  Torgersen           40.3          18                 195
## # ... with 149 more rows, and 3 more variables:
## #   body_mass_g <int>, sex <fct>, year <int>
```

- 種（species）が Adelie のケースのみ選ばれた

```
df %>%
  filter(bill_length_mm >= 50)
```

```
## # A tibble: 57 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm
##   <fct>   <fct>           <dbl>         <dbl>             <int>
## 1 Gentoo  Biscoe           50            16.3               230
## 2 Gentoo  Biscoe           50            15.2               218
## 3 Gentoo  Biscoe           50.2          14.3               218
## # ... with 54 more rows, and 3 more variables:
## #   body_mass_g <int>, sex <fct>, year <int>
```

- くちばしの長さ（bill_length_mm）が 50 以上のケースのみ選ばれた

第 5 章

新しい変数（列）の作成：mutate( )

- パッケージ `dplyr` の関数 `mutate()`
- 新しい変数の列を作成する
  - 既にある
- mutate の機能解説
- 効率化のための `across()`

5.1 データ読み込み

- 国際パーソナリティ項目プールからの 2800 名分のデータ。
- 質問項目が 25 問あり，5 つの構成概念（ここでは因子という）に対応する項目への回答を足し合わせたスコアを計算する
- 性，教育歴，年齢の変数もあり
- 項目に対し想定される因子（因子名の頭文字が変数名と対応）
  - Agree A1 から A5
  - Conscientious C1 から C5
  - Extraversion E1 から E5
  - Neuroticism N1 から N5
  - Openness O1 から O5
- 回答選択肢
  - 1 Very Inaccurate まったくあてはまらない
  - 2 Moderately Inaccurate あてはまらない
  - 3 Slightly Inaccurate ややあてはまらない
  - 4 Slightly Accurate ややあてはまる

  - 5 Moderately Accurate あてはまる
  - 6 Very Accurate 非常にあてはまる

```
# パッケージが入ってなければ下記実行
# install.packages("psychTools")

df_bfi <-
  psychTools::bfi %>%
  as_tibble()          # 表示に便利な tibble 形式に
```

5.2 基本

- データフレームに新しい列を計算して追加する関数
- mutate( ) の中に新しく作成する変数名を入れ，= でつないで計算式を入れる
- ここでは，まず変数 A1 の平均値（全ケース同じ値が入る）を計算し，個々の値の差分をする例を示す

```
df_bfi %>%
  select(A1) %>%                        # A1 のみを残す
  mutate(
    mean_a1 = mean(A1, na.rm = TRUE), # A1 の平均値を作成（NA は除外）
    dif_a1_mean = A1 - mean_a1)       # 各個体の A1 と平均値の差分を計算
```

```
## # A tibble: 2,800 x 3
##      A1 mean_a1 dif_a1_mean
##   <int>   <dbl>       <dbl>
## 1     2    2.41      -0.413
## 2     2    2.41      -0.413
## 3     5    2.41       2.59
## # ... with 2,797 more rows
```

- mean_a1 列には A1 の平均値がすべて同じ値で入る（平均値だけの計算がしたければ6を参照）
- dif_a1_mean 列は，A1 列から mean_a1 列を引いた値が入る

5.3 変数の型の変換

- 変数には型の情報が伴い，統計解析やデータ加工の際に適切な型を求められることがあるため理解が必要
  - 小数も扱う数値 *<dbl>*
  - 整数 *<int>*
  - 文字 *<chr>*
  - 因子 *<fct>*
- 変数の型の確認は色々方法があるが，tibble 形式のデータフレームなら `select()` で OK
  - tibble 形式でなくても，最後に `glimpse()` で確認可能

```
df_bfi %>%
  select(gender, education)
```

```
## # A tibble: 2,800 x 2
##   gender education
##    <int>     <int>
## 1      1        NA
## 2      2        NA
## 3      2        NA
## # ... with 2,797 more rows
```

```
df_bfi %>%
  select(gender, education) %>%
  glimpse()
```

```
## Rows: 2,800
## Columns: 2
## $ gender    <int> 1, 2, 2, 2, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, ~
## $ education <int> NA, NA, NA, NA, NA, 3, NA, 2, 1, NA, 1, NA, N~
```

- gender, education 列が *<int>* になっているので整数型になっている

5.3.1　型の変換

- ここでは，2 つの数値型変数 gender, education を因子型に変換する例を示す
- それぞれ `factor()` で因子型に変換

```
df_bfi %>%
  select(gender, education) %>%
  mutate(gender = factor(gender),
         education = factor(education))
```

```
## # A tibble: 2,800 x 2
##   gender education
##   <fct>  <fct>
## 1 1      <NA>
## 2 2      <NA>
## 3 2      <NA>
## # ... with 2,797 more rows
```

- gender, education 列が *<fct>* になっているので整数型になっている

###【効率化】複数の変数に対し一度の指定で実行 {#mu-kata-across}

- 変換したい変数が大量にあるときは上記の方法では大変
- `across()` を使うと，指定した変数に対して同じ内容の処理なら **1 回**ですむようになる
  - かつての `mutate_at()`, `mutate_if()` など

```
df_bfi %>%
  mutate(across(c(gender, education),
                factor)) %>%
  select(gender, education)    # 結果表示のため冗長だが変わった変数だけ select
```

```
## # A tibble: 2,800 x 2
##   gender education
##   <fct>  <fct>
```

```
## 1 1      <NA>
## 2 2      <NA>
## 3 2      <NA>
## # ... with 2,797 more rows
```

5.4 across( ) の特徴

- 変数の指定に2.3で解説したヘルパー関数が使える

```
df_bfi %>%
  mutate(across(starts_with("n"),
                factor)) %>%
  select(starts_with("n"))   # 結果表示のため
```

```
## # A tibble: 2,800 x 5
##   N1    N2    N3    N4    N5
##   <fct> <fct> <fct> <fct> <fct>
## 1 3     4     2     2     3
## 2 3     3     3     5     5
## 3 4     5     4     2     3
## # ... with 2,797 more rows
```

- ?? で解説した文字も使える

```
vars <- c("N1", "N2", "N3", "N4", "N5")

df_bfi %>%
  mutate(across(all_of(vars),
                factor)) %>%
  select(starts_with("n"))   # 結果表示のため
```

```
## # A tibble: 2,800 x 5
##   N1    N2    N3    N4    N5
##   <fct> <fct> <fct> <fct> <fct>
## 1 3     4     2     2     3
```

```
## 2 3     3     3     5     5
## 3 4     5     4     2     3
## # ... with 2,797 more rows
```

5.4.1 【重要知識 !】新しい変数名にして追加

- ここはこの後色々なところで出てくる方法のため理解しておきたい
- 適用する関数をリストにする（list() に入れる）ことで，変数名を変更して追加できる
- list() に入れるときはこれまでと異なる書き方が必要になる
  - 関数名の前に ~（チルダ）が必要
  - list 内の関数 () 内に.x が必要（この場合は x がなくても動く）。ここに across() の第一引数に指定した変数が入っていくという意味

```
df_bfi %>%
  mutate(across(c(gender, education),
                list(f = ~factor(.x)))) %>%
  select(matches("gender|education"))
```

```
## # A tibble: 2,800 x 4
##   gender education gender_f education_f
##    <int>     <int> <fct>    <fct>
## 1      1        NA 1        <NA>
## 2      2        NA 2        <NA>
## 3      2        NA 2        <NA>
## # ... with 2,797 more rows
```

- 因子型に変換した変数の末尾に _f がつく

5.5 合計点の作成

```
df_bfi_n <-
  df_bfi %>%
```

```
  select(N1:N5) %>%
  mutate(neuroticism = N1 + N2 + N3 + N4 + N5)

df_bfi_n
```

```
## # A tibble: 2,800 x 6
##      N1    N2    N3    N4    N5 neuroticism
##   <int> <int> <int> <int> <int>       <int>
## 1     3     4     2     2     3          14
## 2     3     3     3     5     5          19
## 3     4     5     4     2     3          18
## # ... with 2,797 more rows
```

5.5.1 足し上げる変数に欠損があるとどうなるか

```
df_bfi_n %>%
  filter(is.na(neuroticism))      # neuroticism が NA なケースに限定
```

```
## # A tibble: 106 x 6
##      N1    N2    N3    N4    N5 neuroticism
##   <int> <int> <int> <int> <int>       <int>
## 1     4     5     3     2    NA          NA
## 2    NA     2     1     2     2          NA
## 3     1     2     1     2    NA          NA
## # ... with 103 more rows
```

5.6 連番から ID の作成

- dplyr::row_number() で行番号から ID を作成

```
df_bfi_n %>%
  mutate(id = row_number())
```

```
## # A tibble: 2,800 x 7
##       N1    N2    N3    N4    N5 neuroticism    id
##    <int> <int> <int> <int> <int>       <int> <int>
## 1     3     4     2     2     3          14     1
## 2     3     3     3     5     5          19     2
## 3     4     5     4     2     3          18     3
## # ... with 2,797 more rows
```

5.6.1　【別解】行の名前を直接変数化

- 実は mutate を使わなくてもできて，データの最初に持ってこれる便利関数がある
- `tibble::rowid_to_column()`
  - `var =` で変数名を指定

```
df_bfi_n %>%
  rowid_to_column(var = "id")
```

```
## # A tibble: 2,800 x 7
##       id    N1    N2    N3    N4    N5 neuroticism
##    <int> <int> <int> <int> <int> <int>       <int>
## 1     1     3     4     2     2     3          14
## 2     2     3     3     3     5     5          19
## 3     3     4     5     4     2     3          18
## # ... with 2,797 more rows
```

```
# この先使わないのでデータフレーム削除
rm(df_bfi_n)
```

5.7　逆転項目を作る

5.7.1　逆転項目の確認

- bfi データの場合，どの項目を逆転する必要があるかを示す情報（-変数名で表現）がパッケージに含まれている

  - psychTools::bfi.keys で確認可能
- したがって，“-A1”, “-C4”, “-C5”, “-E1”, “-E2”, “-O2”, “-O5” が対象

5.7.2 逆転（recode）

- dplyr::recode() を使用
- 入れ替えたい値を old = new で並べていく
  - この等式の順番が他 (mutate など) と逆になるため，recode() は将来引退する可能性あり

```
df_bfi %>%
  mutate(A1_r = recode(A1, `1` = 6L, `2` = 5L, `3` = 4L,       # old の数値は ` `で囲む必要が
                           `4` = 3L, `5` = 2L, `6` = 1L)) %>% # new の数値に L がつくのは，型
  select(A1, A1_r)
```

```
## # A tibble: 2,800 x 2
##      A1  A1_r
##   <int> <int>
## 1     2     5
## 2     2     5
## 3     5     2
## # ... with 2,797 more rows
```

5.7.2.1 変数 2 つ以上を逆転

- A1 と同様に同じ形をくり返し変数名だけ変えていけばできるが，コードが長くなりミスも生じやすくなる

```
df_bfi %>%
  mutate(A1_r = recode(A1, `1` = 6L, `2` = 5L, `3` = 4L,
                           `4` = 3L, `5` = 2L, `6` = 1L),
         C4_r = recode(C4, `1` = 6L, `2` = 5L, `3` = 4L,
                           `4` = 3L, `5` = 2L, `6` = 1L)) %>%
  select(A1, A1_r, C4, C4_r)
```

```
## # A tibble: 2,800 x 4
##      A1  A1_r    C4  C4_r
##   <int> <int> <int> <int>
## 1     2     5     4     3
## 2     2     5     3     4
## 3     5     2     2     5
## # ... with 2,797 more rows
```

5.7.2.2 【効率化】変数 2 つ以上を逆転

- 5.4.1 で解説した list に関数を入れる方法

```
df_bfi %>%
  mutate(across(c(A1, C4, C5, E1, E2, O2, O5),
                list(r = ~recode(.x, `1` = 6, `2` = 5, `3` = 4,
                                 `4` = 3, `5` = 2, `6` = 1)))) %>%
  select(A1, A1_r, C4,  C4, C5, C5_r, E1, E1_r, E2, E2_r, O2, O2_r, O5, O5_r)
```

```
## # A tibble: 2,800 x 13
##      A1  A1_r    C4    C5  C5_r    E1  E1_r    E2  E2_r    O2
##   <int> <dbl> <int> <int> <dbl> <int> <dbl> <int> <dbl> <int>
## 1     2     5     4     4     3     3     4     3     4     6
## 2     2     5     3     4     3     1     6     1     6     2
## 3     5     2     2     5     2     2     5     4     3     2
## # ... with 2,797 more rows, and 3 more variables: O2_r <dbl>,
## #   O5 <int>, O5_r <dbl>
```

5.7.3 【別解】逆転（公式）

- 項目を反転する公式が「(max + min) - 回答値」であることを利用
  - `psych::reverse.code()` の help 参照
  - 例：最小値 1, 最大値 4 の場合，max + min = 5 となり，回答値が 2 の場合，5 - 2 = 3 となり反転された結果となる

```
min <- 1
max <- 6

df_bfi %>%
  mutate(A1_r = max + min - A1,
         C4_r = max + min - C4) %>%
  select(A1, A1_r, C4, C4_r)
```

```
## # A tibble: 2,800 x 4
##      A1  A1_r    C4  C4_r
##   <int> <dbl> <int> <dbl>
## 1     2     5     4     3
## 2     2     5     3     4
## 3     5     2     2     5
## # ... with 2,797 more rows
```

5.7.3.1 【効率化】変数 2 つ以上を逆転

- ~ の後に計算式がきても動く
- ここでは，max + min - .x の.x に across 内に置かれた変数が入っていく

```
df_bfi %>%
  mutate(across(c(A1,C4),
                list(r = ~ max + min - .x))) %>%
  select(A1, A1_r, C4, C4_r)
```

```
## # A tibble: 2,800 x 4
##      A1  A1_r    C4  C4_r
##   <int> <dbl> <int> <dbl>
## 1     2     5     4     3
## 2     2     5     3     4
## 3     5     2     2     5
## # ... with 2,797 more rows
```

5.7.3.2　逆転した変数を含むデータフレーム作成

- これ以降で使用するため，項目を逆転した変数を格納しておく

```
df_bfi <-
  df_bfi %>%
  mutate(across(c(A1, C4, C5, E1, E2, O2, O5),
                list(r = ~ max + min - .x)))
```

5.8　2 区分変数

```
  # bfi_a %>%
  # mutate(gender = fct_recode(gender, male   = "1",
  #                                    female = "2"),
  #        education = fct_recode(education, "HS" = "1",
  #                                          "finished HS" = "2",
  #                                          "some college" = "3",
  #                                          "college graduate" = "4",
  #                                          "graduate degree" = "5"
  #                                )
  # )
```

第 6 章

要約値を作る：summarise( )

- パッケージ dplyr の関数 summarise()
- 結果をデータフレームとして出力するため，扱いが便利
- データの要約作業はデータを知るうえで頻繁に行うことが想定される
  - 便利な要約パッケージが色々あるものの，summarise() を使いこなせると役に立つことが多い

6.1 基本

- ( ) の中に出力したい変数名を書き，= の後に関数を入れる
- NA がある場合，引数 na.rm = TRUE がないと結果が出ないので，ほとんどの場合つけて置いた方がよい

```
df %>%
  summarise(blm_平均値 = mean(bill_length_mm, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   blm_平均値
##        <dbl>
## 1       43.9
```

6.2　複数の計算

- 複数の変数について平均値と SD と n を出したいときは，基本知識では全部書くので長くなる

```
df %>%
  summarise(blm_mean = mean(bill_length_mm, na.rm = TRUE),
            bdm_mean = mean(bill_depth_mm, na.rm = TRUE),
            blm_sd = sd(bill_length_mm, na.rm = TRUE),
            bdm_sd = sd(bill_depth_mm, na.rm = TRUE),
            blm_n  = sum(!is.na(bill_length_mm)),
            bdm_n  = sum(!is.na(bill_depth_mm)))
```

```
## # A tibble: 1 x 6
##   blm_mean bdm_mean blm_sd bdm_sd blm_n bdm_n
##      <dbl>    <dbl>  <dbl>  <dbl> <int> <int>
## 1     43.9     17.2   5.46   1.97   342   342
```

6.2.1　【効率化】

- ??で出てきた across() がここでも有用
- across() の第一引数に指定したい変数名ベクトル，またはヘルパー関数を入れる
- 実行したい関数を list 内に名前（接尾辞）をつけて列挙し，関数の前に ~ をつける
  –

```
df %>%
  summarise(across(c(bill_length_mm, bill_depth_mm),
                   list(mean = ~mean(.x, na.rm = TRUE),
                        sd = ~sd(.x, na.rm = TRUE),
                        n = ~sum(!is.na(.x))))))
```

```
## # A tibble: 1 x 6
##   bill_length_mm_mean bill_length_mm_sd bill_length_mm_n
##                 <dbl>             <dbl>            <int>
```

```
## 1                43.9              5.46              342
## # ... with 3 more variables: bill_depth_mm_mean <dbl>,
## #   bill_depth_mm_sd <dbl>, bill_depth_mm_n <int>
```

- across() ではヘルパー関数が使える！

```
df %>%
  summarise(across(starts_with("bill"),
                   list(mean = ~mean(.x, na.rm = TRUE),
                        sd = ~sd(.x, na.rm = TRUE),
                        n = ~sum(!is.na(.x)))))
```

```
## # A tibble: 1 x 6
##   bill_length_mm_mean bill_length_mm_sd bill_length_mm_n
##                 <dbl>             <dbl>            <int>
## 1                43.9              5.46              342
## # ... with 3 more variables: bill_depth_mm_mean <dbl>,
## #   bill_depth_mm_sd <dbl>, bill_depth_mm_n <int>
```

6.2.2 【並び替え】

- tidyr::pivot_longer() で，データフレームの行列入れ替えができる
- 引数を names_pattern と names_to を下記のように指定することで，変数の接尾辞を列名にできる

```
df %>%
  summarise(across(bill_length_mm:body_mass_g,
                   list(mean = ~mean(.x, na.rm = TRUE),
                        sd = ~sd(.x, na.rm = TRUE),
                        n = ~sum(!is.na(.x))))) %>%
  pivot_longer(everything(),
               names_to = c("items", ".value"), # ".value"の部分を列名に
               names_pattern = "(.*)_(.*)")     # 正規表現
```

```
## # A tibble: 4 x 4
```

```
##   items               mean     sd     n
##   <chr>              <dbl>  <dbl> <int>
## 1 bill_length_mm      43.9   5.46   342
## 2 bill_depth_mm       17.2   1.97   342
## 3 flipper_length_mm  201.   14.1    342
## 4 body_mass_g       4202.  802.     342
```

6.3　層別（グループ別）集計

- group_by( ) にグループを表す変数を指定するとできる

```
df %>%
  group_by(species) %>%
  summarise(across(c(bill_length_mm, bill_depth_mm),
                   list(mean = ~mean(.x, na.rm = TRUE),
                        sd = ~sd(.x, na.rm = TRUE))))
```

```
## # A tibble: 3 x 5
##   species   bill_length_mm_mean bill_length_mm_~ bill_depth_mm_m~
##   <fct>                   <dbl>            <dbl>             <dbl>
## 1 Adelie                   38.8             2.66              18.3
## 2 Chinstrap                48.8             3.34              18.4
## 3 Gentoo                   47.5             3.08              15.0
## # ... with 1 more variable: bill_depth_mm_sd <dbl>
```

- グループを重ねることも可能

```
df %>%
  group_by(species, sex) %>%
  summarise(across(c(bill_length_mm, bill_depth_mm),
                   list(mean = ~mean(.x, na.rm = TRUE),
                        sd = ~sd(.x, na.rm = TRUE))))
```

```
## # A tibble: 8 x 6
## # Groups:   species [3]
```

```
##   species   sex    bill_length_mm_mean bill_length_mm_sd
##   <fct>     <fct>                <dbl>              <dbl>
## 1 Adelie    female                37.3               2.03
## 2 Adelie    male                  40.4               2.28
## 3 Adelie    <NA>                  37.8               2.80
## 4 Chinstrap female                46.6               3.11
## 5 Chinstrap male                  51.1               1.56
## 6 Gentoo    female                45.6               2.05
## 7 Gentoo    male                  49.5               2.72
## 8 Gentoo    <NA>                  45.6               1.37
## # ... with 2 more variables: bill_depth_mm_mean <dbl>,
## #   bill_depth_mm_sd <dbl>
```

6.4 【効率化】関数にする

6.4.1 基本

- 関数名 <- function(引数){ 計算式やコード } で関数を定義できる

```
add_one <-
  function(x){
    x + 1
  }

add_one(2)
```

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

6.4.2 複数変数の平均値と SD と n を計算する関数

- 引数にデータフレーム（data）と変数（vars）を入れる

```
mean_sd_n <- function(data, vars){
data %>%
```

```
  summarise(across({{vars}},
                   list(mean = ~mean(.x, na.rm = TRUE),
                        sd = ~sd(.x, na.rm = TRUE),
                        n = ~sum(!is.na(.x)))))
}
```

- ここで定義した関数 mean_sd_n( ) にデータフレームと変数を入れると結果が表示される

```
mean_sd_n(df, bill_length_mm)
```

```
## # A tibble: 1 x 3
##   bill_length_mm_mean bill_length_mm_sd bill_length_mm_n
##                 <dbl>             <dbl>            <int>
## 1                43.9              5.46              342
```

- vas の部分は across( ) の第一引数に入れるものと同じ指定ができるため，変数ベクトルやヘルパー関数が入る

```
# 変数ベクトル
mean_sd_n(df, c(flipper_length_mm, body_mass_g))
```

```
## # A tibble: 1 x 6
##   flipper_length_mm_mean flipper_length_mm_sd flipper_length_mm_n
##                    <dbl>                <dbl>               <int>
## 1                   201.                 14.1                 342
## # ... with 3 more variables: body_mass_g_mean <dbl>,
## #   body_mass_g_sd <dbl>, body_mass_g_n <int>
```

```
# 文字でも可能
# mean_sd_n(df, c("flipper_length_mm", "body_mass_g"))

# ヘルパー関数
mean_sd_n(df, starts_with("bill"))
```

```
## # A tibble: 1 x 6
```

```
##   bill_length_mm_mean bill_length_mm_sd bill_length_mm_n
##                 <dbl>             <dbl>            <int>
## 1                43.9              5.46              342
## # ... with 3 more variables: bill_depth_mm_mean <dbl>,
## #   bill_depth_mm_sd <dbl>, bill_depth_mm_n <int>
```

あとがき

あとがき

本書の執筆にあたり、同人誌制作の先輩である天川榎 @EnokiAmakawa 氏から背中押し＆多くの助言をいただきました。この場を借りてお礼申し上げます。

著者：やわらかクジラ
発行： 2020 年 9 月 12 日
サークル名：ヤサイゼリー
twitter： @matsuchiy
印刷：電子出版のみ