R で学ぶ『完全独習統計学入門』

Sampo Suzuki, CC 4.0 BY-NC-SA

2022-05-19

Contents

List of Figures

List of Tables

はじめに

本資料は『完全独習統計学入門』小島寛之著[1]（以降、テキスト）の内容を R で計算する方法を紹介しています。基本的な統計量を求める際に必要となる関数などの解説が中心です。

テキストに記載されているデータの一部を CSV ファイルにして利用していますが、データ自体の著作権は原著作者にあります。また、CSV ファイルは著作権を侵害する意図で作成したものではありません。

想定読者

本書の想定読者は以下に該当する方です。

- R に関する基本的な知識を有している方
  - データフレーム型や因子型などの変数型の知識
  - 変数の代入・参照方法
  - `tidyverse` の基本的な知識
- R で統計量を求めてみたい方

本書では R のインストール方法や基本文法には触れません。

[1] https://www.diamond.co.jp/book/9784478820094.html

Chapter 1

度数分布表とヒストグラムでデータの特徴を浮き彫りにする

1.1　生データでは何もわからないから統計を使う

テキストの図表 1-1 のデータは 4 列 20 行からなるデータです。

```
## # A tibble: 20 x 4
##       X1    X2    X3    X4
##    <dbl> <dbl> <dbl> <dbl>
##  1   151   152   151   167
##  2   154   161   155   159
##  3   160   159   158   157
##  4   160   164   146   151
##  5   163   168   149   159
##  6   156   153   165   156
##  7   158   160   169   151
##  8   156   155   154   160
##  9   154   160   155   153
## 10   160   158   166   156
## 11   154   160   165   146
## 12   162   150   161   166
## 13   156   163   148   151
## 14   162   155   143   159
## 15   157   157   156   157
## 16   162   162   162   156
## 17   162   154   158   162
## 18   169   161   156   156
## 19   150   153   159   156
## 20   162   154   164   161
```

R では、このような形式のデータ（雑然データ）をそのまま使わずに整然データ（Tidy Data[1]）と呼ばれる形式に変換した方が効率的に処理できます。変換には **tidyverse** ファミリーを利用するのが効率的です。

```
library(tidyverse)

# ローカルにあるデータファイルを読み込む
x <- "../data/P16_図表 1-1 .csv" %>%
  readr::read_csv(col_names = FALSE, show_col_types = FALSE) %>%
  # 読み込んだデータを縦長形式（Tidy Data）に変換する
  tidyr::pivot_longer(cols = dplyr::starts_with("X"),
                      names_to = "name", values_to = "value") %>%
  # X1, X2, ... , X4, X1 の順で行方向に読み込まれるので列方向で整列する
  dplyr::arrange(name) %>%
  # 必要なデータ列（value）の列名を height に変更して取り出す
  dplyr::select(height = value)

x
```

```
## # A tibble: 80 x 1
##    height
##     <dbl>
##  1    151
##  2    154
##  3    160
##  4    160
##  5    163
##  6    156
##  7    158
##  8    156
##  9    154
## 10    160
## # ... with 70 more rows
```

このような形式に変換することで、様々な統計量を効率的に求めることができるようになります。

1.2 ヒストグラムを作る

テキストでは度数分布表を作成してからヒストグラムを描いています。その手順は

[1] https://ja.wikipedia.org/wiki/Tidy_data

1. 最大・最小値を求める
2. 階級を決める
3. 度数をカウントする
4. 相対度数を求める
5. 累積度数を求める
    - ここまでで度数分布表ができる
6. 度数分布表をもとにヒストグラムを描く

となっていますが、R では hist() 関数だけでヒストグラムを描くことができます。hist() 関数の引数はベクトル型に限定されますので、データフレーム型の場合は参照演算子（$）などでベクトル型データを取り出す必要があります。

```
hist(x = x$height)
```

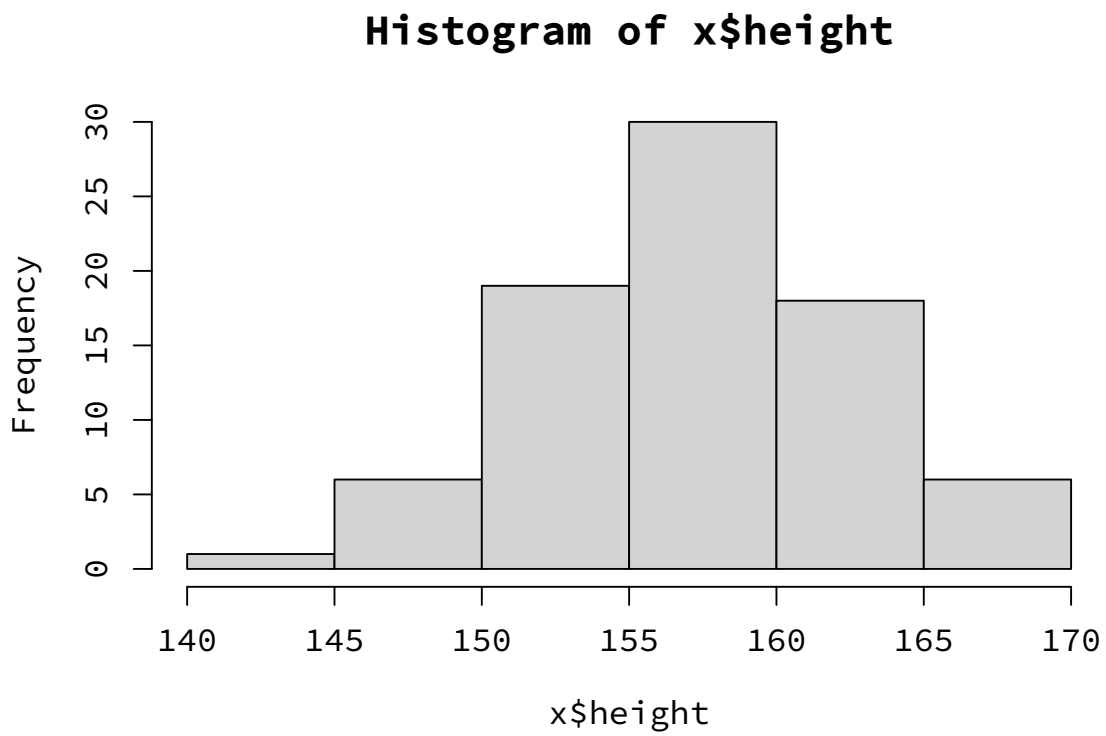


Figure 1.1: R の hist() 関数

hist() 関数はヒストグラムを描くために必要となる階級や度数、階級値などの情報を出力することも可能です。

```
hist(x = x$height, plot = FALSE)
```

```
## $breaks
## [1] 140 145 150 155 160 165 170
##
```

```
## $counts
## [1]  1  6 19 30 18  6
##
## $density
## [1] 0.0025 0.0150 0.0475 0.0750 0.0450 0.0150
##
## $mids
## [1] 142.5 147.5 152.5 157.5 162.5 167.5
##
## $xname
## [1] "x$height"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

Table 1.1: 主な出力の意味

出力名	意味	備考
$breaks	階級	デフォルトはスタージェスの公式* にもとづく区切り（幅）
$counts	度数	各階級に入るデータの個数
$density	密度	密度推定値
$mids	階級値	階級の中央値（階級の単純平均）

* 後述

表形式の度数分布表を作成する場合は以下のようにコード処理します。

```
x %>%
  # 階級を求めて、データを階級ごとに分ける
  dplyr::mutate(class = cut(height,
                            breaks = pretty(height, n = nclass.Sturges(height)),
                            include.lowest = FALSE, right = TRUE)) %>%
  # 階級ごとの度数を求める
  dplyr::count(class) %>%
  # 累積度数、相対度数、累積相対度数を求める
  dplyr::mutate(cumsum_n = cumsum(n),
                prop = prop.table(n), cumsum_prop = cumsum(prop)) %>%
  # 階級値を求める
  dplyr::mutate(class_value = as.character(class)) %>%
  tidyr::separate(class_value, into = c("l", "h"), sep = ",") %>%
  dplyr::mutate(l = as.integer(stringr::str_remove(l, "[[:punct:]]")) + 1L,
```

```
                h = as.integer(stringr::str_remove(h, "[[:punct:]]")),
                mids = (l + h) / 2L) %>%
  # 変数名を日本語にする
  dplyr::select(`階級` = class, `階級値` = mids,
                `度数` = n, `累積度数` = cumsum_n,
                `相対度数` = prop, `累積相対度数` = cumsum_prop)
```

```
## # A tibble: 6 x 6
##   階級        階級値  度数 累積度数 相対度数 累積相対度数
##   <fct>        <dbl> <int>    <int>    <dbl>          <dbl>
## 1 (140,145]    143      1        1   0.0125         0.0125
## 2 (145,150]    148      6        7   0.075          0.0875
## 3 (150,155]    153     19       26   0.238          0.325
## 4 (155,160]    158     30       56   0.375          0.7
## 5 (160,165]    163     18       74   0.225          0.925
## 6 (165,170]    168      6       80   0.075          1
```

以降の項は度数分布表の作成に必要な関数の使い方を紹介していますので、不要な方は演習問題まで読み飛ばしてください。

1.2.1 階級を求める

> 以降に出てくる `with()` 関数は、第一引数で指定するデータフレーム型変数内の変数名を第二引数内で参照演算子（`$`）を用いることなく参照できるようにする関数です。

階級を求めるには `pretty()` 関数を使います。

```
with(x, pretty(x = height, n = nclass.Sturges(height)))
```

```
## [1] 140 145 150 155 160 165 170
```

第一引数 `x` には階級を求めたいベクトル型データを第二引数 `n` には階級を区切りたい数（階級数）を指定します。ここでは、階級数 `n` にはスタージェスの公式から求めた階級数を指定しています。

ただし、必ずしも指定した階級数に分割される訳ではない点に注意してください。`x` で指定したデータが階級に収まるように適切な丸め処理を行いますので、指定した階級数とは異なる階級が求められることもあります。

スタージェスの公式は、階級数を k、データのサイズ（数）を n とした場合、下式で定義されます。

$$k = \lceil \log_2 n + 1 \rceil$$

スタージェスの公式は nclass.Sturgess() 関数として実装されています。

```
with(x, nclass.Sturges(height))
```

```
## [1] 8
```

スタージェスの公式はヒストグラムを平滑化し過ぎる傾向があると言われています。気になる場合はスコットの選択（nclass.scott()）やフリードマン＝ダイアコニスの選択（nclass.FD()）を試して見てください。

1.2.2 データを各階級に分類する

データを階級分けするには cut() 関数を使います。

```
with(x, cut(x = height, breaks = pretty(height, n = nclass.Sturges(height))))
```

```
##  [1] (150,155] (150,155] (155,160] (155,160] (160,165] (155,160] (155,160]
##  [8] (155,160] (150,155] (155,160] (150,155] (160,165] (155,160] (160,165]
## [15] (155,160] (160,165] (160,165] (165,170] (145,150] (160,165] (150,155]
## [22] (160,165] (155,160] (160,165] (165,170] (150,155] (155,160] (150,155]
## [29] (155,160] (155,160] (155,160] (145,150] (160,165] (150,155] (155,160]
## [36] (160,165] (150,155] (160,165] (150,155] (150,155] (150,155] (150,155]
## [43] (155,160] (145,150] (145,150] (160,165] (165,170] (150,155] (150,155]
## [50] (165,170] (160,165] (160,165] (145,150] (140,145] (155,160] (160,165]
## [57] (155,160] (155,160] (155,160] (160,165] (165,170] (155,160] (155,160]
## [64] (150,155] (155,160] (155,160] (150,155] (155,160] (150,155] (155,160]
## [71] (145,150] (165,170] (150,155] (155,160] (155,160] (155,160] (160,165]
## [78] (155,160] (155,160] (160,165]
## Levels: (140,145] (145,150] (150,155] (155,160] (160,165] (165,170]
```

第一引数 x には階級分けの対象となるベクトル型変数を第二引数 breaks には前項で求めた階級を指定します。breaks 引数には任意の階級、例えば breaks = c(140, 155, 170) のように指定することも可能です。

階級の境界値はどちらに含まれるのか？

cut() 関数の出力は、(140,145] (145,150] のように階級間で同じ数値、この場合は 145 が含まれます。では、145 はどちらの階級に含まれるのでしょうか？答えは、添えられている括弧にあります。

(140,145]

は「140 を超えて 145 以下」となりますので、次の

(145,150]

は同様に「145 を超えて 150 以下」となります。階級間の境界値である 145 は (140, 145] 側に入ります。

(は「超えて」（境界値を含まない）、] は「以下」（境界値を含む）

で、逆向きの場合は

) は「未満」（境界値を含まない）、[は「以上」（境界値を含む）

となります。hist() 関数の階級も同様です。

境界値をどちらに含めるかは include.lowest 引数と right 引数で指定できます。

option	right = TRUE	right = FALSE
include.lowest = TRUE	[l,u] ... (l,u]	[l,u) ... [l,u]
include.lowest = FALSE	(l,u] ... (l,u]*	[l,u) ... [l,u)

* デフォルト

例えばデータの最大値と最小値を境界値として含む階級を指定した場合、include.lowest = FALSE, right = FALSE と指定すると最大値が階級に含まれなくなります。

```
with(x, cut(height, breaks = c(143, 155, 169),
            include.lowest = FALSE, right = FALSE))
```

```
##  [1] [143,155) [143,155) [155,169) [155,169) [155,169) [155,169) [155,169)
##  [8] [155,169) [143,155) [155,169) [143,155) [155,169) [155,169) [155,169)
## [15] [155,169) [155,169) [155,169) <NA>     [143,155) [155,169) [143,155)
## [22] [155,169) [155,169) [155,169) [155,169) [143,155) [155,169) [155,169)
## [29] [155,169) [155,169) [155,169) [143,155) [155,169) [155,169) [155,169)
```

```
## [36] [155,169) [143,155) [155,169) [143,155) [143,155) [143,155) [155,169)
## [43] [155,169) [143,155) [143,155) [155,169) <NA>     [143,155) [155,169)
## [50] [155,169) [155,169) [155,169) [143,155) [143,155) [155,169) [155,169)
## [57] [155,169) [155,169) [155,169) [155,169) [155,169) [155,169) [155,169)
## [64] [143,155) [155,169) [155,169) [143,155) [155,169) [143,155) [155,169)
## [71] [143,155) [155,169) [143,155) [155,169) [155,169) [155,169) [155,169)
## [78] [155,169) [155,169) [155,169)
## Levels: [143,155) [155,169)
```

逆に include.lowest = FALSE, right = TRUE と指定すると最小値が階級に含まれなくなります。

```
with(x, cut(height, breaks = c(143, 155, 169),
            include.lowest = FALSE, right = TRUE))
```

```
##  [1] (143,155] (143,155] (155,169] (155,169] (155,169] (155,169] (155,169]
##  [8] (155,169] (143,155] (155,169] (143,155] (155,169] (155,169] (155,169]
## [15] (155,169] (155,169] (155,169] (155,169] (143,155] (155,169] (143,155]
## [22] (155,169] (155,169] (155,169] (155,169] (143,155] (155,169] (143,155]
## [29] (155,169] (155,169] (155,169] (143,155] (155,169] (143,155] (155,169]
## [36] (155,169] (143,155] (155,169] (143,155] (143,155] (143,155] (143,155]
## [43] (155,169] (143,155] (143,155] (155,169] (155,169] (143,155] (143,155]
## [50] (155,169] (155,169] (155,169] (143,155] <NA>     (155,169] (155,169]
## [57] (155,169] (155,169] (155,169] (155,169] (155,169] (155,169] (155,169]
## [64] (143,155] (155,169] (155,169] (143,155] (155,169] (143,155] (155,169]
## [71] (143,155] (155,169] (143,155] (155,169] (155,169] (155,169] (155,169]
## [78] (155,169] (155,169] (155,169]
## Levels: (143,155] (155,169]
```

1.2.3 度数を求める

度数を求めるには各階級の数を数えます。数を数えるには table() 関数や dplyr::count() 関数を使います。table() 関数はベクトル型を対象に、dplyr::count() 関数はデータフレーム型を対象として個数をカウントします。用途によって使い分けてください。

```
with(x, table(cut(height, breaks = pretty(height, n = nclass.Sturges(height)))))
```

```
##
## (140,145] (145,150] (150,155] (155,160] (160,165] (165,170]
##         1         6        19        30        18         6
```

```
x %>%
  # 階級を求めて、データを階級ごとに分ける
  dplyr::mutate(class = cut(height,
                            breaks = pretty(height, n = nclass.Sturges(height)))) %
  # 階級ごとの度数を求める
  dplyr::count(class)
```

```
## # A tibble: 6 x 2
##   class          n
##   <fct>      <int>
## 1 (140,145]      1
## 2 (145,150]      6
## 3 (150,155]     19
## 4 (155,160]     30
## 5 (160,165]     18
## 6 (165,170]      6
```

1.2.4 累積度数、相対度数、累積相対度数を求める

累積度数（度数の累積和）を求めるには cumsum() 関数を使います。

```
with(x, cumsum(table(cut(height, breaks = pretty(height, n = nclass.Sturges(height)
```

```
## (140,145] (145,150] (150,155] (155,160] (160,165] (165,170]
##         1         7        26        56        74        80
```

相対度数を求めるには prop.table() 関数を使います。

```
with(x, prop.table(table(cut(height, breaks = pretty(height, n = nclass.Sturges(he
```

```
##
## (140,145] (145,150] (150,155] (155,160] (160,165] (165,170]
##    0.0125    0.0750    0.2375    0.3750    0.2250    0.0750
```

累積相対度数は相対度数の累積和ですので、prop.table() 関数の結果を cumsum() 関数に渡すことで求めることができます。

```
with(x, cumsum(prop.table(table(cut(height, breaks = pretty(height, n = nclass.Stu
```

```
## (140,145] (145,150] (150,155] (155,160] (160,165] (165,170]
##    0.0125    0.0875    0.3250    0.7000    0.9250    1.0000
```

1.2.5 階級値を求める

階級値は階級の文字列から下端と上端の境界値を抜き出し、下式で単純平均として求めています。

$$\frac{(\text{下端}+1)+\text{上端}}{2}$$

手順としては以下のようにしています。

1. as.character() 関数で階級（class）を文字型に変換し、新しい変数（class_value）を作成する
2. tidyr::separate() 関数で文字型の変数（class_value）を, の前後で二つの変数（l, h）に分割する
3. stringr::str_remove 関数で二つの変数（l, h）から (や] を取り除き as.numeric() 関数で文字列から数値に変換する
4. 数値に変換された二つの変数から（l, h）平均値を求める

```
x %>%
  # 階級を求めて、データを階級ごとに分ける
  dplyr::mutate(class = cut(height,
                            breaks = pretty(height, n = nclass.Sturges(height)),
                            include.lowest = FALSE, right = TRUE)) %>%
  # 階級ごとの度数を求める
  dplyr::count(class) %>%
  # 階級値を求める
  dplyr::mutate(class_value = as.character(class)) %>%
  tidyr::separate(class_value, into = c("l", "h"), sep = ",") %>%
  dplyr::mutate(l = as.numeric(stringr::str_remove(l, "[[:punct:]]")),
                h = as.numeric(stringr::str_remove(h, "[[:punct:]]"))) %>%
  dplyr::mutate(mids = ((l + 1) + h) / 2)
```

```
## # A tibble: 6 x 5
##   class         n     l     h  mids
##   <fct>     <int> <dbl> <dbl> <dbl>
## 1 (140,145]     1   140   145   143
## 2 (145,150]     6   145   150   148
## 3 (150,155]    19   150   155   153
## 4 (155,160]    30   155   160   158
## 5 (160,165]    18   160   165   163
## 6 (165,170]     6   165   170   168
```

1.3 練習問題

テキスト P23 にあるデータで度数分布表とヒストグラムを作成する。

```
## # A tibble: 10 x 8
##       X1    X2    X3    X4    X5    X6    X7    X8
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1    48    54    47    50    53    43    45    43
##  2    44    47    58    46    46    63    49    50
##  3    48    43    46    45    50    53    51    58
##  4    52    53    47    49    45    42    51    49
##  5    58    54    45    53    50    69    44    50
##  6    58    64    40    57    51    69    58    47
##  7    62    47    40    60    48    47    53    47
##  8    52    61    55    55    48    48    46    52
##  9    45    38    62    47    55    50    46    47
## 10    55    48    50    50    54    55    48    50
```

解答例

まず、処理しやすいように下記のように対象データを変形し df というデータフレーム型の変数に格納しておきます。

```
df
```

```
## # A tibble: 80 x 1
##    weight
##     <int>
```

```
##  1     48
##  2     44
##  3     48
##  4     52
##  5     58
##  6     58
##  7     62
##  8     52
##  9     45
## 10     55
## # ... with 70 more rows
```

度数分布表を身長の場合と同じ要領で作成します。

```
df %>%
  # 階級を求めて、データを階級ごとに分ける
  dplyr::mutate(class = cut(weight,
                            breaks = pretty(weight, n = nclass.Sturges(weight)),
                            include.lowest = FALSE, right = TRUE)) %>%
  # 階級ごとの度数を求める
  dplyr::count(class) %>%
  # 累積度数、相対度数、累積相対度数を求める
  dplyr::mutate(cumsum_n = cumsum(n),
                prop = prop.table(n), cumsum_prop = cumsum(prop)) %>%
  # 階級値を求める
  dplyr::mutate(class_value = as.character(class)) %>%
  tidyr::separate(class_value, into = c("l", "h"), sep = ",") %>%
  dplyr::mutate(l = as.integer(stringr::str_remove(l, "[[:punct:]]")) + 1L,
                h = as.integer(stringr::str_remove(h, "[[:punct:]]")),
                mids = (l + h) / 2L) %>%
  # 変数名を日本語にする
  dplyr::select(`階級` = class, `階級値` = mids,
                `度数` = n, `累積度数` = cumsum_n,
                `相対度数` = prop, `累積相対度数` = cumsum_prop)
```

```
## # A tibble: 7 x 6
##   階級      階級値  度数 累積度数 相対度数 累積相対度数
##   <fct>      <dbl> <int>    <int>    <dbl>        <dbl>
## 1 (35,40]     38     3        3   0.0375       0.0375
## 2 (40,45]     43    11       14   0.138        0.175
## 3 (45,50]     48    33       47   0.412        0.588
## 4 (50,55]     53    19       66   0.238        0.825
## 5 (55,60]     58     7       73   0.0875       0.912
## 6 (60,65]     63     5       78   0.0625       0.975
```

```
## 7 (65,70]     68     2       80   0.025        1
```

ヒストグラムを描きます。

```
with(df, hist(weight))
```

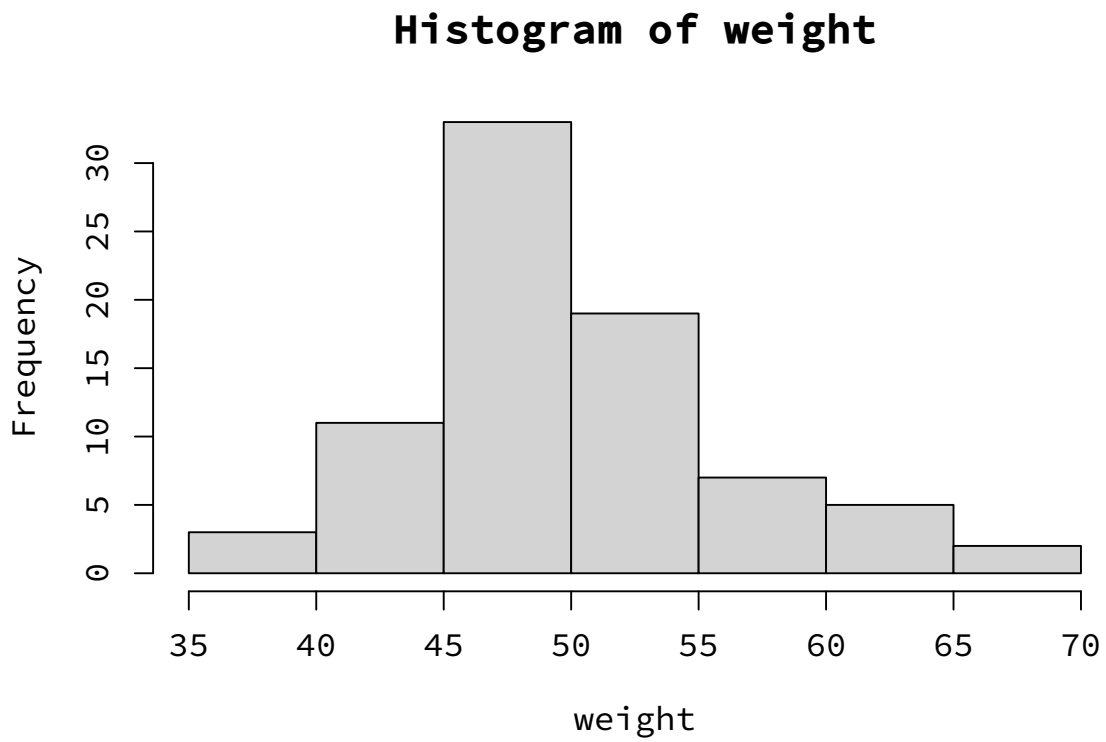


Figure 1.2: 解答例

以上

ヒストグラムを描くポイント

ヒストグラムを描く（度数分布表を作成する）際のポイントは下記の点です。

1. 階級の決め方
2. 階級の境界の扱い

階級の決め方

階級を決める方法は特に定められていません。データが取る幅を見て切のよい値にすることが多いようです。ただし、階級のとり方によりヒストグラムの形状が変わる点には注意が必要です。例えば、身長データに対する階級をテキストと同様に $5cm$ 幅とした場合は下図のような形状になります。

```
with(x, hist(height, breaks = seq(from = 140, to = 170, by = 5)))
```

Figure 1.3: 階級幅 5cm の場合

階級を倍の $10cm$ 幅とすると下図のようになります。

```
with(x, hist(height, breaks = seq(from = 140, to = 170, by = 10)))
```

Figure 1.4: 階級幅 10cm の場合

逆に階級を半分の 2.5cm 幅とすると下図のようになります。

```
with(x, hist(height, breaks = seq(from = 140, to = 170, by = 2.5)))
```

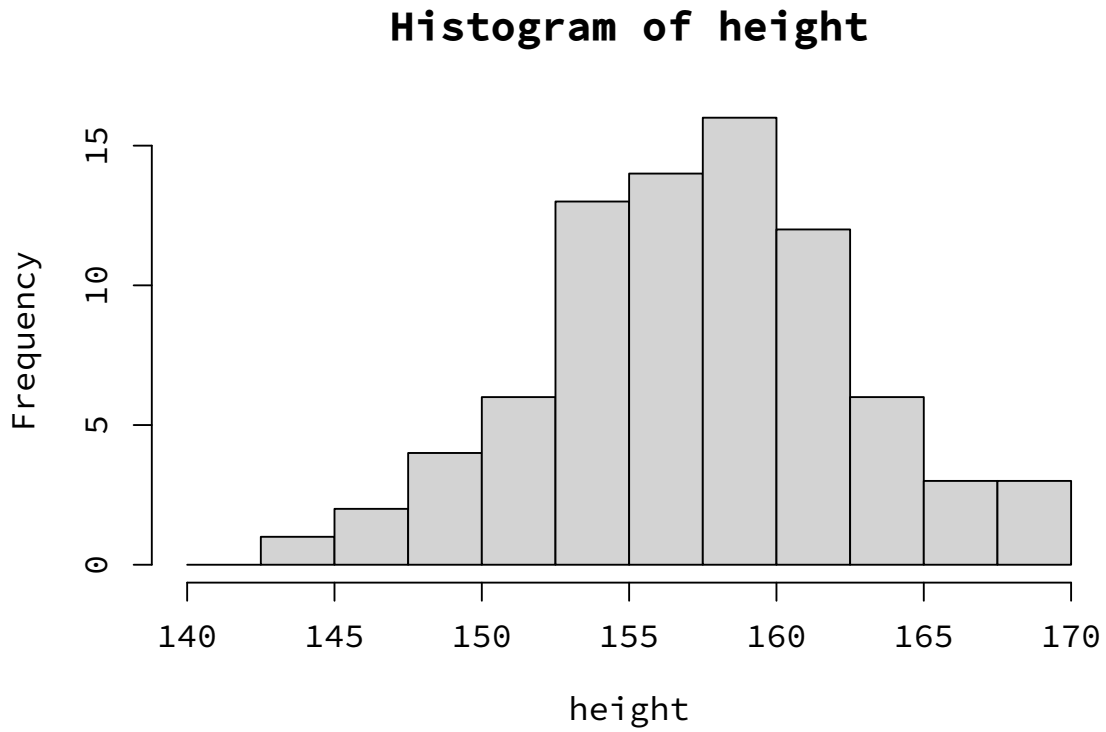


Figure 1.5: 階級幅 2.5cm の場合

更に細かくして 1cm 幅にすると下図のように歯抜けがある形状になります。

```
with(x, hist(height, breaks = seq(from = 140, to = 170, by = 1.0)))
```

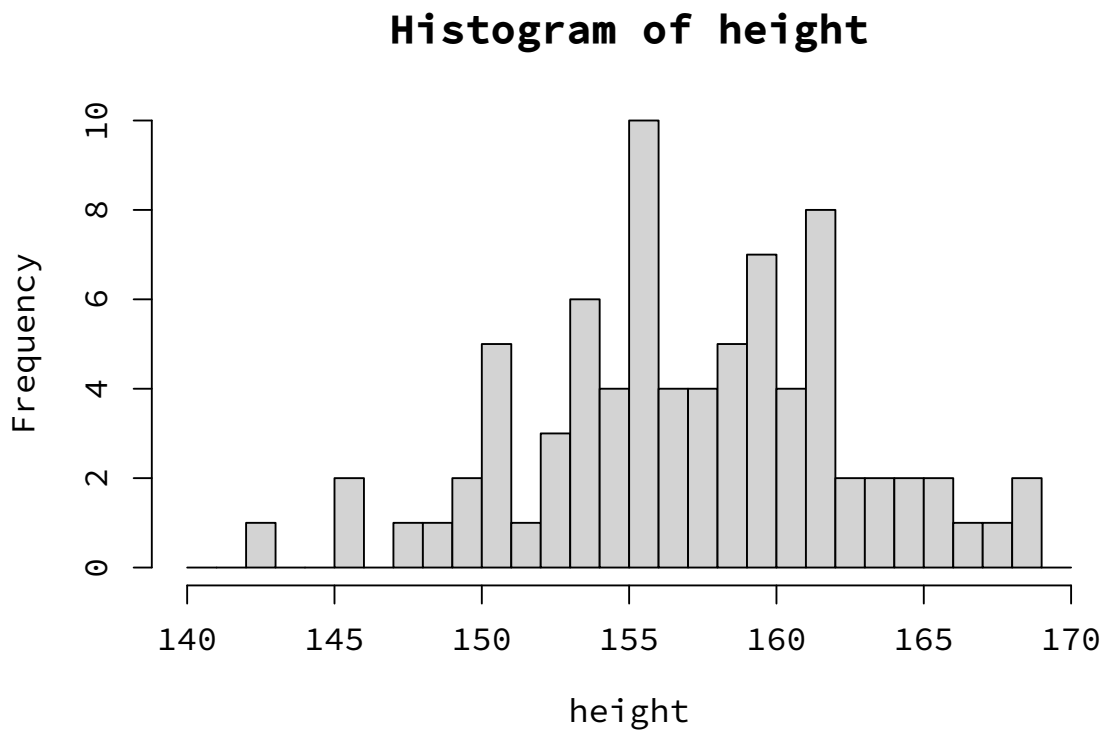


Figure 1.6: 階級幅 1cm の場合

このように階級の決め方次第でヒストグラムの形状が変わってくることが分かります。ヒストグラムはデータの分布をみるために使うグラフですので、過大な階級幅や過小な階級幅で描くことは好ましくありませんので、適切な値を選ぶようにしてください。

計算で階級を決める

適切な階級をどのように決めれば良いか迷う場合には、下表のような計算方法が提案されていますのでこれらを試してみてください。

階級の求め方	階級数（k）・階級幅（h）	備考
平方根選択（Square-root choice）	$k = \sqrt{n}$	
スタージェスの公式（Sturges's formula）	$k = \lceil \log_2 n + 1 \rceil$	$n \geq 30$ が前提
スコットの選択（Scott's choice）	$h = \frac{3.5\sigma}{n^{1/3}}$	
フリードマン・ダイアコニスの選択（F-D's choice）	$h = 2\frac{IQR(x)}{n^{1/3}}$	

階級数（k）から階級幅（h）を求める場合は下式を使います。

$$h = \lceil \frac{max(x) - min(x)}{k} \rceil$$

k : 階級の数
h : 階級の幅
n : データの個数
$\lceil x \rceil$: 天井関数（実数 x に対して x 以上の最小の整数を返す関数）
σ : 標準偏差
IQR : 四分位範囲

数式出典：Wikipedia[2]

スタージェスの公式　スタージェスの公式は、比較的、よく使われる計算方法です。ただし、データの数が 30 個未満の場合には適切ではありませんし、データの数が多くなるとヒストグラムを平滑化し過ぎる傾向[3]があると言われています。そのような傾向が見られた場合には、他の計算方法や任意の階級も試してみてください。

```
with(x, hist(height, breaks = "Sturges"))
```

[2] https://ja.wikipedia.org/wiki/%E3%83%92%E3%82%B9%E3%83%88%E3%82%B0%E3%83%A9%E3%83%A0#%E9%9A%8E%E7%B4%9A%E3%81%AE%E5%80%8B%E6%95%B0%E3%81%A8%E5%B9%85

[3] https://k-metrics.netlify.app/post/2018-09/histogram/#/%E4%BB%BB%E6%84%8F%E3%81%AE%E9%9A%8E%E7%B4%9A%E3%82%92%E6%8C%87%E5%AE%9A%E3%81%99%E3%82%8B

スコットの選択　スコットの選択は、データによってはスタージェスの公式と比べて階級数が少なくなる傾向があります。

```
with(x, hist(height, breaks = "Scott"))
```

フリードマン・ダイアコニスの選択　フリードマン・ダイアコニスの法則とも呼ばれます。

```
with(x, hist(height, breaks = "FD"))
```

Figure 1.7: 階級をフリードマン・ダイアコニスの選択で求めた場合

階級の境界の扱い

階級は下表のように「下端値〜上端値」の形式で表示されることが多いですが、140は階級に含まれるのか？145 はどちらの階級に含まれるのか？を決めておかないと意図しない度数になってしまう場合があります。

階級	度数
140〜145	1
145〜150	6
…	…
160〜165	16
165〜170	6

例えば、階級を「下端値 $< x \leq$ 上端値」と定義した場合は、下図のようになります。

```
with(x, hist(height, right = TRUE))
```

Figure 1.8: 上端を含める場合

階級を「下端値 $\leq x < l$上端値」と定定義した場合は、分布形状が異なることが分かります。

```
with(x, hist(height, right = FALSE))
```

Figure 1.9: 下端を含める場合

このように階級によりヒストグラムの形状が変わってくる点には注意が必要です。なお、下端、上端の含め方を「左閉じ、右閉じ」と表現[4]することもあります。

等間隔ではない階級

度数分布表の階級は必ずしも等間隔である必要はありません。例えば、下図の世帯貯蓄のように度数の小さい階級をまとめたヒストグラムをしばしば見かけます。等間隔でないは階級は（最小値と最大値で桁が数桁異なるなど）幅が広いデータにおいて、階級を細かくしたい場合や等幅の階級でグラフ化するとデータを読み取りにくい場合などに使われます。

```
knitr::include_graphics("https://www.stat.go.jp/naruhodo/img/picture/4_8_10.png")
```

このように階級幅が等間隔でない場合には、柱（棒）の面積が度数に対応するように高さを調整する必要があります。

ヒストグラムは棒グラフとは異なり階級ごとの柱（棒）の面積が意味を持っています。高さは一般的に度数として表示されますが、密度として表示することもあります。高さ（密度）と幅（階級幅）を乗じたものが相対度数になります。相対度数は名前の通り度数に比例していますので、柱（棒）の面積が等しければ同じ度数ということを表します。

例えば、テキストにある身長のデータの $140cm$ から $150cm$ をひとつの階級にまとめた場合、度数分布表は下表のようになります。

```
x %>%
  dplyr::mutate(class = cut(height,
                            breaks = c(140, 150, 155, 160, 165, 170))) %>%
  dplyr::count(class) %>%
  dplyr::mutate(prop = prop.table(n)) %>%
  dplyr::rename(`階級` = class, `度数` = n, `相対度数` = prop)
```

```
## # A tibble: 5 x 3
##   階級         度数 相対度数
##   <fct>      <int>    <dbl>
## 1 (140,150]      7   0.0875
## 2 (150,155]     19   0.238
## 3 (155,160]     30   0.375
## 4 (160,165]     18   0.225
## 5 (165,170]      6   0.075
```

[4] https://k-metrics.netlify.app/post/2018-09/histogram/#/%E5%B7%A6%E9%96%89%E3%81%98%E3%81%A8%E5%8F%B3%E9%96%89%E3%81%98

これをヒストグラムとして描いた場合は、下図のようにならなければなりません。この図では縦軸は度数でなく密度になっています。

```
with(x, hist(height, breaks = c(140, 150, 155, 160, 165, 170)))
```

Figure 1.10: 正しいヒストグラム

密度（高さ）は左から
0.00875, 0.0475, 0.075, 0.045, 0.015
となっていますので、これに個々の階級幅（幅）
10, 5, 5, 5, 5
を乗じると個々の柱（棒）の面積は
0.0875, 0.2375, 0.375, 0.225, 0.075
となり、相対度数と等しいことがわかります。

度数を表示させるために下図のようなヒストグラムを描くことは好ましくありません。階級幅が等幅でない場合は度数分布表と共に密度のヒストグラムを示した方がよいでしょう。

```
with(x, hist(height, breaks = c(140, 150, 155, 160, 165, 170), freq = TRUE))
```

Figure 1.11: 好ましくないヒストグラム（単なる棒グラフ）

Chapter 2

平均値とはやじろべえの視点である

本講では以下のデータフレームを利用しています。

身長データ

```
x
```

```
## # A tibble: 80 x 1
##    height
##     <dbl>
##  1    151
##  2    154
##  3    160
##  4    160
##  5    163
##  6    156
##  7    158
##  8    156
##  9    154
## 10    160
## # ... with 70 more rows
```

度数分布表

```
df
```

```
## # A tibble: 6 x 9
##   class         n     l     h  mids range cumsum_n   prop cumsum_prop
##   <fct>     <int> <dbl> <dbl> <dbl> <dbl>    <int>  <dbl>       <dbl>
```

```
## 1 (140,145]     1   140   145   143     5        1 0.0125      0.0125
## 2 (145,150]     6   145   150   148     5        7 0.075       0.0875
## 3 (150,155]    19   150   155   153     5       26 0.238       0.325
## 4 (155,160]    30   155   160   158     5       56 0.375       0.7
## 5 (160,165]    18   160   165   163     5       74 0.225       0.925
## 6 (165,170]     6   165   170   168     5       80 0.075       1
```

2.1 統計量は、データを要約する数値

本稿で扱っている統計量（要約統計量）は平均のみです。

2.2 平均値とは

本稿の平均値とは算術平均を意味しています。身長データ（x）の平均値は 157.575 です。

2.3 度数分布表での平均値

```
df %>%
  dplyr::select(`階級` = class, `度数` = n, `階級値` = mids, `階級幅` = range,
                `累積度数` = cumsum_n, `相対度数` = prop, `累積相対度数` = cumsum_prop)
```

```
## # A tibble: 6 x 7
##   階級        度数  階級値  階級幅  累積度数  相対度数  累積相対度数
##   <fct>      <int>  <dbl>  <dbl>     <int>     <dbl>         <dbl>
## 1 (140,145]     1    143      5         1   0.0125        0.0125
## 2 (145,150]     6    148      5         7   0.075         0.0875
## 3 (150,155]    19    153      5        26   0.238         0.325
## 4 (155,160]    30    158      5        56   0.375         0.7
## 5 (160,165]    18    163      5        74   0.225         0.925
## 6 (165,170]     6    168      5        80   0.075         1
```

（算術）平均は全ての観測値の総和を総データ数で除したものですので、観測値を各階級の代表値である階級値で近似すると下式が成り立つことが分かります。

$$\text{平均} = \frac{\sum \text{観測値}_i}{\sum_i} = \frac{\sum \text{観測値}_i}{\sum_i} \fallingdotseq \frac{\sum (\text{階級値}_i \times \text{度数}_i)}{\sum_i} = \sum (\text{階級値}_i \times \text{相対度数}_i)$$

なぜなら

$$\frac{\text{度数}_i}{\sum \text{度数}_i} = \text{相対度数}_i$$

$$i = 1, 2, ..., n$$

```
df %>%
  dplyr::select(A = mids, B = prop) %>%
  dplyr::mutate(`A x B` = A * B) %>%
  dplyr::rename(`A 階級値` = A, `B 相対度数` = B, `A x B` = `A x B`)
```

```
## # A tibble: 6 x 3
##   `A 階級値` `B 相対度数` `A x B`
##        <dbl>         <dbl>   <dbl>
## 1        143        0.0125    1.79
## 2        148        0.075    11.1
## 3        153        0.238    36.3
## 4        158        0.375    59.2
## 5        163        0.225    36.7
## 6        168        0.075    12.6
```

```
df %>%
  dplyr::select(A = mids, B = prop) %>%
  dplyr::mutate(`A x B` = A * B) %>%
  dplyr::summarise(`平均値` = sum(`A x B`))
```

```
## # A tibble: 1 x 1
##   平均値
##    <dbl>
## 1   158.
```

実データで求めた平均値は 157.575 ですので、階級値から求めた平均値との差は-0.175 となり、度数分布表を作ることは平均値に大きな影響を与えないことが分かります。

2.4 平均値のヒストグラムの中での役割

階級値から求めた平均値をヒストグラム上にプロットすると下図のようになります。

```
with(x, hist(height))
abline(v = 157.75, lty = 2)
```

2.5 平均値をどう捉えるべきか

平均値の捉え方は様々考えられますが、テキストでは下記のようにまとめています。

- 全データ（値）を代表する値（点）
- データは平均値の周辺に分布する
- 数多く現れるデータは平均値への影響が大きい
- データの分布が対象の場合、平均値は対象軸

2.6 練習問題

2.6.1 解答例

```
data.frame(mids = c(30, 50, 70, 90, 110, 130), n = c(5, 10, 15, 40, 20, 10)) %>%
  dplyr::mutate(prop = prop.table(n), cm = mids * prop) %>%
  dplyr::rename(`階級値` = mids, `度数` = n, `相対度数` = prop, `階級値×相対度数` = cm)
```

```
##   階級値 度数 相対度数 階級値×相対度数
## 1     30    5     0.05             1.5
## 2     50   10     0.10             5.0
## 3     70   15     0.15            10.5
## 4     90   40     0.40            36.0
## 5    110   20     0.20            22.0
## 6    130   10     0.10            13.0
```

```
data.frame(mids = c(30, 50, 70, 90, 110, 130), n = c(5, 10, 15, 40, 20, 10)) %>%
  dplyr::mutate(prop = prop.table(n), cm = mids * prop) %>%
  dplyr::summarise(`平均値` = sum(cm))
```

```
##   平均値
## 1     88
```

【コラム】平均値のとり方は、１つではない

算術平均（Arithmetic Mean）

$$\frac{\sum_{i=1}^{n} x_i}{n}$$

```
sum(c(10, 90)) / length(c(10, 90))
```

```
## [1] 50
```

```
mean(c(10, 90))
```

```
## [1] 50
```

幾何平均（Geometric Mean）または相乗平均

$$\sqrt{\prod_{i=1}^{n} x_i}$$

```
psych::geometric.mean(c(10, 90))
```

```
## [1] 30
```

二乗平均（Root Mean Square）

$$\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}$$

```
sqrt(mean(c(10, 90) ^ 2))
```

```
## [1] 64.03124
```

調和平均（Harmonic Mean）

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

```
psych::harmonic.mean(c(10, 90))
```

```
## [1] 18
```

トリム平均

トリム平均は体操競技の評点などで使われる平均値の計算方法で、値を小さい方から順に並べ、上位側と下位側から一定の割合で値を除き算術平均を求めます。外れ値や異常値などを影響を排除するために使われることが多いです。

```
mean(c(0, 1:8, 30))
```

```
## [1] 6.6
```

```
mean(c(0, 1:8, 30), trim = 0.25)  # データの両側 2.5% を対象外とする
```

```
## [1] 4.5
```

追加問題

身長のデータを用いて階級幅が変わると度数分布表から求める平均値がどのように変化するか確認してみましょう。

階級幅が倍になった場合

```
x %>%
  dplyr::mutate(class = cut(height,
                            breaks = c(140, 150, 160, 170),
                            include.lowest = FALSE, right = TRUE)) %>%
  dplyr::count(class) %>%
  dplyr::mutate(class_value = as.character(class)) %>%
  tidyr::separate(class_value, into = c("l", "h"), sep = ",") %>%
  dplyr::mutate(l = as.numeric(stringr::str_remove(l, "[[:punct:]]")),
                h = as.numeric(stringr::str_remove(h, "[[:punct:]]"))) %>%
  dplyr::mutate(mids = ((l + 1) + h) / 2) %>%
  dplyr::mutate(range = h - l, cumsum_n = cumsum(n),
                prop = prop.table(n), cumsum_prop = cumsum(prop)) %>%
  dplyr::select(A = mids, B = prop) %>%
  dplyr::mutate(`A x B` = A * B) %>%
  dplyr::summarise(`階級値による平均値` = sum(`A x B`)) %>%
  dplyr::mutate(`算術平均` = mean(x$height), `差` = `算術平均` - `階級値による平均値`)
```

```
## # A tibble: 1 x 3
##   階級値による平均値 算術平均      差
##                <dbl>    <dbl>   <dbl>
## 1               158.     158. -0.0500
```

階級幅が半分になった場合

```
x %>%
  dplyr::mutate(class = cut(height,
                            breaks = seq(from = 140, to = 170, by = 2.5),
                            include.lowest = FALSE, right = TRUE)) %>%
  dplyr::count(class) %>%
  dplyr::mutate(class_value = as.character(class)) %>%
  tidyr::separate(class_value, into = c("l", "h"), sep = ",") %>%
  dplyr::mutate(l = as.numeric(stringr::str_remove(l, "[[:punct:]]")),
                h = as.numeric(stringr::str_remove(h, "[[:punct:]]"))) %>%
  dplyr::mutate(mids = ((l + 1) + h) / 2) %>%
  dplyr::mutate(range = h - l, cumsum_n = cumsum(n),
                prop = prop.table(n), cumsum_prop = cumsum(prop)) %>%
  dplyr::select(A = mids, B = prop) %>%
  dplyr::mutate(`A x B` = A * B) %>%
  dplyr::summarise(`階級値による平均値` = sum(`A x B`)) %>%
  dplyr::mutate(`算術平均` = mean(x$height), `差` = `算術平均` - `階級値による平均値`)
```

```
## # A tibble: 1 x 3
##   階級値による平均値 算術平均      差
##                <dbl>    <dbl>  <dbl>
## 1               158.     158. -0.281
```

Chapter 3

データの散らばり具合を見積もる統計量

3.1　データの散らばりやバラツキを知りたい

第 2 講で学んだ平均値（算術平均）は、データを代表する統計量で、ある一点（支点・重心）を示しているに過ぎません。データが平均値の周辺にどのように分布しているかはヒストグラムを使うことで視覚的には確認できますが、統計量として数値的に扱えた方がなにかと便利なはずです。

3.2　バスの到着時刻の例で分散を理解する

蛇足ですが実際の路線バスでは予定時刻より早く到着しないように調整運転がなされていますので、時刻より大幅に早着することは稀です。

```
x <- scan(file = "../data/P36_図表 3-1.csv", sep = ",")
x
```

```
## [1] 32 27 29 34 33
```

平均値を求める mean() 関数がありますが、ここでは平均値の計算式通りの計算を行っています。

```
`平均値` <- sum(x) / length(x)
`平均値`
```

```
## [1] 31
```

```
`偏差` = x - `平均値`
`偏差`
```

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

偏差の総和は必ずゼロ（0）になります。

```
sum(`偏差`)
```

```
## [1] 0
```

偏差の二乗和をデータの個数で割ったものは、（標本）分散と呼ばれます。不偏分散と呼ばれる分散は計算式が異なります。R で分散を計算する `var()` 関数は不偏分散を求める関数です。

```
sum(`偏差` ^ 2) / length(x)
```

```
## [1] 6.8
```

偏差の二乗平均値（$= \sqrt{\text{分散}}$）は標準偏差と呼ばれます。R で標準偏差を計算する `sd()` 関数は不偏分散のルートを取ったものです。

```
sqrt(sum(`偏差` ^ 2) / length(x))
```

```
## [1] 2.607681
```

3.3 標準偏差の意味

```
x <- read.csv(file = "../data/P39_図表3-4.csv")
x
```

```
##   X Y
## 1 4 1
## 2 4 2
## 3 5 6
## 4 6 7
## 5 6 9
```

x

```
mean <- x %>%
  dplyr::summarise(`X 平均値` = mean(X), `Y 平均値` = mean(Y))
mean
```

```
##   X平均値 Y平均値
## 1       5       5
```

```
x %>%
  dplyr::mutate(`X 偏差` = X - mean$`X 平均値`, `Y 偏差` = Y - mean$`Y 平均値`)
```

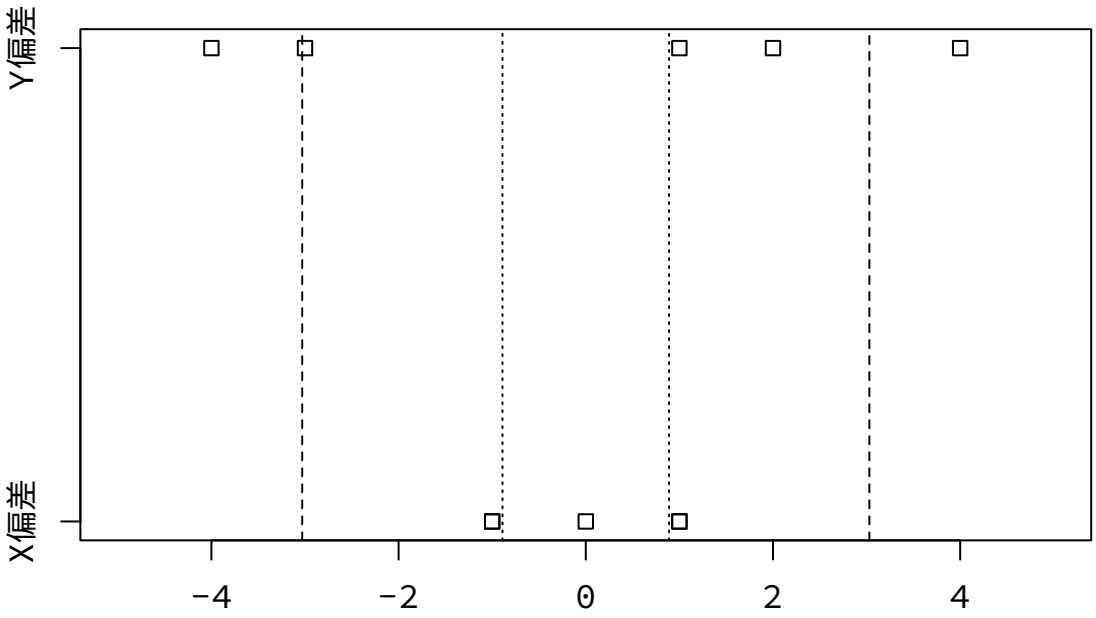
```
##   X Y X偏差 Y偏差
## 1 4 1    -1    -4
## 2 4 2    -1    -3
## 3 5 6     0     1
## 4 6 7     1     2
## 5 6 9     1     4
```

```
x %>%
  dplyr::mutate(`X 偏差` = X - mean$`X 平均値`, `Y 偏差` = Y - mean$`Y 平均値`) %>%
  dplyr::summarise(`X 標準偏差` = sqrt(sum(`X 偏差` ^ 2) / length(X)),
                   `Y 標準偏差` = sqrt(sum(`Y 偏差` ^ 2) / length(Y)))
```

```
##   X標準偏差 Y標準偏差
## 1 0.8944272   3.03315
```

偏差と標準偏差を可視化すると下図のようになり、X より Y の方が広範囲に分布していることが分かります。

```
x %>%
  dplyr::mutate(`X 偏差` = X - mean$`X 平均値`, `Y 偏差` = Y - mean$`Y 平均値`) %>%
  dplyr::select(-X, -Y) %>%
  stripchart(xlim = c(-5, 5))
abline(v = c(-0.89, 0.89), lty = 3)  # X 標準偏差／ドット（点線）
abline(v = c(-3.03, 3.03), lty = 2)  # Y 標準偏差／ダッシュ（鎖線）
```



3.4 度数分布表から標準偏差を求める

```
x <- read.csv(file = "../data/P40_図表3-6.csv")
x
```

```
##   A階級値 B相対度数
## 1       1       0.3
## 2       2       0.5
## 3       3       0.1
## 4       4       0.1
```

第 2 講の「2–3 度数分布表での平均値」で学んだように度数分布表から平均値（の近似値）は

$$\text{平均値} \fallingdotseq \sum(\text{階級値} \times \text{相対度数})$$

で求めることができます。

```
x %>%
  dplyr::mutate(`AxB` = `A階級値` * `B相対度数`)
```

```
##   A階級値 B相対度数 AxB
## 1       1       0.3 0.3
## 2       2       0.5 1.0
## 3       3       0.1 0.3
## 4       4       0.1 0.4
```

```
`平均値` <- x %>%
  dplyr::mutate(`AxB` = `A 階級値` * `B 相対度数`) %>%
  dplyr::summarize(`平均値` = sum(`AxB`)) %>%
  dplyr::pull(`平均値`)
`平均値`
```

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

平均値を求められたので

$$\text{階級の偏差（C 階級値-平均値）} = \text{階級値} - \text{平均値}$$

とすると度数分布表の計算値と同じ考え方を適用し

$$\text{階級の偏差の二乗平均} = \sum(\text{階級の偏差}^2 \times \text{相対度数}) = \text{分散}$$

となります。

```
x %>%
  dplyr::mutate(`C` = `A 階級値` - `平均値`) %>%
  dplyr::mutate(`C^2` = `C` ^ 2) %>%
  dplyr::mutate(`C^2xB` = `C^2` * `B 相対度数`) %>%
  dplyr::select(`A 階級値`, `C 階級値-平均値` = `C`, `C^2`, `B 相対度数`, `C^2xB`)
```

```
##   A階級値 C階級値-平均値 C^2 B相対度数 C^2xB
## 1       1           -1   1       0.3   0.3
## 2       2            0   0       0.5   0.0
## 3       3            1   1       0.1   0.1
## 4       4            2   4       0.1   0.4
```

```
x %>%
  dplyr::mutate(`C` = `A 階級値` - `平均値`) %>%
  dplyr::mutate(`C^2` = `C` ^ 2) %>%
  dplyr::mutate(`C^2xB` = `C^2` * `B 相対度数`) %>%
  dplyr::select(`A 階級値`, `C 階級値-平均値` = `C`, `C^2`, `B 相対度数`, `C^2xB`) %>%
  dplyr::summarise(`分散` = sum(`C^2xB`)) %>%
  dplyr::mutate(`標準偏差` = sqrt(`分散`))
```

```
##   分散  標準偏差
## 1  0.8 0.8944272
```

練習問題

次の架空のデータの標準偏差を次のステップで計算してみよ。

```
`データ` <- scan(file = "../data/P42_練習問題_v.csv", sep = ",")
`データ`
```

```
##  [1] 6 4 6 6 6 3 7 2 2 8
```

解答例

```
`平均値` <- sum(`データ`) / length(`データ`)
`平均値`
```

```
## [1] 5
```

```
`偏差` <- `データ` - `平均値`
`偏差`
```

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

```
`二乗偏差` <- `偏差` ^ 2
`二乗偏差`
```

```
##  [1] 1 1 1 1 1 4 4 9 9 9
```

```
`分散` <- sum(`二乗偏差`) / length(`二乗偏差`)
`分散`
```

```
## [1] 4
```

```
`標準偏差` <- sqrt(`分散`)
`標準偏差`
```

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

追加問題

P16 にある身長データの度数分布表を作り、分散と標準偏差を求めてみましょう。

```
## # A tibble: 8 x 2
##   A階級値 B相対度数
##     <dbl>     <dbl>
## 1    145     0.0375
## 2    148.    0.025
## 3    152     0.1
## 4    155     0.288
## 5    158     0.162
## 6    161     0.238
## 7    164.    0.075
## 8    168     0.075
```

解答例

```
x %>%
  dplyr::mutate(`AxB` = `A階級値` * `B相対度数`)
```

```
## # A tibble: 8 x 3
##   A階級値 B相対度数    AxB
##     <dbl>     <dbl> <dbl>
## 1    145     0.0375  5.44
## 2    148.    0.025   3.71
## 3    152     0.1    15.2
## 4    155     0.288  44.6
## 5    158     0.162  25.7
## 6    161     0.238  38.2
## 7    164.    0.075  12.3
## 8    168     0.075  12.6
```

```
`平均値` <- x %>%
  dplyr::mutate(`AxB` = `A階級値` * `B相対度数`) %>%
  dplyr::summarise(`平均値` = sum(`AxB`)) %>%
  dplyr::pull(`平均値`)
`平均値`
```

```
## [1] 157.7625
```

```
x %>%
  dplyr::mutate(`C` = `A 階級値` - `平均値`) %>%
  dplyr::mutate(`C^2` = `C` ^ 2) %>%
  dplyr::mutate(`C^2xB` = `C^2` * `B 相対度数`)
```

```
## # A tibble: 8 x 5
##   A階級値 B相対度数       C    `C^2` `C^2xB`
##     <dbl>     <dbl>   <dbl>    <dbl>   <dbl>
## 1    145     0.0375 -12.8   163.     6.11
## 2    148.    0.025   -9.26   85.8    2.14
## 3    152     0.1     -5.76   33.2    3.32
## 4    155     0.288   -2.76    7.63   2.19
## 5    158     0.162    0.238   0.0564 0.00917
## 6    161     0.238    3.24   10.5    2.49
## 7    164.    0.075    6.74   45.4    3.40
## 8    168     0.075   10.2   105.     7.86
```

```
freq_summary <- x %>%
  dplyr::mutate(`C` = `A 階級値` - `平均値`) %>%
  dplyr::mutate(`C^2` = `C` ^ 2) %>%
  dplyr::mutate(`C^2xB` = `C^2` * `B 相対度数`) %>%
  dplyr::summarise(`平均` = sum(`A 階級値` * `B 相対度数`),
                   `分散` = sum(`C^2xB`)) %>%
  dplyr::mutate(`標準偏差` = sqrt(`分散`))
freq_summary
```

```
## # A tibble: 1 x 3
##     平均   分散 標準偏差
##   <dbl> <dbl>    <dbl>
## 1  158.  27.5     5.25
```

生データで計算した場合

```
"../data/P16_図表 1-1 .csv" %>%
  readr::read_csv(col_names = FALSE, show_col_types = FALSE) %>%
  tidyr::pivot_longer(cols = dplyr::starts_with("X"),
                      names_to = "name", values_to = "value") %>%
  dplyr::arrange(name) %>%
  dplyr::select(height = value) %>%
  dplyr::mutate(`偏差` = height - mean(height)) %>%
  dplyr::mutate(`偏差^2` = `偏差` ^ 2)
```

```
## # A tibble: 80 x 3
##    height   偏差 `偏差^2`
##     <dbl>  <dbl>    <dbl>
##  1    151 -6.57    43.2
##  2    154 -3.57    12.8
##  3    160  2.43     5.88
##  4    160  2.43     5.88
##  5    163  5.43    29.4
##  6    156 -1.57     2.48
##  7    158  0.425    0.181
##  8    156 -1.57     2.48
##  9    154 -3.57    12.8
## 10    160  2.43     5.88
## # ... with 70 more rows
```

```
raw_summary <- "../data/P16_図表1-1 .csv" %>%
  readr::read_csv(col_names = FALSE, show_col_types = FALSE) %>%
  tidyr::pivot_longer(cols = dplyr::starts_with("X"),
                      names_to = "name", values_to = "value") %>%
  dplyr::arrange(name) %>%
  dplyr::select(height = value) %>%
  dplyr::mutate(`偏差` = height - mean(height)) %>%
  dplyr::mutate(`偏差^2` = `偏差` ^ 2) %>%
  dplyr::summarise(`平均` = sum(height) / length(height),
                   `分散` = sum(`偏差^2`) / length(height),
                   `標準偏差` = sqrt(`分散`))
raw_summary
```

```
## # A tibble: 1 x 3
##    平均   分散 標準偏差
##   <dbl> <dbl>    <dbl>
## 1  158.  28.8     5.37
```

要約統計量を比べてみると以下のようになります。

```
freq_summary # 度数分布表からの要約統計量
```

```
## # A tibble: 1 x 3
##    平均   分散 標準偏差
##   <dbl> <dbl>    <dbl>
## 1  158.  27.5     5.25
```

```
raw_summary  # 生データからの要約統計量
```

```
## # A tibble: 1 x 3
##     平均   分散  標準偏差
##   <dbl> <dbl>     <dbl>
## 1  158.  28.8      5.37
```

```
freq_summary - raw_summary # 両者の差
```

```
##      平均       分散     標準偏差
## 1 0.1875 -1.313281 -0.1236879
```

Chapter 4

そのデータは「月並み」か「特殊」か？標準偏差で評価する

4.1　標準偏差は波の「激しさ」

　標準偏差は平均値からどの程度データがばらついているかを表しています。テキストではこのばらつきを「波打ちの激しさ」と表現しています。

4.2　標準偏差がわかるとデータの特殊性を評価できる

　標準偏差がわかるということは、以下がわかるということになります。

- 一つのデータ（セット）の中にある任意の値の持つ意味
- 複数のデータ（セット）の違い

4.3　複数のデータセットの比較

4.4　加工されたデータの平均値と標準偏差

　以下のデータに一定の加工を加えた際に平均値や標準偏差がどのように変化するかを見てみましょう。

```
x <- read.csv(file = "../data/P49_図表 4-5.csv")
x
```

```
##   X
## 1 1
## 2 3
## 3 4
## 4 5
## 5 7
```

データに一定数を加えた場合

それぞれに 4 を加える。

```
x %>%
  dplyr::mutate(Y = X + 4)
```

```
##   X  Y
## 1 1  5
## 2 3  7
## 3 4  8
## 4 5  9
## 5 7 11
```

```
mean <- x %>%
  dplyr::mutate(Y = X + 4) %>%
  dplyr::summarise(X = mean(X), Y = mean(Y))
mean
```

```
##   X Y
## 1 4 8
```

偏差

```
dev <- x %>%
  dplyr::mutate(Y = X + 4) %>%
  dplyr::mutate(X = X - mean$X, Y = Y - mean$Y)
dev
```

```
##    X  Y
## 1 -3 -3
## 2 -1 -1
## 3  0  0
## 4  1  1
## 5  3  3
```

分散

```
var <- dev %>%
  dplyr::summarize(X = sum(X ^ 2) / length(X), Y = sum(Y ^ 2) / length(Y))
var
```

```
##   X Y
## 1 4 4
```

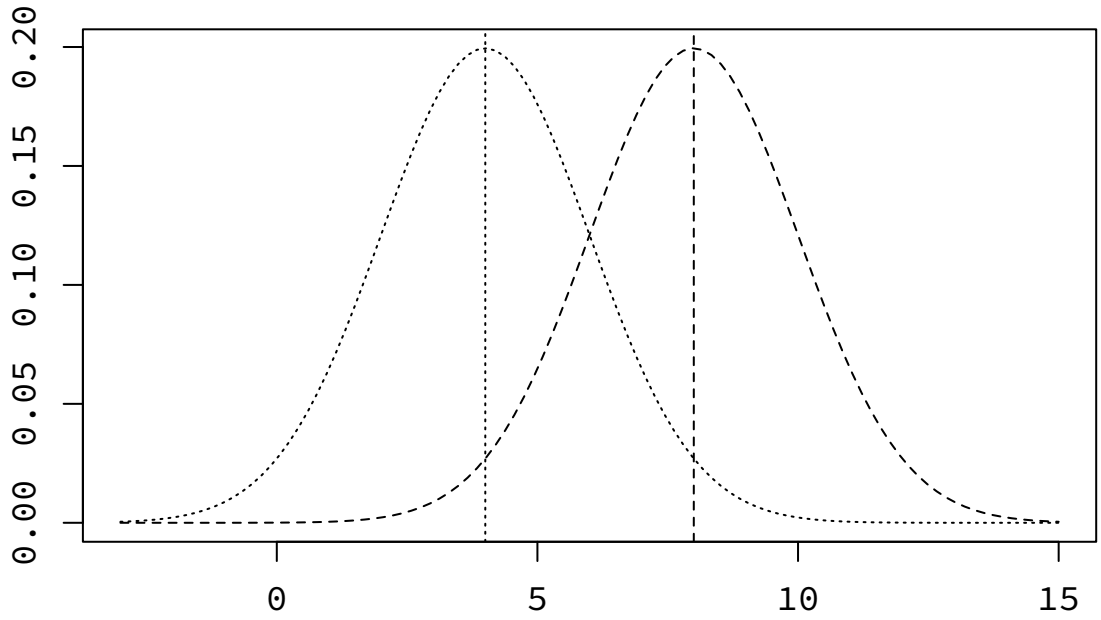
標準偏差

```
sqrt(var)
```

```
##   X Y
## 1 2 2
```

以上のようにデータに一定数を加えると平均値は加えた一定値の分だけ大きくなりますが、他の統計量は変化していないことが分かります。これを正規分布で図示すると下図のようになります。

```
curve(dnorm(x, mean = 4, sd = 2), -3, 15, lty = 3, xlab = "", ylab = "")
abline(v = 4, lty = 3)
curve(dnorm(x, mean = 8, sd = 2), -3, 15, add = TRUE, lty = 2, xlab = "", ylab = ""
abline(v = 8, lty = 2)
```



データに一定数を乗じた場合

それぞれに 2 を乗ずる。

```
x %>%
  dplyr::mutate(Y = X * 2)
```

```
##   X  Y
## 1 1  2
## 2 3  6
## 3 4  8
## 4 5 10
## 5 7 14
```

```
mean <- x %>%
  dplyr::mutate(Y = X * 2) %>%
  dplyr::summarise(X = mean(X), Y = mean(Y))
mean
```

```
##   X Y
## 1 4 8
```

偏差

```
dev <- x %>%
  dplyr::mutate(Y = X * 2) %>%
  dplyr::mutate(X = X - mean$X, Y = Y - mean$Y)
dev
```

```
##    X  Y
## 1 -3 -6
## 2 -1 -2
## 3  0  0
## 4  1  2
## 5  3  6
```

分散

```
var <- dev %>%
  dplyr::summarize(X = sum(X ^ 2) / length(X), Y = sum(Y ^ 2) / length(Y))
var
```

```
##   X  Y
## 1 4 16
```

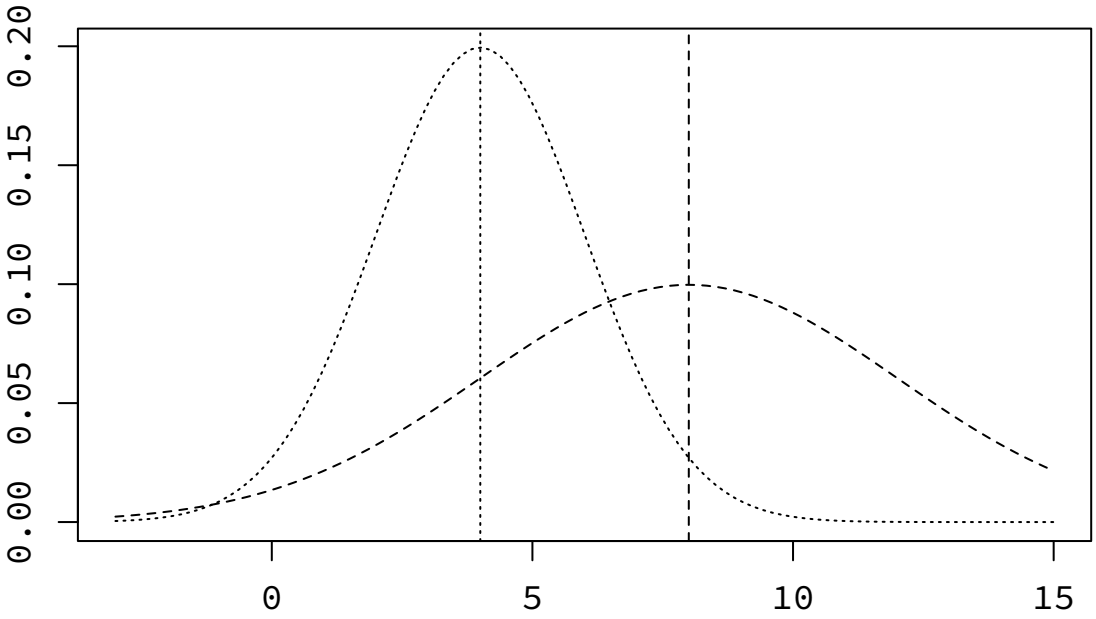
標準偏差

```
sqrt(var)
```

```
##   X Y
## 1 2 4
```

以上のようにデータに一定数を乗じると平均値は乗じた一定値の分だけ大きくなり、他の統計量も乗じた一定値の分だけ大きくなっていることが分かります。これを正規分布で図示すると下図のようになります。

```
curve(dnorm(x, mean = 4, sd = 2), -3, 15, lty = 3, xlab = "", ylab = "")
abline(v = 4, lty = 3)
curve(dnorm(x, mean = 8, sd = 4), -3, 15, add = TRUE, lty = 2,
      xlab = "", ylab = "")
abline(v = 8, lty = 2)
```



加工方法	平均値	標準偏差	備考
無加工（元データ x）	$\bar{x}$	SD	
定数の加算 $(x + k)$	$\bar{x} + k$	SD	変化するのは平均値のみ
定数の乗算 $(x \times k)$	$\bar{x} \times k$	$SD \times k$	平均値・標準偏差ともに変化

k: 任意の実数

標準偏差何個分となるようにデータを加工する効果

ここで説明されているのは「正規化」です。

```
x <- read.csv(file = "../data/P49_図表4-5.csv")
x
```

```
##   X
## 1 1
## 2 3
## 3 4
## 4 5
## 5 7
```

上記のP49のデータは、平均値4、標準偏差2ですので、(－　)÷　を計算すると下記のようになります。

```
x %>%
  dplyr::mutate(dev = (X - (sum(X) / length(X))),              # 偏差
                nX = dev / sqrt(sum(dev ^ 2) / length(dev))) # 正規化値
```

```
##   X dev   nX
## 1 1  -3 -1.5
## 2 3  -1 -0.5
## 3 4   0  0.0
## 4 5   1  0.5
## 5 7   3  1.5
```

nXの平均値と標準偏差は以下のように0と1になることが分かります。

```
x %>%
  dplyr::mutate(nX = (X - 4) / 2) %>%
  dplyr::summarise(mean = sum(nX) / length(nX),
                   sd = sqrt(sum((nX - mean) ^ 2) / length(nX)))
```

```
##   mean sd
## 1    0  1
```

練習問題

（省略）

【コラム】偏差値で嫌な思いをしたことのあるあなたに

追加問題

P16 の身長データを正規化（(データ − 平均値) ÷ 標準偏差）してみましょう。

解答例

```
x <- "../data/P16_図表 1-1 .csv" %>%
  readr::read_csv(col_names = FALSE, show_col_types = FALSE) %>%
  tidyr::pivot_longer(cols = dplyr::starts_with("X"),
                      names_to = "name", values_to = "value") %>%
  dplyr::arrange(name) %>%
  dplyr::select(height = value) %>%
  dplyr::mutate(normalized = scale(height))
x
```

```
## # A tibble: 80 x 2
##    height normalized[,1]
##     <dbl>          <dbl>
##  1    151        -1.22
##  2    154        -0.661
##  3    160         0.449
##  4    160         0.449
##  5    163         1.00
##  6    156        -0.291
##  7    158         0.0786
##  8    156        -0.291
##  9    154        -0.661
## 10    160         0.449
## # ... with 70 more rows
```

正規化前後の平均値と標準偏差を比べると下記のようになります。

```
x %>%
  dplyr::summarise(mean = sum(height) / length(height),
                   sd = sqrt(sum((height - mean)^ 2) / length(height)))
```

```
## # A tibble: 1 x 2
##    mean    sd
##   <dbl> <dbl>
## 1  158.  5.37
```

```
x %>%
  dplyr::summarise(mean = sum(normalized) / length(normalized),
                   sd = sqrt(sum((normalized - mean)^ 2) / length(normalized)))
```

```
## # A tibble: 1 x 2
##       mean    sd
##      <dbl> <dbl>
## 1 2.08e-15 0.994
```