

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

データ構造

- R に用意されている基本的なデータ構造
 - ベクトル (vector): 1次元配列
 - 行列 (matrix): 2次元配列
 - 配列 (array): 多次元配列
 - データフレーム (data.frame, tibble): 表 (2次元配列)
- 特殊なもの
 - リスト (list): オブジェクトの集合

データフレーム

- 複数の個体について、いくつかの属性を集計した表
 - 長さの等しい列ベクトルをまとめたもの
 - 各列のデータ型はバラバラでも良い
- データフレームの例

ある小学校の1年生の身長・体重・性別・血液型のデータ

名前	身長 [cm]	体重 [kg]	性別	血液型
太郎	108	19	男	B
花子	116	21	女	O
次郎	130	25	男	AB
...

- (特殊な) 行列 でもあり リスト でもある

tidyverse パッケージ

- データ操作とグラフィクスの拡張 (再掲)
 - tidyverse : Hadley Wickham @posit による拡張パッケージ集
 - * <https://www.tidyverse.org/packages/>
 - * <https://tidyverse.tidyverse.org/>
- パッケージ集の利用には以下が必要

```
#' 最初に一度だけ以下のいずれかを実行しておく
#' - Package タブから tidyverse をインストール
#' - コンソール上で次のコマンドを実行 'install.packages("tidyverse")'
#' tidyverse パッケージの読み込み
library(tidyverse)
```

- データフレームの拡張である tibble を利用

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)
 - `'?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
- `tibble` 形式への変換

```
aq_tbl <- as_tibble(airquality) # tibble 形式へ変換
aq_df <- airquality # 比較のため data.frame を別名で定義 (コピー)
aq_tbl # 最初の 10 行のみ表示される
aq_df # 全てのデータが表示される
print(aq_tbl, n = 15) # 最初の 15 行が表示される. '?tibble::print.tbl' を参照
print(aq_tbl, n = Inf) # 全て表示
head(aq_df, n = 10) # 最初の 10 行が表示される. tibble でも同様
tail(aq_df, n = 10) # 最後の 10 行が表示される. tibble でも同様
```

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**

```
aq_tbl[1,2] # 1 行 2 列の要素を選択
aq_tbl[-seq(2, nrow(aq_tbl), by = 2),] # 偶数行を除外
aq_tbl[,c(TRUE,FALSE,TRUE,TRUE,FALSE,TRUE)] # 1,3,4,6 列を選択
aq_tbl["Ozone"] # Ozone 列を選択. aq_tbl[, "Ozone"] でも同様
aq_tbl[,c("Ozone", "Wind")] # 複数列の選択も同様
aq_tbl[[ "Ozone" ]] # リストとして扱い 1 列を選択するとベクトルになる
aq_tbl$Ozone # 同上
```

- **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 (論理値)

行の選択 (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
aq_tbl[20:30,] # 20-30行を抽出
```

- 条件の指定

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

- 条件に合致する行の抽出

```
#' 関数 which() で TRUE となる行番号を抽出して指定
which(with(aq_tbl, Ozone>100 & Wind<=5))
aq_tbl[which(with(aq_tbl, Ozone>100 & Wind<=5)), ]
```

列の選択 (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
aq_tbl[which(with(aq_tbl, Ozone>100 & Wind <= 5)), c(1,5,6)]
```

- 列名による指定

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

- 列名による指定 (1つの場合)

```
#' 1つの列の場合はリストの操作として取り出せるが、ベクトルとなることに注意
aq_tbl[which(with(aq_tbl, Ozone > 100 & Wind <= 5)),]$Month # ベクトル
aq_tbl[which(with(aq_tbl, Ozone > 100 & Wind <= 5)),][["Month"]] # 同上
aq_tbl[which(with(aq_tbl, Ozone > 100 & Wind <= 5)), "Month"] # データフレーム
```

行の選択

- 関数 `dplyr::filter()` : 条件を指定して行を選択

```
filter(.data, ..., .by = NULL, .preserve = FALSE)
#' .data: データフレーム
#' ...: 行に関する条件
#' .by: グループ化を指定 (実験的な実装)
#' .preserve: グループ化を維持するか指定 (実験的な実装)
#' 詳細は '?dplyr::filter' を参照
```

- 行に関する条件指定には以下を用いることができる
 - 等号: `==` (否定は `!=`)
 - 不等号: `<`, `>`, `<=`, `>=`
 - 論理式: `&` (かつ), `|` (または)

列の選択

- 関数 `dplyr::select()` : 条件を指定して列を選択

```
select(.data, ...)
#' .data: データフレーム
#' ...: 列に関する条件 (列の番号, 名前, 名前に関する条件式を利用する)
#' 詳細は '?dplyr::select' を参照
```

- 条件指定には例えば以下のような方法がある
 - 含まない: !列名, !c(列名, 列名, ...), !(列名: 列名)
 - 特定の文字列で始まる: starts_with("文字列")
 - 特定の文字列で終わる: ends_with("文字列"),
 - 組み合わせ: & (かつ), | (または)

パイプ演算子

- 処理を順次結合する演算子 (いくつか定義がある)
 - |> (base R で定義; この講義ではこちらで記述する)
 - %>% (package::magrittr)
- データフレームの部分集合の取得

```
#!/ 素朴の方法の例は以下のように書ける
aq_tbl |> filter(Ozone > 100 & Wind <= 5) |> select(1,5,6)
aq_tbl |> filter(Ozone > 100 & Wind <= 5) |> select(Month,Day)
#!/ 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
aq_tbl |> filter(!is.na(Ozone) & Day %in% c(5,10)) |> select(Wind:Day)
#!/ Ozone が 120 以上か, または Wind が 3 以下の Temp 以外の列を抽出
aq_tbl |> filter(Ozone > 120 | Wind <= 3) |> select(!Temp)
```

実習

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さない。
 - 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 の処理は **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
 - 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
 - **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()`
- プロジェクトでは適切に設定される

作業ディレクトリの操作

- 関数 `base::getwd()` : 作業ディレクトリの確認

```
getwd() # 環境によって実行結果は異なる
```

- 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 `base::setwd()` : 作業ディレクトリの変更

```
setwd("~/Documents") # ホームディレクトリ下の「書類」フォルダに移動
```

- 環境によって指定の仕方は異なる

CSV ファイルの操作

- 関数 `readr::write_csv()` : ファイルの書き出し

```
write_csv(x,          # データフレーム
           file, ...) # ファイル名
#' 細かなオプションについては '?readr::write_csv' を参照
```

- 関数 `readr::read_csv()` : ファイルの読み込み

```
read_csv(file, ...) # ファイル名
#' 細かなオプションについては '?readr::read_csv' を参照
```

- `tibble` クラスとして読み込まれる

- 書き出しの例

```
## 関数 write_csv() の使い方 (CSV ファイルの操作)
my_data <- # データフレームの整理
aq_tbl |>
  filter(Ozone > 80) |> # Ozone が 80 を越える日を抽出
  select(!Temp)        # 温度は除く
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, # 保存するデータフレーム
          file = "data/my_data.csv") # (場所と)ファイル名
```

- 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
dim(new_data) # 正しく読み込めたか大きさを確認
```

RData ファイルの操作

- 関数 base::save() : ファイルの書き出し

```
save(..., # 保存するオブジェクト (複数可, データフレーム以外も可)
      list = character(), # 保存するオブジェクトの名前 (文字列) でも指定可能
      file = stop("'file' must be specified"), ...) # ファイル名
#' 細かなオプションについては '?base::save' を参照
```

- 複数のオブジェクトをまとめて保存することができる

- 関数 base::load() : ファイルの読み込み

```
load(file, ...) # ファイル名
#' 細かなオプションについては '?base::load' を参照
```

- 同じ名前のオブジェクトがあると上書きするので注意

- 書き出しの例

```
#' 関数 save() の使い方 (RDataファイルの操作)
my_data_1 <- aq_tbl |> filter(Temp > 90) |> select(!Ozone)
my_data_2 <- aq_tbl |> filter(Temp < 60) |> select(!Ozone)
dim(my_data_1); dim(my_data_2) # 大きさを確認
save(my_data_1, my_data_2, # 保存するオブジェクトを列挙
     file = "data/my_data.rdata") # ファイル名
```

- 読み込みの例

```
#' 関数 load() の使い方 (RDataファイルの操作)
my_data_1 <- aq_tbl |> filter(Ozone > 160) # 新たに作成
load(file = "data/my_data.rdata") # ファイル名
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
```

実習

練習問題

- 以下のデータを読み込んで操作してみよう
 - 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
 - <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)
 - 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")
jp_item <- read_csv(file = "data/jpdata2.csv")
jp_area <- read_csv(file = "data/jpdata3.csv")
```

- 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")
jp_area_en <- read_csv(file = "data/jpdata3-en.csv")
```

データの集計

統計量の計算

- データを集約した値 = 統計量
 - 関数 `base::sum()`: 総和を計算する
 - 関数 `base::mean()`: 平均
 - 関数 `base::max()`: 最大値
 - 関数 `base::min()`: 最小値
 - 関数 `stats::median()`: 中央値
 - 関数 `stats::quantile()`: 分位点
- これ以外にも沢山あるので調べてみよう

集約のための関数の使い方

- データの集計の例

```
#' 練習問題のデータを用いる
sum(jp_data$人口)      # 全国の総人口 (列名でベクトルを選択)
sum(jp_data[, "人口"]) # 1列のデータフレームとして計算
jp_data |> select(人口) |> sum() # 同上
mean(jp_data[, 5])    # 1列のデータフレームではエラーになる
mean(jp_data[[5]])    # ベクトルとして抜き出す必要がある
median(jp_data[[5]])  # 面積の中央値 (リストとして列を選択)
jp_data |> select(5) |> median() # データフレームなので動かない
min(jp_data["若年"])  # 若年人口の最小値 (列名で選択)
jp_data |> select("若年") |> min() # 同上
with(jp_data, max(老人)) # 老年人口の最大値 (関数 with() を利用)
jp_data |> select("老人") |> max() # 同上
```

- ベクトルでないと正しく動かない関数 (`mean`, `median`) もあるので注意

列ごとの集約

- 関数 `dplyr::summarise()`: 列ごとに計算する

```
summarise(.data, ..., .by = NULL, .groups = NULL)
#' .data: データフレーム
#' ...: 求めたい統計量を計算するための処理を記述
#' .by: グループ化を指定 (実験的な実装)
#' .groups: グループ化の結果を指定 (実験的な実装)
```

- 集計値の算出

```
#' 練習問題のデータを用いた例
jp_data |> summarise(平均失業率 = mean(失業), 件数 = n()) # 失業の列の平均
jp_data |> summarise(across(婚姻:失業, median)) # 婚姻から失業の列の中央値
```

```
jp_data |> summarise(across(!県名, max)) # 県名の列以外の最大値
jp_data |> summarise(across(when(is.double), min)) # 数値列の最小値
```

グループごとの操作

- 関数 `dplyr::group_by()` : グループ化を行う

```
group_by(.data, ..., .add = FALSE, .drop = group_by_drop_default(.data))
#' .data: データフレーム
#' ...: グループ化を行う項目を含む列や条件を記述
#' .add: グループ化の上書きを制御 (既定値 FALSE は上書き)
#' .drop: グループ化に関与しない因子の扱い方
```

- グループごとに集計

```
#' 練習問題のデータを用いた例
#' 地方ごとに人口から面積の列の合計を計算する
jp_data |>
  mutate(地方 = as_factor(jp_area[["地方"]])) |> # 地方の情報を付加
  group_by(地方) |> # 地方ごとにグループ化
  summarize((across(人口:面積, sum))) # グループごとに集計
```

実習

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