データの整理と集計

第 *4* 講 - データフレームのより進んだ操作

村田 昇

講義概要

- データフレームの操作
- ファイルの取り扱い
- データの集計

データフレームの操作

R に用意されているデータ構造

- 下記は基本的なもので標準環境で利用できる
  - ベクトル (vector)
  - 行列 (matrix)
  - リスト (list)
  - **データフレーム (data frame)**
  - 配列 (array)

データフレームからの項目の抽出

- 添字の番号 (行と列) を指定
- 要素の名前で指定
- **除外** : マイナス記号 (-) をつけて指定
- 論理値で指定
  - `TRUE` : 要素の **選択**
  - `FALSE` : 要素の **除外**
- **欠損値** `NA` の扱いは状況依存なので注意
  - `NA` : 値が得られていないことを表すスカラー値

データ例

- **`datasets::airquality`**

  New York Air Quality Measurements
  - Description: Daily air quality measurements in New York, May to September 1973.
  - Format: A data frame with 153 observations on 6 variables.
    - [,1] Ozone numeric Ozone (ppb)
    - [,2] Solar.R numeric Solar R (lang)

    - [,3] Wind numeric Wind (mph)
    - [,4] Temp numeric Temperature (degrees F)
    - [,5] Month numeric Month (1–12)
    - [,6] Day numeric Day of month (1–31)
  - `help("airquality")` で詳細を確認
  - `datasets` は R の標準パッケージ
  - **パッケージ名:: オブジェクト** という書き方で同名のオブジェクトを区別できる

行の抽出 (素朴な方法)

- 行番号による指定

```
## 行番号のベクトルで指定して抽出
airquality[1:10,] # 1-10 行を抽出
```

```
   Ozone Solar.R Wind Temp Month Day
1     41     190  7.4   67     5   1
2     36     118  8.0   72     5   2
3     12     149 12.6   74     5   3
4     18     313 11.5   62     5   4
5     NA      NA 14.3   56     5   5
6     28      NA 14.9   66     5   6
7     23     299  8.6   65     5   7
8     19      99 13.8   59     5   8
9      8      19 20.1   61     5   9
10    NA     194  8.6   69     5  10
```

- 条件の指定

```
## 条件の指定の仕方
airquality[1:15,]$Ozone>100 # 条件に合致する行は TRUE (NA は欠損値)
airquality[1:15,]$Ozone>100 & airquality[1:15,]$Wind<=5 # 条件の AND
with(airquality[1:15,], Ozone>100 & Wind<=5) # 上と同じ (短い書き方)
with(airquality[1:60,], Ozone>100 | Wind<=5) # 条件の OR
```

```
 [1] FALSE FALSE FALSE FALSE    NA FALSE FALSE FALSE FALSE    NA FALSE FALSE
[13] FALSE FALSE FALSE
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[13] FALSE FALSE FALSE
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[13] FALSE FALSE FALSE
 [1] FALSE FALSE FALSE FALSE    NA FALSE FALSE FALSE FALSE    NA FALSE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[25]    NA    NA    NA FALSE FALSE  TRUE FALSE    NA    NA    NA    NA    NA
[37]    NA FALSE    NA FALSE FALSE    NA    NA FALSE    NA    NA FALSE FALSE
[49] FALSE FALSE FALSE    NA  TRUE  TRUE    NA    NA    NA    NA    NA    NA
```

- 条件に合致する行番号の抽出

```
## 関数 which() で TRUE の番号を抽出
which(with(airquality, Ozone>100 & Wind<=5)) # 全データから TRUE を抽出
```

```
[1]  62  99 117 121
```

- 条件に合致する行の抽出

```
## 条件を指定して行を抽出
airquality[which(with(airquality, Ozone>100 & Wind<=5)), ]
```

```
    Ozone Solar.R Wind Temp Month Day
62    135     269  4.1   84     7   1
99    122     255  4.0   89     8   7
117   168     238  3.4   81     8  25
121   118     225  2.3   94     8  29
```

列の抽出 (素朴な方法)

- 列番号による指定

```
## 列番号のベクトルで指定して抽出
airquality[which(with(airquality, Ozone>100 & Wind<=5)), c(1,5,6)]
```

```
    Ozone Month Day
62    135     7   1
99    122     8   7
117   168     8  25
121   118     8  29
```

- 列名による指定

```
## 複数の列の場合
airquality[which(with(airquality, Ozone>100 & Wind<=5)),
           c("Month","Day")]
```

```
    Month Day
62      7   1
99      8   7
117     8  25
121     8  29
```

- 列名による指定 (1 つの場合)

```
## 1つの列の場合は以下でも良い (ただしデータフレームではなくベクトルになる)
airquality[which(with(airquality, Ozone>100 & Wind<=5)),]$Month
airquality[which(with(airquality, Ozone>100 & Wind<=5)),"Month"] # 上と同じ
## データフレームとして抽出したい場合は drop=FALSE を指定する
airquality[which(with(airquality, Ozone>100 & Wind<=5)),"Month",drop=FALSE]
```

```
[1] 7 8 8 8
[1] 7 8 8 8
    Month
62      7
99      8
117     8
121     8
```

関数 subset()

- 複合的な条件を指定してデータを整理する関数

```
subset(x, subset, select, drop = FALSE)
## x:  データフレーム
## subset: 抽出する行の条件
## select: 列の選択 (未指定の場合は全ての列)
```

```
## drop: 結果が 1 行または 1 列の場合の扱い．ベクトル (TRUE)・データフレーム (FALSE)
```

関数 subset() の使い方

- 前出の例の書き換え

```
### 関数 subset() の使い方
subset(airquality,
       subset = Ozone>100 & Wind<=5,
       select = c(1,5,6))
subset(airquality,
       Ozone>100 & Wind<=5, # 順序通りなら引数の名前は省略可
       c(Month,Day)) # 名前は$の後と同じ扱いで "" は不要
```

```
    Ozone Month Day
62    135     7   1
99    122     8   7
117   168     8  25
121   118     8  29
    Month Day
62      7   1
99      8   7
117     8  25
121     8  29
```

- いろいろな記法の例 (!,is.na(),%in%)

```
## Ozone に欠測 (NA) がなく，かつ Day が 5 か 10 のデータの Wind から Day までの列を抽出
subset(airquality,
       subset = !is.na(Ozone) & Day %in% c(5,10),
       select = Wind:Day)
```

```
    Wind Temp Month Day
41  11.5   87     6  10
66   4.6   83     7   5
71   7.4   89     7  10
97   7.4   85     8   5
128  7.4   87     9   5
133  9.7   73     9  10
```

- いろいろな記法 (|,-)

```
## Ozone が 120 以上か，または Wind が 3 以下の Temp 以外の列を抽出
subset(airquality,
       subset = Ozone>120 | Wind<=3,
       select = -Temp)
```

```
    Ozone Solar.R Wind Month Day
53     NA      59  1.7     6  22
62    135     269  4.1     7   1
99    122     255  4.0     8   7
117   168     238  3.4     8  25
121   118     225  2.3     8  29
126    73     183  2.8     9   3
```

演習

練習問題

- `datasets::airquality` に対して以下の条件を満たすデータを取り出しなさい.
  - 7 月のオゾン濃度 (`Ozone`)
  - 風速 (`Wind`) が時速 10 マイル以上で，かつ気温 (`Temp`) が華氏 80 度以上の日のデータ
  - オゾン (`Ozone`) も日射量 (`Solar.R`) も欠測 (`NA`) でないデータの月 (`Month`) と日 (`Day`)

ファイルの取り扱い

データファイルの読み書き

- 実際の解析においては以下の操作が必要
  - 収集されたデータを読み込む
  - 整理したデータを保存する
- R で利用可能なデータファイル
  - CSV 形式 (comma separated values) : テキストファイル
  - RData 形式 : R の内部表現を用いたバイナリーファイル
  - Excel 形式 : RStudio の読み込み機能が利用可能
- データフレームを対象とした扱いを整理する

作業ディレクトリ

- R は **作業ディレクトリ** で実行される
  - ファイルは作業ディレクトリに存在するものとして扱われる
  - それ以外のファイルを扱う場合はパスを含めて指定する
- 作業ディレクトリの確認の仕方
  - コンソールの上部の表示
  - 関数 `getwd()`
- 作業ディレクトリの変更の仕方
  - **Session** メニューの **Set Working Directory** で指定
    - 読み込んだファイルの場所を選択
    - Files Pane の場所を選択
    - ディレクトリを直接選択
  - 関数 `setwd()`

関数 `getwd()/setwd()` の使い方

- コンソール・RScript からの作業ディレクトリの操作

```
## 作業ディレクトリの確認 (環境によって実行結果が異なる)
getwd()
## 作業ディレクトリの移動 (環境によって指定の仕方も異なる)
setwd("~/Documents") # ホームディレクトリ下の「書類」フォルダに移動
```

  - 作業ディレクトリはコンソールのタブにも表示されている

関数 write.csv()

- データフレームを CSV ファイルへ書き出す関数

```
write.csv(x, file = "ファイル名")
## x: 書き出すデータフレーム
## file: 書き出すファイルの名前（作業ディレクトリ下，またはパスを指定）
```

  - 他にも細かい指定ができるので詳しくはヘルプを参照

関数 write.csv() の使い方

- CSV ファイルの書き出しの例

```
## 関数 write.csv() の使い方（CSV ファイルの操作）
(my_data <- subset(airquality,
                   subset = Ozone>120,
                   select = -Temp)) # データフレームの作成
dim(my_data) # データフレームの大きさを確認
## 作業ディレクトリの中に data というフォルダを用意しておく
write.csv(my_data,file="data/my_data.csv") # csv ファイルとして書き出し
```

```
    Ozone Solar.R Wind Month Day
62    135     269  4.1     7   1
99    122     255  4.0     8   7
117   168     238  3.4     8  25
[1] 3 5
```

関数 read.csv()

- CSV ファイルからデータフレームを読み込む関数

```
read.csv(file = "ファイル名", header = TRUE,
         row.names, fileEncoding)
## file: 読み込むファイルの名前（作業ディレクトリ下，またはパスを指定）
## header: 1 行目を列名として使うか否か
## row.names: 行名の指定（行名を含む列番号/列名，または行名の直接指定が可能）
## fileEncoding: 文字コードの指定（日本語の場合，主に使うのは "utf8","sjis"）
```

  - 他にも細かい指定ができるので詳しくはヘルプを参照
  - 必要に応じて関数 read.table() , scan() なども参考に

関数 read.csv() の使い方

- CSV ファイルの読み込みの例

```
## 関数 read.csv() の使い方（CSV ファイルの操作）
(new_data <- read.csv(file="data/my_data.csv", # 前の例のファイル
                      row.names=1)) # 1 列目を行名に指定
dim(new_data) # 正しく読み込めたか大きさを確認
```

```
    Ozone Solar.R Wind Month Day
62    135     269  4.1     7   1
99    122     255  4.0     8   7
117   168     238  3.4     8  25
[1] 3 5
```

関数 save()

- RData ファイルへ書き出す関数

```
save(..., file = "ファイル名")
## ...: 保存するオブジェクト名（複数可，データフレーム以外も可）
## file: 書き出すファイルの名前（作業ディレクトリ下，またはパスを指定）
```

- CSV 形式と異なり**複数**のデータフレームを 1 つのファイルに保存することができる

関数 save() の使い方

- RData ファイルの書き出しの例

```
### 関数 save() の使い方（RData ファイルの操作）
(my_data_1 <- subset(airquality, Temp>95, select=-Ozone))
(my_data_2 <- subset(airquality, Temp<57, select=-Ozone))
dim(my_data_1); dim(my_data_2) # 大きさを確認
save(my_data_1,my_data_2,file="data/my_data.rdata") # RData 形式で書き出し
```

```
    Solar.R Wind Temp Month Day
120     203  9.7   97     8  28
122     237  6.3   96     8  30
  Solar.R Wind Temp Month Day
5      NA 14.3   56     5   5
[1] 2 5
[1] 1 5
```

関数 load()

- RData ファイルから読み込む関数

```
load(file = "ファイル名")
## file: 読み込むファイルの名前（作業ディレクトリ下，またはパスを指定）
```

- 同じ名前のオブジェクトがあると上書きするので注意

関数 load() の使い方

- RData ファイルの読み込みの例

```
## 関数 load() の使い方（RData ファイルの操作）
(my_data_1 <- subset(airquality, Ozone > 160)) # 新たに作成
load(file="data/my_data.rdata") # RData 形式の読み込み
my_data_1 # save したときの名前で読み込まれ上書きされる
my_data_2
```

```
    Ozone Solar.R Wind Temp Month Day
117   168     238  3.4   81     8  25
    Solar.R Wind Temp Month Day
120     203  9.7   97     8  28
122     237  6.3   96     8  30
  Solar.R Wind Temp Month Day
5      NA 14.3   56     5   5
```

csv 形式の操作 (package::readr)

- 日本語などの扱いに問題がある場合に推奨

```
install.packages("readr") # Package タブを使っても可能
```

- 関数 write_csv() : csv ファイルの書き出し

```
write_csv(x, file="ファイル名") # 行名は書き出されない
```

- 関数 read_csv() : csv ファイルの読み込み

```
y <- read_csv(file="ファイル名") # 行名を列から付けるオプションはない
```

- 行名の扱いに違いがあるので注意

演習

練習問題

- 以下のデータを読み込み，操作を行ってみよう．
  - https://www.e-stat.go.jp より取得したデータ
    (地域から探す / 全県を選択 / 項目を選択してダウンロード)
  - データファイル (文字コード : utf8)
    - jpdata1.csv : 県別の対象データ
    - jpdata2.csv : 対象データの内容説明
    - jpdata3.csv : 県と地域の対応関係
  - 作業ディレクトリの data 内に置いて読み込む

```
jp_data <- read.csv(file="data/jpdata1.csv", fileEncoding="utf8", row.names=1)
jp_item <- read.csv(file="data/jpdata2.csv", fileEncoding="utf8")
jp_area <- read.csv(file="data/jpdata3.csv", fileEncoding="utf8")
```

  - 日本語に問題がある場合は英語版を読み込む

```
jp_data_en <- read.csv(file="data/jpdata1-en.csv", row.names=1)
jp_area_en <- read.csv(file="data/jpdata3-en.csv")
```

データの集計

集約のための関数

- データを集約するために用意されている関数群
  - 関数 sum(): 総和
  - 関数 mean(): 平均
  - 関数 max(): 最大値
  - 関数 min(): 最小値
  - 関数 summary(): 基本統計量
- これ以外にも集約を行なう関数は沢山ある

集約の関数の使い方

- 練習問題のデータの集計を行う

```
jp_data <- read.csv(file="data/jpdata1.csv", # ファイルの指定
                    row.names=1, # 第 1 列を用いて各行の名前を設定
                    fileEncoding="utf8") # 文字コードの指定
## 一度読み込んでいれば上の行は不要
sum(jp_data$人口) # 全国の総人口 (列名で選択)
mean(jp_data[,4]) # 面積の平均値 (行列として列を選択)
median(jp_data[[4]]) # 面積の中央値 (リストとして列を選択)
```

```
min(jp_data["若年"])  # 若年人口の最小値 (列名で選択)
with(jp_data,max(老人))  # 老年人口の最大値 (関数 with() を利用)
```

```
[1] 126708000
[1] 793554.5
[1] 609719
[1] 72000
[1] 3160000
```

関数 apply()

- 列あるいは行ごとの計算を行う関数

```
apply(X, MARGIN, FUN)
## X: データフレーム
## MARGIN: 行 (1) か列 (2) かを指定
## FUN: 計算すべき統計量の関数
```

  - 変数名が全て大文字で定義されているので注意
  - 総和や平均は専用の関数も用意されている
    - 行和・列和 : **rowSums()/colSums()**
    - 行の平均・列の平均 : **rowMeans()/colMeans()**

関数 apply() の使い方

- 抽出したデータの集計を行う

```
### 関数 apply() の使い方
x <- subset(jp_data, select=婚姻:失業) # 抽出
colMeans(x) # 各列の平均
apply(x, 2, max) # 列ごとの最大値
sapply(x, max)   # 上と同じ (help("sapply") を参照)
## 自作関数の適用 (関数に名前を付けないで利用することができる)
apply(x, 2, function(z){return(sum(z>mean(z)))}) # 平均より大きいデータ数
## return を省略すると関数内で最後に評価された値が返り値になる
## apply(x, 2, function(z){sum(z>mean(z))}) # 慣れたらこちらでも可
```

```
    婚姻      離婚      失業
4.437021 1.631064 4.221277
婚姻 離婚 失業
6.19 2.41 6.30
婚姻 離婚 失業
6.19 2.41 6.30
婚姻 離婚 失業
  20   22   25
```

関数 aggregate()

- 各行をグループにまとめて統計量を計算する関数

```
aggregate(x, by, FUN)
## x: データフレーム
## by: 各行が属するグループを指定するベクトルをリストで与える (複数可)
## FUN: 求めたい統計量を計算するための関数
aggregate(x, data, FUN)
## x: 条件式 (formula)
## data: データフレーム
```

```
## FUN: 求めたい統計量を計算するための関数
```

- 同様な目的に関数 tapply() も利用可

関数 aggregate() の使い方

- 同じ値を持つグループごとの合計値を求める

```
### 関数 aggregate() の使い方
## 人口から面積まで地方ごとの平均値を計算
x <- subset(jp_data, select = 人口:面積)
aggregate(x, by = list(地方 = jp_area$地方), FUN = sum)
```

```
    地方      人口     若年      老人     面積
1   関東 43248000 5159000 10948000 3243305
2   近畿 22431000 2770000  6291000 3312565
3   九州 14360000 1951000  4088000 4451160
4   四国  3788000  449000  1223000 1880366
5   中国  7369000  932000  2243000 3192167
6   中部 21356000 2726000  6008000 6680678
7   東北  8836000 1016000  2716000 6694744
8 北海道  5320000  588000  1632000 7842078
```

- 代入せずにまとめて書くこともできる

```
aggregate(subset(jp_data, select = 人口:面積),
          by = list(地方 = jp_area$地方),
          FUN = sum)
```

```
    地方      人口     若年      老人     面積
1   関東 43248000 5159000 10948000 3243305
2   近畿 22431000 2770000  6291000 3312565
3   九州 14360000 1951000  4088000 4451160
4   四国  3788000  449000  1223000 1880366
5   中国  7369000  932000  2243000 3192167
6   中部 21356000 2726000  6008000 6680678
7   東北  8836000 1016000  2716000 6694744
8 北海道  5320000  588000  1632000 7842078
```

- 以下も同じ結果を返す

```
y <- transform(x, 地方 = jp_area$地方) # データフレームを変更
aggregate( . ~ 地方, # 右辺で条件付けて左辺 (右辺以外) を計算
          data = y, FUN = sum)
```

```
    地方      人口     若年      老人     面積
1   関東 43248000 5159000 10948000 3243305
2   近畿 22431000 2770000  6291000 3312565
3   九州 14360000 1951000  4088000 4451160
4   四国  3788000  449000  1223000 1880366
5   中国  7369000  932000  2243000 3192167
6   中部 21356000 2726000  6008000 6680678
7   東北  8836000 1016000  2716000 6694744
8 北海道  5320000  588000  1632000 7842078
```

- help("transform") を参照

- まとめて書くこともできる

```
aggregate( . ~ 地方, # 右辺で条件付けて左辺（右辺以外）を計算
          data = transform(subset(jp_data, select = 人口:面積),
                           地方 = jp_area$地方),
          FUN = sum)
```

```
    地方      人口     若年      老人     面積
1   関東 43248000 5159000 10948000 3243305
2   近畿 22431000 2770000  6291000 3312565
3   九州 14360000 1951000  4088000 4451160
4   四国  3788000  449000  1223000 1880366
5   中国  7369000  932000  2243000 3192167
6   中部 21356000 2726000  6008000 6680678
7   東北  8836000 1016000  2716000 6694744
8 北海道  5320000  588000  1632000 7842078
```

- 複数の条件でグループ分け

```
## 地方と，人口が中央値以下か否かでグループ分けして平均値を計算
aggregate(x,
          by = list(地方 = jp_area$地方,
                    過疎 = with(jp_data, 人口<=median(人口))),
          FUN = sum)
```

```
     地方  過疎      人口     若年      老人     面積
1    関東 FALSE 43248000 5159000 10948000 3243305
2    近畿 FALSE 18725000 2295000  5222000 2069269
3    九州 FALSE  6872000  912000  1915000 1239600
4    中国 FALSE  4736000  611000  1376000 1559395
5    中部 FALSE 17551000 2257000  4866000 4971734
6    東北 FALSE  4205000  500000  1200000 2106612
7  北海道 FALSE  5320000  588000  1632000 7842078
8    近畿  TRUE  3706000  475000  1069000 1243296
9    九州  TRUE  7488000 1039000  2173000 3211560
10   四国  TRUE  3788000  449000  1223000 1880366
11   中国  TRUE  2633000  321000   867000 1632772
12   中部  TRUE  3805000  469000  1142000 1708944
13   東北  TRUE  4631000  516000  1516000 4588132
```

- 別の書き方

```
aggregate( . ~ 地方 + 過疎,
          FUN = sum, # + で条件を追加
          data = transform(subset(jp_data, select = 人口:面積),
                           地方 = jp_area$地方,
                           過疎 = 人口<=median(人口)))
```

```
     地方  過疎      人口     若年      老人     面積
1    関東 FALSE 43248000 5159000 10948000 3243305
2    近畿 FALSE 18725000 2295000  5222000 2069269
3    九州 FALSE  6872000  912000  1915000 1239600
4    中国 FALSE  4736000  611000  1376000 1559395
5    中部 FALSE 17551000 2257000  4866000 4971734
6    東北 FALSE  4205000  500000  1200000 2106612
7  北海道 FALSE  5320000  588000  1632000 7842078
8    近畿  TRUE  3706000  475000  1069000 1243296
9    九州  TRUE  7488000 1039000  2173000 3211560
10   四国  TRUE  3788000  449000  1223000 1880366
11   中国  TRUE  2633000  321000   867000 1632772
12   中部  TRUE  3805000  469000  1142000 1708944
13   東北  TRUE  4631000  516000  1516000 4588132
```

演習

練習問題

- サンプルデータ (jpdata) の整理をしてみよう.
  - 県別の人口密度を求めよ.
  - 地方別の人口密度を求めよ.
    - 県ごとに人口が異なるので単純に人口密度を平均してはいけない.
  - 地方別の 1000 人当たりの婚姻・離婚数を概算せよ.
    - データの記述では「人口 1000 人当たり」とあるが，この「人口」とは若年層は婚姻不可として除いた「婚姻可能な人口 1000 人当たり」と考えて計算しなさい.

次回の予定

- 可視化の重要性
- 基本的な描画
- 分布の視覚化
- 比率の視覚化
- 多次元データの視覚化