がんばらないデータ加工ーＲによる繰り返し作業入門ー　前編

やわらかクジラ

目次

はじめに

- 本書の目的
  - データ加工での面倒な作業を R でらくらく実行できるようになるための基礎知識を紹介
- 本書の内容
  - 実際は核心の部分に入る前の準備段階までにとどまる。タイトルに「前編」とあるのはその理由による
  - 本当は，複数データセットの複数変数をいっぺんに加工，集計，視覚化！みたいなのをまとめたかったが，そこに入るための事前知識が思っていたより多かったため，まずはそれらを解説することに徹した
  - 既刊書では省いた R のモダンな方法を使ったデータ加工の過程（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）で用いる基本関数の紹介
- 執筆動機
  - 本書を書こうと思ったのは拙既刊書『R で読む Excel ファイル』と同じく，「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
- 今後の展望
  - よりタイトルの内容に沿った次回作の「後編」（もしかしたら「中編」も）をお楽しみに！

本書の特徴

- タイトルの「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
- 扱う内容は自分が学び始めの時に教えてもらいたかったこと
- これまでの解説で不足していると考えられるポイント
  - 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
- 本書の強み
  - 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
  - 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整理
- まずモダンな R のデータ加工法での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
- 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，手持ちの武器が増えデータ加工の幅が広がる

想定読者

- R と RStudio をダウンロードして PC にインストールまでできることが最低条件
  - web 上に様々な解説があり，あとは基本的に OK していけばできるはず
- 初学者から始めてちょっと背伸びできるくらいまでが到達目標

各セクションの概要

まず1章では、R と RStudio に初めて触れる方，初学者を対象とした前提知識を解説する。ゆくゆく楽をするためには避けて通れない知識なので，用語になじんでおきたい

2章はデータの列（変数）を選ぶ方法を解説する。データをコンパクトにしたり，後のデータ解析等で必要な変数を取得したりするなど，データ加工プロセス全体で必要な基本知識もあるので最初に学んでおきたい

3章はデータの列名（変数名）を変える方法について解説する。単純に見えるがデータ加工の際になくてはならない技術である。効率化させるためには初心者から少し脱する必要があり，奥が深い

4章はデータの行（ケースまたはオブザベーション）を選ぶ方法を解説する。データや加工した結果，分析した結果をコンパクトにするのに役立つ。

5章はデータに新しい列を追加する方法について解説する。例えば合計点の作成や，年齢層カテゴリや 2 区分変数（いわゆるダミー変数）の作成など，変数を計算して新しい変数

を作る作業はよく発生する。効率化のために避けて通れない `across( )` についてもここで解説する

6章は要約値の計算について解説する。実務では大量の変数を一気に処理する必要がある場面が多いので，効率化を意識した説明を多く入れている

執筆環境

- 本書はbookdownにて執筆

R および RStudio、パッケージのバージョン

- rstudio だけなぜか表示されないので手動で…
  - バージョン 2021.09.1+372 Ghost Orchid (desktop)

ind	values
version	R version 4.1.0 (2021-05-18)
os	Windows 10 x64 (build 19043)
system	x86_64, mingw32
date	2022-01-16

package	loadedversion
bookdown	0.24
tidyverse	1.3.1

注意事項など

- 本書の内容はすべて windows 環境を想定
- 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ない。使用する際は、自己責任で

ライセンス

  - ただし，ライセンスの適用は本書での著作部分のみとなり，用いているデータやパッケージや画像などはそれぞれのライセンスに準じる
- 本書の内容は、github レポジトリですべて公開

関連情報

- 『R で読む Excel ファイル』
  - 技術書典 9 で頒布した R での Excel および csv ファイル読み込み解説本
  - github
- ggplot2 の辞書
  - 視覚化のための `ggplot2` パッケージの辞書的メモ

第 1 章

前提知識

- ここに出てくる用語は初学者にとってなじみがないものばかりかもしれないが，R でデータ加工をらくらくできるようになるためには避けて通れない

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
- RStudio 画面の右上に Project 設定のメニューがある
  - `Project（None）> New Project > Existing Directory` と選び，プロジェクトにしたいフォルダを設定する（Figure1.1）

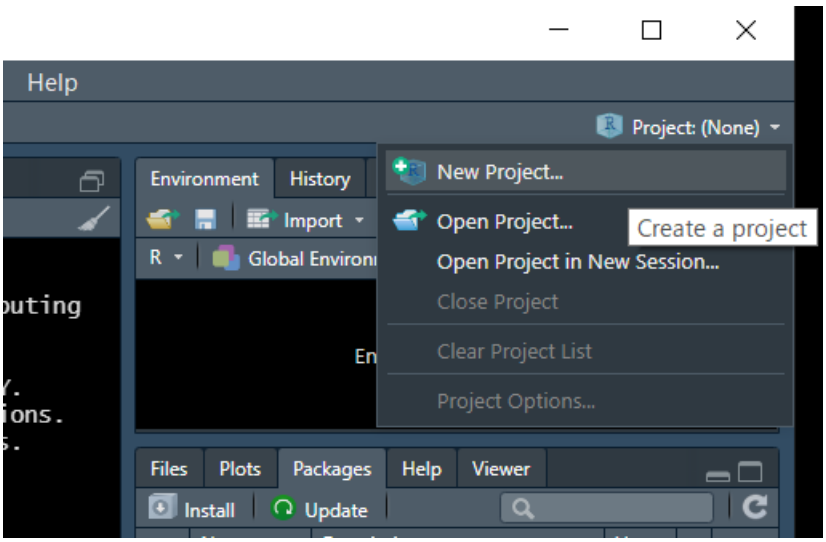


図1.1　プロジェクトの設定

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
- インストールされているパッケージは RStudio のデフォルト画面で右下にあるウィンドウ（ペインと呼ぶ）のパッケージタブで確認可能（Figure1.2）

- 入っていないパッケージは，インターネットにつながっていれば以下の方法でインストールできる
  - パッケージタブの `install` をクリックして出てくるウィンドウでパッケージ名を入力
  - コマンドから `install.packages("` パッケージ名をここに入れる`")`

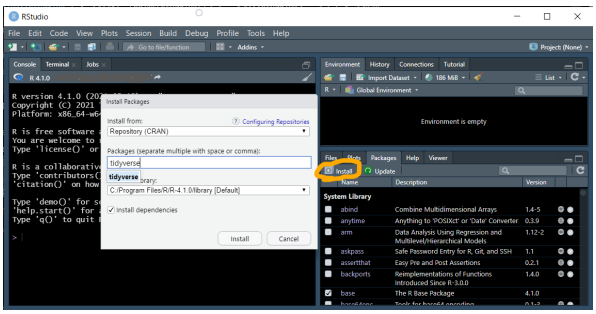


図1.2 パッケージタブからのインストール

- 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
  - 以下，例えば「パッケージ `dplyr` の関数 `select( )`」は `dplyr::select( )` と表現する

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
  - データ加工の技術は，色々な便利関数をどの場面でどうやって使うかにつきる
- 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
  - ベクトルと聞くと数学苦手だった人はいやな記憶を思い出すかもしれないが，R ではとにかく「**複数のものを 1 つにしたもの**」と理解しておけば何となると思う
- c() は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
- ベクトルは，後に解説するデータフレームでの列単位のデータを扱う際にも有用

```
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
- <-の矢印の先にあるのがオブジェクト。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)
```

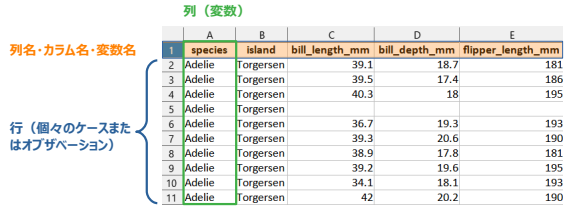
1.6　データフレーム

- 行（ケースまたはオブザベーション）と列（変数）が碁盤の目のようになった集まりの形のデータ（Figure1.3）
  - Excel で表現するのであれば通常 1 行目に列名が入り、2 行目以降が個別のケース（データ）を表す形。R のデータフレームでは列名は別途与えられ，1 行目からケースが表される（Figure1.4）
  - 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
  - R のモダンな方法では，データの加工や統計処理のプロセスをデータフレームの形で返すことが多い
  - 上記のような状態を **tidy**(読み：タイディー，意味：整然) と呼び，データ加工において理想的な形とされている

- オブジェクトに格納することで，別のデータフレームを作れる
- 列単位で取り出すとベクトルになる
- 本書では，データフレームの中でも表示に便利な tibble 形式を使う

```
# A tibble: 344 x 8
   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex     year
   <fct>   <fct>              <dbl>         <dbl>             <int>       <int> <fct>   <int>
 1 Adelie  Torgersen           39.1          18.7               181        3750 male     2007
 2 Adelie  Torgersen           39.5          17.4               186        3800 female   2007
 3 Adelie  Torgersen           40.3          18                 195        3250 female   2007
 4 Adelie  Torgersen           NA            NA                  NA          NA NA       2007
 5 Adelie  Torgersen           36.7          19.3               193        3450 female   2007
 6 Adelie  Torgersen           39.3          20.6               190        3650 male     2007
 7 Adelie  Torgersen           38.9          17.8               181        3625 female   2007
 8 Adelie  Torgersen           39.2          19.6               195        4675 male     2007
 9 Adelie  Torgersen           34.1          18.1               193        3475 NA       2007
10 Adelie  Torgersen           42            20.2               190        4250 NA       2007
# ... with 334 more rows
```

図1.3 R のデータフレーム (tibble 形式)



	A	B	C	D	E
1	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm
2	Adelie	Torgersen	39.1	18.7	181
3	Adelie	Torgersen	39.5	17.4	186
4	Adelie	Torgersen	40.3	18	195
5	Adelie	Torgersen			
6	Adelie	Torgersen	36.7	19.3	193
7	Adelie	Torgersen	39.3	20.6	190
8	Adelie	Torgersen	38.9	17.8	181
9	Adelie	Torgersen	39.2	19.6	195
10	Adelie	Torgersen	34.1	18.1	193
11	Adelie	Torgersen	42	20.2	190

図1.4 Excel 画面風なイメージ

- 本書では紙面の都合上，表示行数をしぼっているが，任意の行数を見たいときは `print( )` 関数で出力ごとに設定

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（CC0）

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

palmerpenguins::penguins
```

```
## # 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>
```

- tibble 形式のデータフレームの出力の見方
  - 出力の最上段にある `A tibble: 344 x 8` で，tibble 形式のデータフレーム，344 行 × 8 列という情報が分かる
  - flipper_length_のように，長い変数名は で省略して表示される
  - 変数名の下の行にある `<fct>`, `<dbl>`, `<int>` は変数の型を示し，それぞれ因子型，数値型，整数型であることを示している。詳しくは5.3で説明する
  - 下から 2 行目にある `... with 341 more rows, and 3 more variables:` で，さらに 341 行と 3 列が非表示であることが分かる
  - 非表示になった変数名は最下部に表示される
- tibble 型のデフォルトは最初の 10 行のみ表示され，本書では最初の 3 行のみに絞っているが，10 行を越えて表示させたい場合は，`print( )` 関数を使う

```
palmerpenguins::penguins %>%
  print(n = 15)
```

```
## # A tibble: 344 x 8
##    species island   bill_length_mm bill_depth_mm flipper_length_~
##    <fct>   <fct>             <dbl>         <dbl>            <int>
##  1 Adelie  Torgers~           39.1          18.7              181
##  2 Adelie  Torgers~           39.5          17.4              186
##  3 Adelie  Torgers~           40.3          18                195
```

```
##  4 Adelie  Torgers~           NA            NA                NA
##  5 Adelie  Torgers~           36.7          19.3             193
##  6 Adelie  Torgers~           39.3          20.6             190
##  7 Adelie  Torgers~           38.9          17.8             181
##  8 Adelie  Torgers~           39.2          19.6             195
##  9 Adelie  Torgers~           34.1          18.1             193
## 10 Adelie  Torgers~           42            20.2             190
## 11 Adelie  Torgers~           37.8          17.1             186
## 12 Adelie  Torgers~           37.8          17.3             180
## 13 Adelie  Torgers~           41.1          17.6             182
## 14 Adelie  Torgers~           38.6          21.2             191
## 15 Adelie  Torgers~           34.6          21.1             198
## # ... with 329 more rows, and 3 more variables:
## #   body_mass_g <int>, sex <fct>, year <int>
```

1.7 %>% （パイプ演算子）

- 名前は**パイプ**で発音は"*and then*"（参照）
- コードを読みやすくするための便利な機能を持つ演算子。初めてみた人は全然わからないと思うが，この本を読んでコードを書きはじめてみたらこれなしではいられなくなるくらいお世話になると思う
  - 主に使用が想定される場面でざっくりいうと，「このデータフレームに対して`%>%` の後にある関数を適用する」という機能
  - 具体的な使用法は2.2.1で解説
- RStudio のショートカットは `Ctrl + Shift + M`(Mac は `Cmd + Shift + M`)。たぶん，RStudio 以外でもこのショートカット押してしまうぐらい中毒性がある
- R version 4.1 からは `|>` が大体同じ機能を持つ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 `|>` に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
  - 現時点ではデータフレームを第一引数へ渡す形式でない関数の場合（回帰分析の `lm( )` など），工夫が必要な場合があるようなので，本書では`%>%` を使用

第 2 章

列（変数）を選ぶ：select

- `dplyr: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
- `palmerpenguins::penguins` というのは，「palmerpenguins パッケージの::penguins データ」という意味

```
library(tidyverse)

df <-
  palmerpenguins::penguins

# データの表示
df
```

```
## # 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
```

```
rm(df2)
```

2.2.1 【補足】%>% の意味

- 1.7で説明したパイプ演算子の実例を解説する
- 基本的に select( ) を始めとしたモダンな R の処理は，以下のように第一引数にデータフレームを指定する

```
select(df, bill_length_mm)
```

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

- %>% の役割は，その左側にあるものを右側の関数の第一引数に入れる，ということなので，第一引数にデータフレームが来ることが決まっていれば，常に次のようにかける
- このようにすると複雑な処理を重ねていく場合も，コードの可読性が高まるので，データラングリングの過程で有用

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

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

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

2.2.2 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
```

2.2.3 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
- select( ) は文字の変数名を与えても動く

```
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ベクトルとして代入した方が読みやすい
  - あらかじめ作成したベクトルとして代入するときは，all_of( ) で囲む必要がある
  - 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ようにする
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
```

2.4 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　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

2.5.1　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"
```

2.5.2　選択した変数名を取得

- ベクトルとしてオブジェクトに格納

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

bill_vars
```

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

2.5.3 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必要なものを選んでそのまま `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")
```

- この出力から必要な変数を選んでコピペができる
  - `names()` で出てくるのと違い，, がついているのが地味にうれしい
- " " 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力技も

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」
- 分かりやすい列名にすることだけでなく，本書の範囲を超えるが複数データの連結や同時処理関連で重要な役割を果たす

3.1 基本

- まずはデータにどういう変数名があるかの確認

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

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

- 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

```
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>
```

- 複数の変数名を変更する場合は，rename() の中に，でつなげていく
  – でもたくさんある場合に一つ一つ書いていくのは大変

```
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
- 語の置き換えに stringr::str_replace() を使う
  - 第一引数に対して，その次の文字列をその後の文字列に置換する（ここでは”bill” → “くちばし”）
- 適用したい関数の中にある.x の部分に，その後選ぶ変数が入っていく (なおこのような単純な場合は. だけでも動く）
- この場合適用したい関数の前には ~（チルダ）が必ずつく
  - この部分の理解は今すぐできなくても使えるが，キーワードだけ示しておくと，無名関数 (anonymous function) という処理をしている

```
df %>%
  rename_with(~str_replace(.x, "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
  - 第一引数についてその次に来る文字列を取り除く

```
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るが数が多いと大変

```
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 以外すべてなので，year に ! 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】

- 変数を選ぶときに該当する単語を持つ変数を選びたければ, ヘルパー関数 matches( ) で正規表現を使って柔軟に選べる

```
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>
```

第 4 章

行（ケース）を選ぶ：filter

- 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
- データフレームとして出力した結果を限定して見るときに使うことが多い気がする

4.1 使用データ

- dplyr::starwars データを使用
  - スターウォーズのキャラクターのデータ。filter のヘルプでも例に使用されている
  - 身長や質量 (mass) の連続量データに加え，色や種 (species) など豊富なカテゴリを持つ変数がある

```
starwars
```

```
## # A tibble: 87 x 14
##   name    height  mass hair_color skin_color eye_color birth_year
##   <chr>    <int> <dbl> <chr>      <chr>      <chr>          <dbl>
## 1 Luke S~    172    77 blond      fair       blue              19
## 2 C-3PO      167    75 <NA>       gold       yellow           112
## 3 R2-D2       96    32 <NA>       white, bl~ red               33
## # ... with 84 more rows, and 7 more variables: sex <chr>,
## #   gender <chr>, homeworld <chr>, species <chr>, films <list>,
```

```
## #   vehicles <list>, starships <list>
```

- 例示しやすくするため種を先頭にしたデータを作成

```
df_st <-
  starwars %>%
  select(species, name:homeworld)
```

4.2 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
- 例：種（species）が”Droid” のケースのみ選ぶ
  - イコールを表すときは = を 2 つつなげる

```
df_st %>%
  filter(species == "Droid")
```

```
## # A tibble: 6 x 11
##   species name   height  mass hair_color skin_color  eye_color
##   <chr>   <chr>   <int> <dbl> <chr>      <chr>       <chr>
## 1 Droid   C-3PO     167    75 <NA>       gold        yellow
## 2 Droid   R2-D2      96    32 <NA>       white, blue red
## 3 Droid   R5-D4      97    32 <NA>       white, red  red
## 4 Droid   IG-88     200   140 none       metal       red
## 5 Droid   R4-P17     96    NA none       silver, red red, blue
## 6 Droid   BB8        NA    NA none       none        black
## # ... with 4 more variables: birth_year <dbl>, sex <chr>,
## #   gender <chr>, homeworld <chr>
```

- 例：身長（height）が 200 以上のケースのみ選ぶ

```
df_st %>%
  filter(height >= 200)
```

```
## # A tibble: 11 x 11
##    species  name    height  mass hair_color skin_color  eye_color
##    <chr>    <chr>    <int> <dbl> <chr>      <chr>       <chr>
##  1 Human    Darth ~    202   136 none       white       yellow
##  2 Wookiee  Chewba~    228   112 brown      unknown     blue
##  3 Droid    IG-88      200   140 none       metal       red
##  4 Gungan   Roos T~    224    82 none       grey        orange
##  5 Gungan   Rugor ~    206    NA none       green       orange
##  6 Quermian Yarael~    264    NA none       white       yellow
##  7 Kaminoan Lama Su    229    88 none       grey        black
##  8 Kaminoan Taun We    213    NA none       grey        black
##  9 Kaleesh  Grievo~    216   159 none       brown, whi~ green, y~
## 10 Wookiee  Tarfful    234   136 brown      brown       blue
## 11 Pau'an   Tion M~    206    80 none       grey        black
## # ... with 4 more variables: birth_year <dbl>, sex <chr>,
## #   gender <chr>, homeworld <chr>
```

- ～以外を表すときは! をつけ，この場合は = は 1 つでよい
- 例：種が Human のケース**以外**を選ぶ

```
df_st %>%
  filter(species != "Human")
```

```
## # A tibble: 48 x 11
##   species name  height  mass hair_color skin_color  eye_color
##   <chr>   <chr>  <int> <dbl> <chr>      <chr>       <chr>
## 1 Droid   C-3PO    167    75 <NA>       gold        yellow
## 2 Droid   R2-D2     96    32 <NA>       white, blue red
## 3 Droid   R5-D4     97    32 <NA>       white, red  red
## # ... with 45 more rows, and 4 more variables: birth_year <dbl>,
## #   sex <chr>, gender <chr>, homeworld <chr>
```

4.2.1 欠損値（NA）の扱い

- 現実のデータでは，データが入手できない対象が発生することも多く，変数の値の中にデータがない変数の行（excel 風にいうとセル）が発生する
- R ではデータのない部分，いわゆる欠損値は NA で表される
- 例：種が NA のケースを選ぶ
  - NA かどうかを判定する論理式は is.na( )

```
df_st %>%
  filter(is.na(species))
```

```
## # A tibble: 4 x 11
##   species name       height  mass hair_color skin_color eye_color
##   <chr>   <chr>       <int> <dbl> <chr>      <chr>      <chr>
## 1 <NA>    Ric Olie      183    NA brown      fair       blue
## 2 <NA>    Quarsh Pa~    183    NA black      dark       brown
## 3 <NA>    Sly Moore     178    48 none       pale       white
## 4 <NA>    Captain P~     NA    NA unknown    unknown    unknown
## # ... with 4 more variables: birth_year <dbl>, sex <chr>,
## #   gender <chr>, homeworld <chr>
```

4.3 複数条件

- 例：種が Droid または Human のケースを選ぶ
  - | は「または」を表す

```
df_st %>%
  filter(species == "Droid" | species == "Human")
```

```
## # A tibble: 41 x 11
##   species name       height  mass hair_color skin_color eye_color
##   <chr>   <chr>       <int> <dbl> <chr>      <chr>      <chr>
## 1 Human   Luke Skyw~    172    77 blond      fair       blue
## 2 Droid   C-3PO         167    75 <NA>       gold       yellow
```

```
## 3 Droid   R2-D2          96    32 <NA>       white, bl~ red
## # ... with 38 more rows, and 4 more variables: birth_year <dbl>,
## #   sex <chr>, gender <chr>, homeworld <chr>
```

- 種が Droid かつ身長が 100 未満のケースのみ選ぶ
  - & は「かつ」を表す

```
df_st %>%
  filter(species == "Droid" & height < 100)
```

```
## # A tibble: 3 x 11
##   species name   height  mass hair_color skin_color  eye_color
##   <chr>   <chr>   <int> <dbl> <chr>      <chr>       <chr>
## 1 Droid   R2-D2      96    32 <NA>       white, blue red
## 2 Droid   R5-D4      97    32 <NA>       white, red  red
## 3 Droid   R4-P17     96    NA none       silver, red red, blue
## # ... with 4 more variables: birth_year <dbl>, sex <chr>,
## #   gender <chr>, homeworld <chr>
```

4.3.1 【効率化】

- 選びたいものが多くなると，書くのが大変。“species ==” とかをいちいち書きたくない
- 例: 種で”Aleena” または “Dug” または “Yoda’s species” を選びたいとき

```
df_st %>%
  filter(species == "Aleena" | species == "Dug" | species == "Yoda's species")
```

```
## # A tibble: 3 x 11
##   species    name    height  mass hair_color skin_color eye_color
##   <chr>      <chr>    <int> <dbl> <chr>      <chr>      <chr>
## 1 Yoda's sp~ Yoda        66    17 white      green      brown
## 2 Dug        Sebulba    112    40 none       grey, red  orange
## 3 Aleena     Ratts ~     79    15 none       grey, blue unknown
## # ... with 4 more variables: birth_year <dbl>, sex <chr>,
```

```
## #   gender <chr>, homeworld <chr>
```

- %in% で解決

```
df_st %>%
  filter(species %in% c("Aleena", "Dug", "Yoda's species"))
```

```
## # A tibble: 3 x 11
##   species    name    height  mass hair_color skin_color eye_color
##   <chr>      <chr>    <int> <dbl> <chr>      <chr>      <chr>
## 1 Yoda's sp~ Yoda        66    17 white      green      brown
## 2 Dug        Sebulba    112    40 none       grey, red  orange
## 3 Aleena     Ratts ~     79    15 none       grey, blue unknown
## # ... with 4 more variables: birth_year <dbl>, sex <chr>,
## #   gender <chr>, homeworld <chr>
```

- 例: 種で”Droid”, “Human” 以外を選びたいとき
  - この場合，& が必須

```
df_st %>%
  filter(species != "Droid" & species != "Human")
```

```
## # A tibble: 42 x 11
##   species name      height  mass hair_color skin_color  eye_color
##   <chr>   <chr>      <int> <dbl> <chr>      <chr>       <chr>
## 1 Wookiee Chewbacca    228   112 brown      unknown     blue
## 2 Rodian  Greedo       173    74 <NA>       green       black
## 3 Hutt    Jabba De~    175  1358 <NA>       green-tan,~ orange
## # ... with 39 more rows, and 4 more variables: birth_year <dbl>,
## #   sex <chr>, gender <chr>, homeworld <chr>
```

- 変数の前に! をつけるだけで省略できる

```
df_st %>%
  filter(!species %in% c("Droid", "Human"))
```

```
## # A tibble: 46 x 11
##   species name      height  mass hair_color skin_color  eye_color
##   <chr>   <chr>      <int> <dbl> <chr>      <chr>       <chr>
## 1 Wookiee Chewbacca    228   112 brown      unknown     blue
## 2 Rodian  Greedo       173    74 <NA>       green       black
## 3 Hutt    Jabba De~    175  1358 <NA>       green-tan,~ orange
## # ... with 43 more rows, and 4 more variables: birth_year <dbl>,
## #   sex <chr>, gender <chr>, homeworld <chr>
```

4.4 キーワードによる検索

- 手元で特定の名前の行のデータを見たいときに便利
- キーワード検索には，正規表現の結果を TRUE or FALSE で返す関数 `stringr::str_detect( )` を使う
- 例：変数 name に”Luke” を含む行を見たい

```
df_st %>%
  filter(str_detect(name, "Luke"))
```

```
## # A tibble: 1 x 11
##   species name       height  mass hair_color skin_color eye_color
##   <chr>   <chr>       <int> <dbl> <chr>      <chr>      <chr>
## 1 Human   Luke Skyw~    172    77 blond      fair       blue
## # ... with 4 more variables: birth_year <dbl>, sex <chr>,
## #   gender <chr>, homeworld <chr>
```

- 例：変数 name が”R” で始まる行を見たい
  - 正規表現で ^ はその次の文字から始まる文字列という意味

```
df_st %>%
  filter(str_detect(name, "^R"))
```

```
## # A tibble: 9 x 11
##   species name       height  mass hair_color skin_color eye_color
##   <chr>   <chr>       <int> <dbl> <chr>      <chr>      <chr>
```

```
## 1 Droid   R2-D2          96    32 <NA>       white, bl~ red
## 2 Droid   R5-D4          97    32 <NA>       white, red red
## 3 Gungan  Roos Tarp~    224    82 none       grey       orange
## 4 Gungan  Rugor Nass    206    NA none       green      orange
## 5 <NA>    Ric Olie      183    NA brown      fair       blue
## 6 Aleena  Ratts Tye~     79    15 none       grey, blue unknown
## 7 Droid   R4-P17          96    NA none       silver, r~ red, blue
## 8 Human   Raymus An~    188    79 brown      light      brown
## 9 Human   Rey            NA    NA brown      light      hazel
## # ... with 4 more variables: birth_year <dbl>, sex <chr>,
## #   gender <chr>, homeworld <chr>
```

- 例：変数 name が”Y” または”L” で始まる行を見たい
  - 正規表現で「または」は" " の中に入れる

```
df_st %>%
  filter(str_detect(name, "^Y|^L"))
```

```
## # A tibble: 8 x 11
##   species   name     height  mass hair_color skin_color eye_color
##   <chr>     <chr>     <int> <dbl> <chr>      <chr>      <chr>
## 1 Human     Luke Sk~    172  77   blond      fair       blue
## 2 Human     Leia Or~    150  49   brown      light      brown
## 3 Yoda's s~ Yoda         66  17   white      green      brown
## 4 Human     Lando C~    177  79   black      dark       brown
## 5 Human     Lobot       175  79   none       light      blue
## 6 Quermian  Yarael ~    264  NA   none       white      yellow
## 7 Mirialan  Luminar~    170  56.2 black      yellow     blue
## 8 Kaminoan  Lama Su     229  88   none       grey       black
## # ... with 4 more variables: birth_year <dbl>, sex <chr>,
## #   gender <chr>, homeworld <chr>
```

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
- 効率化のための `across()`

5.1 データ読み込み

- `psychTools` パッケージに入っている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または置き換える関数
- 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，個々のケースの値の差分を新しく列として追加する例を示す

```
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章を参照）
- 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（double-precision）*<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

```
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
```

- tibble 形式でない普通のデータフレームでも，最後に `glimpse()` で出力することで型を確認可能

```
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~
```

- `glimpse()` の結果はデータフレームの表示が行列入れ替わっており，変数の情報

が行ごとに出力される

- 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

5.3.2 【効率化】複数の変数に対し一度の指定で実行

- 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
  - かつては `mutate_at( )`, `mutate_if( )`, `mutate_all( )` など別々の関数だったが，`across( )` の登場で統一的に `mutate( )` 内で扱えるようになった

```
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
```

- 同じく2.2.3で解説した文字型の変数名も指定に使える

```
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（3.2.1参照）
  - list 内の関数 `()` 内に`.x` が必要。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
- 例：gender と education の型を因子型に変換し，変換前後で変わっているかどうか確認

```
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

- 変数の四則演算の式を入れれば合計得点として計算された列をデータフレームに追加できる

```
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
```

- 別解として，変数の逆転項目を反映させた後に，5.9 で異なるやり方で合計した例を解説する
  - 項目数が多い場合などはこちらの方が効率化できる場合も

5.5.1 足し上げる変数に欠損値があるとどうなるか

- 欠損値については4.2.1参照
- 合計得点の計算の場合，対象となる変数の内 1 つでも NA があれば合計点も NA となる

```
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　変数の値を数値から文字列に変える

- 一度因子型に変換してから forcats パッケージの fct_recode( ) 関数を使うと簡単
- 例：gender の値 1,2 をそれぞれ male, female という文字に置き換える
- ちゃんと変換の対応がついているかどうかを dplyr パッケージの count( ) 関数で確認
  - 適切に変換されていなければ，1 = male, 2 = female 以外の組み合わせも発生するため
  - count( ) の強みは，出力がデータフレームで出てくる点なので，結果が扱いやすい

```
df_bfi %>%
  mutate(gender = factor(gender),
         gender_c = fct_recode(gender,
                               male   = "1",
                               female = "2")) %>%
  count(gender, gender_c)
```

```
## # A tibble: 2 x 3
##   gender gender_c     n
##   <fct>  <fct>    <int>
## 1 1      male       919
## 2 2      female    1881
```

5.7 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
```

5.7.1 【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)
```

5.8　逆転項目を作る

- 心理尺度などの場合，質問内容に対する回答選択肢の意味が，項目間で逆になるように設定されることがあり，合計点などを作る際に尺度の意味を適切に表すように，取りうる数値の範囲内で値を入れ替える作業が発生することがある
  - たとえば，感情の状態を項目を合計してたずねる尺度で，「いつも楽しい」という項目と，「いつも悲しい」という聞き方をしていたら，それぞれの回答を得点化したときに意味が反対になるため，同じ方向になるようにする必要がある
    例：「1. まったくあてはまらない ←→ 6. 非常にあてはまる」のルールを「いつも悲しい」項目に適用し，1 の回答を 6 に置き換えれば，「いつも楽しい」とポジティブな方向で点数の意味がそろう

5.8.1　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

5.8.2　逆転：recode

- `dplyr::recode()` を使用
- 対象の変数を `recode( )` の第一引数に，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とされている
  - また，下記のように考慮すべき点があるから，後述の5.8.3【別解】を使う方がよいかもしれない
- 値の指定で考慮すべき点
  - old の数値はで囲む必要がある

– new の数値に L がつくのは，型を整数のままにするため

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

5.8.2.1 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
```

5.8.2.2 【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
  - これで対象となる across( ) 内変数に _r の接尾辞が付いた逆転項目が追加される

```
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>
```

5.8.3 【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
```

5.8.3.1 【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
```

5.8.3.2 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)))
```

5.9　【別解】合計点の作成

- base::rowSums( )
  - データフレームで行の単位で総計するので，行（ケース）ごとに合計点を作成できる
- まず総計の対象となる変数名を2.2.3で解説したように文字ベクトルにしていく
- rowSums( ) の中で across( ) が使えるので，あとは定義した項目のオブジェクトを指定していくだけ

```
# 合計する項目の定義

Ag <-
df_bfi %>%
  select(A1_r, A2:A5) %>%
  names()

Co <-
df_bfi %>%
  select(C1:C3, C4_r, C5_r) %>%
  names()

Ex <-
  df_bfi %>%
  select(E1_r, E2_r, E3:E5) %>%
  names()

Ne <-
df_bfi %>%
  select(N1:N5) %>%
  names()

Op <-
```

```
df_bfi %>%
  select(O1, O2_r, O3, O4, O5_r) %>%
  names()



df_bfi <-
  df_bfi %>%
  mutate(
    Agree = rowSums(across(all_of(Ag))),
    Conscientious = rowSums(across(all_of(Co))),
    Extraversion = rowSums(across(all_of(Ex))),
    Neuroticism = rowSums(across(all_of(Ne))),
    Openness = rowSums(across(all_of(Op)))
    )
```

5.9.1 【確認】

- 変数の中に NA が入る場合は合計も NA になる。
- rowSums(across(all_of(Op)), **na.rm = TRUE**) と引数を追加すれば，NA を無視して合計できる

```
df_bfi %>% select(all_of(Ag), Agree)
```

```
## # A tibble: 2,800 x 6
##    A1_r    A2    A3    A4    A5 Agree
##   <dbl> <int> <int> <int> <int> <dbl>
## 1     5     4     3     4     4    20
## 2     5     4     5     2     5    21
## 3     2     4     5     4     4    19
## # ... with 2,797 more rows
```

```
df_bfi %>% select(all_of(Ex), Extraversion) %>%
  filter(is.na(Extraversion))
```

```
## # A tibble: 87 x 6
```

```
##    E1_r  E2_r    E3    E4    E5 Extraversion
##   <dbl> <dbl> <int> <int> <int>        <dbl>
## 1     2     4    NA     4     3           NA
## 2     6     6     4     4    NA           NA
## 3     2    NA     3     2     3           NA
## # ... with 84 more rows
```

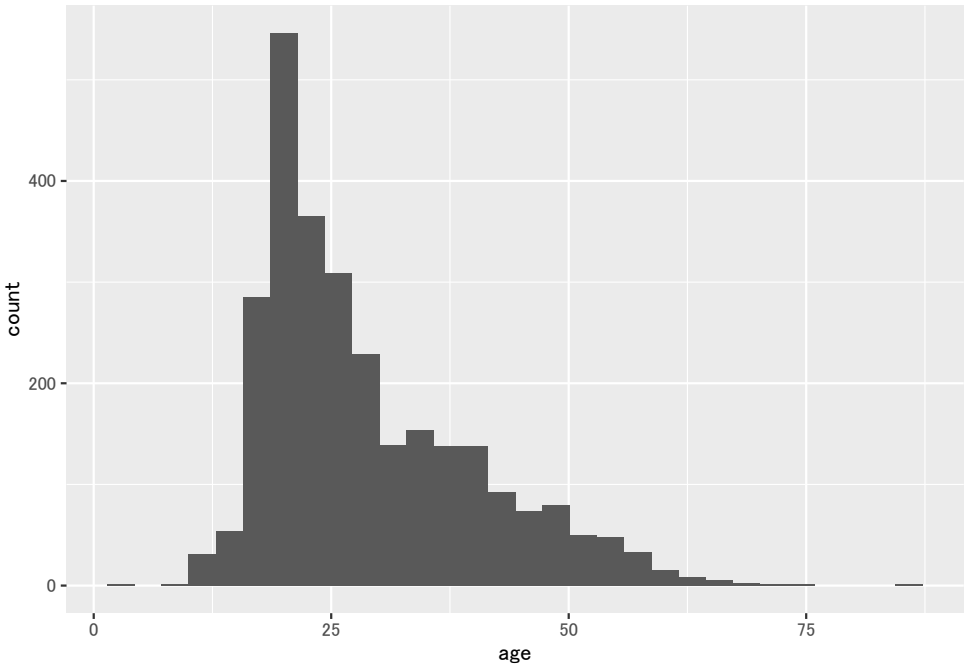
- rowSums( ) を base::rowMeans( ) に変えれば平均値も計算できる

5.10　連続変数をカテゴリに区分する

5.10.1　分布の把握

- 変数 age のヒストグラムを描き，分布を確認する
- グラフ作成パッケージ ggplot2 でどんなグラフが作れるかは著者作成の辞書参照
  - ggplot2 の辞書

```
ggplot(df_bfi) +            # ここにデータフレーム
  geom_histogram(aes(age)) # aes( ) の中に対象変数
```



- さっと中央値だけ見たいのであれば，従来の R の書き方が早い
- 役に立つ場面が多いので，慣れたら従来の書き方を学んでおくとよい
  –
- データフレーム $ 変数名と指定することで変数として扱える

```
# 中央値
median(df_bfi$age)
```

```
## [1] 26
```

```
# モダンな方法だと少し長くなる
# df_bfi %>%
#   summarise(median(age))
```

5.10.2　2 区分

- `dplyr::if_else( )` で条件式（TRUE または FALSE を返すもの）によって値を 2 区分する

- 構造：if_else(条件式, TRUE の場合の値, FALSE の場合の値)
  - TRUE の場合の値, FALSE の場合の値はそれぞれ文字型を入れることもできる（例:“27 歳以上”, “27 歳未満”）

```
res_age2 <-
  df_bfi %>%
  mutate(age2 = if_else(age >= 27, 1, 0)) %>%
  select(age, age2)

res_age2 %>% count(age2)
```

```
## # A tibble: 2 x 2
##    age2     n
##   <dbl> <int>
## 1     0  1495
## 2     1  1305
```

5.10.2.1 確認

- age >= 27 が 1, 27 未満が 0 にコーディングされているか filter( ) で限定して確認

```
res_age2 %>%
  filter(age >= 20 & age <= 30) %>%
  count(age, age2) %>%
  print(n = 11)
```

```
## # A tibble: 11 x 3
##      age  age2     n
##    <int> <dbl> <int>
##  1    20     0   212
##  2    21     0   144
##  3    22     0   122
##  4    23     0   138
##  5    24     0   105
##  6    25     0   113
```

```
##  7    26     0    99
##  8    27     1    97
##  9    28     1    86
## 10    29     1    78
## 11    30     1    65
```

- 念のため最初 3 行（1-3 行目）と最後 3 行（n-2 行目から n 行目）も確認
- `dplyr::slice( )` で 1:3 行目と最後の 3 行を表示させる

```
res_age2 %>%
  count(age, age2) %>%
  slice(1:3, (n()-2):n())
```

```
## # A tibble: 6 x 3
##      age  age2     n
##    <int> <dbl> <int>
## 1     3     0     1
## 2     9     0     1
## 3    11     0     3
## 4    72     1     1
## 5    74     1     1
## 6    86     1     1
```

```
rm(res_age2)
```

5.10.3　3 区分以上

- 年齢層を 10 歳区切りでカテゴリ化

```
res_age6 <-
  df_bfi %>%
  mutate(age6 = case_when(
    age < 20               ~ "20 歳未満",
    age >= 20  & age < 30 ~ "20-29 歳",
    age >= 30  & age < 40 ~ "30-39 歳",
```

```
    age >= 40  & age < 50 ~ "40-49 歳",
    age >= 50  & age < 60 ~ "50-59 歳",
    age >= 60             ~ "60 歳以上"
  ))

# 確認するには以下のコードの最初の 2 行だけでよいが，出力が長いためランダムに 10 件抽出し age
res_age6 %>%
  count(age, age6) %>%
  slice_sample(n = 10) %>% # ランダムに 10 件抽出
  arrange(age)             # age をキーに行を昇順にソート
```

```
## # A tibble: 10 x 3
##      age age6          n
##    <int> <chr>     <int>
##  1    19 20歳未満    190
##  2    23 20-29歳     138
##  3    26 20-29歳      99
##  4    28 20-29歳      86
##  5    31 30-39歳      73
##  6    38 30-39歳      52
##  7    61 60歳以上      4
##  8    62 60歳以上      4
##  9    65 60歳以上      1
## 10    70 60歳以上      1
```

```
rm(res_age6)
```

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
- データを知るうえで要約作業は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

- summarise( ) の中に出力したい変数名を書き，= の後に計算する関数を入れる
- 例：bill_length_mm の平均値を算出する

```
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】

- 5.3.2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
- sum(!is.na(.x)) は，NA のない行の数を総計するので，平均値や SD の計算に用いた人数を取得できる

```
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】

- 上記の出力は横に長いため見にくい
- 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
- 下記コードの summarise( ) 部分の構造は前のチャンクと変数名以外同じ

```
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

- 自分で名づける関数名 <- function(引数){ 計算式やコード } で関数を定義できる
- 例：関数の引数に数値を入れると +1 した値を返す関数

```
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

- {{ }} は curly curly と読み，関数を作成するときに，代入先の変数名の場所を指定する時などに活躍
  - 下記の例の場合，{{ }} を外すと動かない
- 例：引数にデータフレーム（data）と変数（vars）を入れると平均値と SD と n を返す関数

```
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