

# 情報処理応用演習

## 第2講: データハンドリング

SONG Jaehyun (同志社大学)

2020/8/24

# 本日の内容

## dplyrとtidyrパッケージの使い方

- パイプ演算子 (%>%) の使い方
- 行と列の抽出
- 行のソート
- 表の中身を要約
  - 列の平均値、標準誤差、分散など
- 表の中身をグループごとに要約
- 表の結合
- データ分析に適したデータ (Tidy data; 整然データ) とは何か
  - 3日目の内容は整然データを用いた可視化を解説
- Wide型とLong型の変換など

# Tidyverseとパイプ演算子

# Tidyverseの世界



データサイエンスのために考案された、強い信念と思想に基づいたRパッケージの集合

- Tidyverseに属するパッケージは思想、文法およびデータ構造を共有
  - `dplyr`、`tidyr`、`readr`、`ggplot2`など
- オブジェクトはパイプ (`%>%`)で繋ぐ

# パイプ演算子

Tidyverseにおいてオブジェクトは`%>%`で繋がっている。

- 既存の書き方: 書き方と読み方が逆
  - 一般的なプログラミング言語共通
  - 書き方: `print(sum(X))` (`print`、 `sum`、 `X`の順で書く)
  - 読み方1: `X`を`sum()`し、 `print()`する (コードの順番と逆)
  - 読み方2: `print()`する内容は`sum()`で、 `sum()`は`X`に対して行う (直感的でない読み方)
- Tidyverseな書き方: 書き方と読み方が一致
  - 今どきのRの書き方
  - 書き方: `X %>% sum() %>% print()`
  - 読み方: `X`を`sum()`し、 `print()`する

# パイプ演算子の原理

- `%>%`の左側を右側の最初の引数として渡すだけ
- `X %>% sum(na.rm = TRUE)`は`sum(X, na.rm = TRUE)`と同じ
- `X %>% sum(na.rm = TRUE) %>% print()`は`print(sum(X, na.rm = TRUE))`と同じ

```
# 既存の書き方
X <- c(2, 3, 5, NA, 11)
print(sum(X, na.rm = TRUE))
```

```
## [1] 21
```

```
# Tidyverseな書き方
X <- c(2, 3, 5, NA, 11)
X %>% sum(na.rm = TRUE) %>% print()
```

```
## [1] 21
```

# dplyr: 抽出

# dplyrとは

- 表形式データ(データフレームやtibble)を操作するパッケージ
- 前回の講義で解説した行・列の抽出も簡単に可能
- Tidyverseパッケージを読み込む際に自動的に読み込まれる

```
library(tidyverse)
```

# 実習用データ

Countries.csv: 186カ国 の社会経済・政治体制のデータ

```
df <- read_csv("Data/Countries.csv")
print(df, n = 5)

## # A tibble: 186 x 18
##   Country Population    Area     GDP     PPP GDP_per_capita
##   <chr>        <dbl>    <dbl>    <dbl>    <dbl>        <dbl>
## 1 Afghan...    38928346 6.53e5  1.91e4  82737.       491.
## 2 Albania      2877797  2.74e4  1.53e4  39658.       5309.
## 3 Algeria      43851044 2.38e6  1.70e5  496572.      3876.
## 4 Andorra       77265  4.70e2  3.15e3     NA        40821.
## 5 Angola        32866272 1.25e6  9.46e4  218533.      2879.
## # ... with 181 more rows, and 12 more variables: PPP_per_capita <dbl>,
## #   G7 <dbl>, G20 <dbl>, OECD <dbl>, HDI_2018 <dbl>,
## #   Polity_Score <dbl>, Polity_Type <chr>, FH_PR <dbl>, FH_CL <dbl>,
## #   FH_Total <dbl>, FH_Status <chr>, Continent <chr>
```

- データのサイズ: 186カ国 × 18変数 (`dim(df)`でも確認可能)

# 列の選択

`select()`関数を使用

- 書き方: `select(データ, 変数名1, 変数名2, ...)`
- 推奨: `データ %>% select(変数名1, 変数名2, ...)`

```
# dfからCountry, Population, Area, GDP, PPP, HDI_2018列を抽出
df %>%
  select(Country, Population, Area, GDP, PPP, HDI_2018) %>%
  print(n = 5)
```

```
## # A tibble: 186 x 6
##   Country     Population      Area      GDP      PPP  HDI_2018
##   <chr>        <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 Afghanistan 38928346  652860  19101.  82737.  0.496
## 2 Albania      2877797   27400  15278.  39658.  0.791
## 3 Algeria      43851044 2381740 169988. 496572. 0.759
## 4 Andorra       77265     470    3154.    NA      0.857
## 5 Angola        32866272 1246700  94635.  218533. 0.574
## # ... with 181 more rows
```

# 変数指定の方法(1)

- 隣接した列を選択する: 「:」
- CountryからPPP列は隣接している-> Country:PPP

```
# dfからCountry~PPP, HDI_2018列を抽出  
df %>%  
  select(Country:PPP, HDI_2018) %>%  
  print(n = 5)
```

```
## # A tibble: 186 x 6  
##   Country     Population      Area      GDP      PPP  HDI_2018  
##   <chr>        <dbl>      <dbl>    <dbl>    <dbl>    <dbl>  
## 1 Afghanistan 38928346  652860  19101.  82737.  0.496  
## 2 Albania     2877797   27400  15278.  39658.  0.791  
## 3 Algeria     43851044 2381740 169988. 496572.  0.759  
## 4 Andorra      77265     470   3154.    NA     0.857  
## 5 Angola       32866272 1246700  94635.  218533.  0.574  
## # ... with 181 more rows
```

# 変数指定の方法(2)

- 特定の文字列で始まる列を選択: `starts_with()`
  - 特定の文字列で終わる列を選択: `ends_with()`
  - 特定の文字列を含む列を選択: `contains()`
- FHで始まる列の選択 -> `starts_with("FH")`

```
# dfからCountry列, FHで始まる列を抽出
df %>%
  select(Country, starts_with("FH")) %>%
  print(n = 5)
```

```
## # A tibble: 186 x 5
##   Country     FH_PR   FH_CL   FH_Total FH_Status
##   <chr>       <dbl>   <dbl>     <dbl>   <chr>
## 1 Afghanistan    13     14      27   NF
## 2 Albania        27     40      67   PF
## 3 Algeria         10     24      34   NF
## 4 Andorra         39     55      94   F
## 5 Angola          11     21      32   NF
## # ... with 181 more rows
```

# 変数指定の方法(3)

- 特定の列を除外する: !または-
  - :、starts\_with()などと組み合わせることも可能

```
# dfからGDP_per_capita~FH_Status列を除外
df %>%
  select(!GDP_per_capita:FH_Status) %>%
  print(n = 5)
```

```
## # A tibble: 186 x 6
##   Country     Population     Area     GDP     PPP Continent
##   <chr>        <dbl>      <dbl>    <dbl>    <dbl>   <chr>
## 1 Afghanistan 38928346  652860 19101.  82737. Asia
## 2 Albania      2877797   27400  15278.  39658. Europe
## 3 Algeria      43851044 2381740 169988. 496572. Africa
## 4 Andorra       77265      470   3154.    NA    Europe
## 5 Angola        32866272 1246700  94635. 218533. Africa
## # ... with 181 more rows
```

# 変数名(列名)の確認

names()関数を使う

```
# dfの変数名を出力
```

```
names(df)
```

```
## [1] "Country"          "Population"        "Area"  
## [4] "GDP"               "PPP"                 "GDP_per_capita"  
## [7] "PPP_per_capita"   "G7"                  "G20"  
## [10] "OECD"              "HDI_2018"           "Polity_Score"  
## [13] "Polity_Type"       "FH_PR"              "FH_CL"  
## [16] "FH_Total"          "FH_Status"         "Continent"
```

# 変数名(列名)の変更

2つの方法がある

- `select()`: 列の抽出と同時に、新しい列名が指定可能
- `rename()`: 列は温存したまま、列名のみ変更

# 変数名(列名)の変更: select()

- 抽出と同時に行うにはselect()内に=を使用
- 列名のみ変更の場合は、変更しない列も指定する必要があるため、やや面倒
- PopulationをJinkoへ、AreaをMensekiへ変更

```
df %>%
  select(Country,
         Jinko = Population, Menseki = Area,
         GDP:Continent) %>%
  print(n = 5)

## # A tibble: 186 x 18
##   Country   Jinko   Menseki     GDP      PPP  GDP_per_capita PPP_per_capita G
##   <chr>     <dbl>    <dbl>    <dbl>    <dbl>        <dbl>        <dbl> <dbl>
## 1 Afghan... 3.89e7  652860  1.91e4  82737.          491.        2125.
## 2 Albania  2.88e6   27400  1.53e4  39658.         5309.       13781.
## 3 Algeria  4.39e7  2381740 1.70e5  496572.        3876.       11324.
## 4 Andorra  7.73e4     470  3.15e3     NA        40821.        NA
## 5 Angola   3.29e7 1246700  9.46e4  218533.        2879.       6649.
## # ... with 181 more rows, and 10 more variables: G20 <dbl>, OECD <dbl>,
## #   HDI_2018 <dbl>, Polity_Score <dbl>, Polity_Type <chr>, FH_PR <dbl>,
## #   FH_CL <dbl>, FH_Total <dbl>, FH_Status <chr>, Continent <chr>
```

# 変数名(列名)の変更: `reanme()`

- `select()`と`=`の組み合わせと書き方は同じ
  - ただし、変更しない列名は指定しなくてもOK
- `Population`を`Jinko`へ、`Area`を`Menseki`へ変更

```
df %>%
  rename(Jinko = Population, Menseki = Area) %>%
  print(n = 5)
```

```
## # A tibble: 186 x 18
##   Country   Jinko Menseki     GDP      PPP GDP_per_capita PPP_per_capita G
##   <chr>     <dbl>   <dbl>   <dbl>   <dbl>       <dbl>       <dbl>   <dbl>
## 1 Afghan... 3.89e7  652860  1.91e4  82737.        491.       2125.
## 2 Albania  2.88e6   27400  1.53e4  39658.       5309.      13781.
## 3 Algeria  4.39e7  2381740 1.70e5  496572.      3876.      11324.
## 4 Andorra  7.73e4    470  3.15e3     NA      40821.       NA
## 5 Angola   3.29e7 1246700  9.46e4  218533.      2879.      6649.
## # ... with 181 more rows, and 10 more variables: G20 <dbl>, OECD <dbl>,
## #   HDI_2018 <dbl>, Polity_Score <dbl>, Polity_Type <chr>, FH_PR <dbl>,
## #   FH_CL <dbl>, FH_Total <dbl>, FH_Status <chr>, Continent <chr>
```

# 列の順番変更

2つの方法がある

- `select()`: ()内に指定した順番で列が抽出される
- `relocate()`: 指定した変数を特定の変数の前、または後ろに移動させる

# 列の順番変更: select()

G7からOECD列をCountryとPopulationの間へ移動

```
df %>%
  select(Country, G7:OECD,
         Population:PPP_per_capita, HDI_2018:Continent)
```

```
## # A tibble: 186 x 18
##   Country     G7     G20    OECD Population     Area     GDP     PPP GDP_per_capita
##   <chr>     <dbl>   <dbl>   <dbl>      <dbl>   <dbl>   <dbl>   <dbl>      <dbl>
## 1 Afghan...     0       0       0      38928346 6.53e5 1.91e4 8.27e4      499
## 2 Albania      0       0       0      2877797  2.74e4 1.53e4 3.97e4      530
## 3 Algeria      0       0       0      43851044 2.38e6 1.70e5 4.97e5      387
## 4 Andorra      0       0       0       77265  4.70e2 3.15e3 NA          4082
## 5 Angola        0       0       0      32866272 1.25e6 9.46e4 2.19e5      287
## 6 Antigu...     0       0       0       97929  4.40e2 1.73e3 2.08e3      1764
## 7 Argent...     0       1       0      45195774 2.74e6 4.50e5 1.04e6      994
## 8 Armenia       0       0       0      2963243  2.85e4 1.37e4 3.84e4      461
## 9 Austra...     0       1       1      25499884 7.68e6 1.39e6 1.28e6      5461
## 10 Austria      0       0       1      9006398  8.24e4 4.46e5 5.03e5      4955
## # ... with 176 more rows, and 9 more variables: PPP_per_capita <dbl>,
## #   HDI_2018 <dbl>, Polity_Score <dbl>, Polity_Type <chr>, FH_PR <dbl>
```

# 列の順番変更: relocate()

G7からOECD列をCountryの後ろへ移動

- .after = Countryを指定 (.に注意)
- .before = PopulationもOK

```
df %>%  
  relocate(G7:OECD, .after = Country)
```

```
## # A tibble: 186 x 18  
##   Country     G7     G20    OECD Population     Area     GDP     PPP GDP_per_capita  
##   <chr>     <dbl>   <dbl>   <dbl>      <dbl>   <dbl>   <dbl>   <dbl>      <dbl>  
## 1 Afghan...     0       0       0     38928346  6.53e5  1.91e4  8.27e4      4930  
## 2 Albania      0       0       0     2877797   2.74e4  1.53e4  3.97e4      5300  
## 3 Algeria      0       0       0     43851044  2.38e6  1.70e5  4.97e5      3870  
## 4 Andorra      0       0       0      77265   4.70e2  3.15e3    NA      4082  
## 5 Angola        0       0       0     32866272  1.25e6  9.46e4  2.19e5      2870  
## 6 Antigu...     0       0       0      97929   4.40e2  1.73e3  2.08e3      1764  
## 7 Argent...     0       1       0     45195774  2.74e6  4.50e5  1.04e6      9940  
## 8 Armenia       0       0       0     2963243   2.85e4  1.37e4  3.84e4      4610  
## 9 Austra...     0       1       1     25499884  7.68e6  1.39e6  1.28e6      5461  
## 10 Austria      0       0       1     9006398   8.24e4  4.46e5  5.03e5      4955
```

# 行の選択 (1)

filter()関数を使用

- 書き方: filter(データ, 条件1, 条件2, ...)
- 推奨: データ %>% filter(条件1, 条件2, ...)

```
# dfからContinentが"Europe"の行を抽出し、Country~PPP, HDI_2018列を抽出
df %>%
  filter(Continent == "Europe") %>%
  select(Country:PPP, HDI_2018) %>%
  print(n = 5)
```

```
## # A tibble: 50 x 6
##   Country     Population     Area      GDP      PPP HDI_2018
##   <chr>        <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 Albania     2877797  27400    15278.  39658.    0.791
## 2 Andorra      77265    470     3154.     NA       0.857
## 3 Armenia      2963243  28470    13673.  38446.    0.76
## 4 Austria      9006398  82409    446315. 502771.   0.914
## 5 Azerbaijan   10139177 82658    48048.  144556.   0.754
## # ... with 45 more rows
```

# 行の選択 (1): 注意事項1

`filter()`と`select()`の順番

- 先に`select()`を行うと、`Continent`列がなくなるため、`filter()`内の条件式が無効に
- `select()`に`Continent`が含まれるなら、問題なし

```
# select()の後にfilter()を使うと...
df %>%
  select(Country:PPP, HDI_2018) %>%
  filter(Continent == "Europe") %>%
  print(n = 5)

## Error: Problem with `filter()` input `..1`.
## x オブジェクト 'Continent' がありません
## i Input `..1` is `Continent == "Europe"`.
```

# 行の選択 (1): 注意事項2

欠損値を含む行の除去について

- 例) `PPP_per_capita`が欠損しているケースを除去
- `PPP_per_capita != NA`は使えない

```
df %>%
  filter(PPP_per_capita != NA) # ダメな例
```

- 代わりに`!is.na(PPP_per_capita)`を使用

```
# ダメな例
df %>%
  filter(!is.na(PPP_per_capita)) # 正しい例
```

# 行の選択 (2)

- 等号 (==) だけでなく、不等号も使用可能

```
# dfからContinentが"Asia" (条件1)、HDI_2018が0.8以上 (条件2)の行を抽出し、  
# Country~PPP, HDI_2018列を抽出  
df %>%  
  filter(Continent == "Asia", HDI_2018 >= 0.85) %>%  
  select(Country:PPP, HDI_2018)
```

```
## # A tibble: 6 x 6  
##   Country           Population     Area     GDP     PPP  HDI_2018  
##   <chr>              <dbl>     <dbl>    <dbl>    <dbl>    <dbl>  
## 1 Israel            8655535  21640  395099.  357633.  0.906  
## 2 Japan             126476461 364555 5081770. 5247581.  0.915  
## 3 South Korea       51269185  97230 1642383. 2046766.  0.906  
## 4 Saudi Arabia      34813871 2149690 792967. 1643080.  0.857  
## 5 Singapore          5850342    700  372063.  557255.  0.935  
## 6 United Arab Emirates 9890402  83600  421142.  650568.  0.866
```

# 行のソート (1)

`arrange()`関数を使用

- 書き方: `arrange(データ, 変数名1, 変数名2, ...)`
- 推奨: `データ %>% arrange(変数名1, 変数名2, ...)`

基本的には昇順 (小さい行が先にくる)

- 降順にする場合は`desc(変数名)`
- `変数名1`を基準にソートし、同点の場合は`変数名2`を基準に

# 行のソート

dfからアフリカのみを抽出し、Polity\_Scoreが低い行を上位にする。そして、CountryとPPP列のみ残す。

- Polity Scoreが高い（低い） = より民主主義（権威主義）に近い

```
df %>%
  filter(Continent == "Africa") %>%
  arrange(Polity_Score) %>%
  select(Country, Polity_Score)
```

```
## # A tibble: 54 x 2
##   Country          Polity_Score
##   <chr>              <dbl>
## 1 Eswatini            -9
## 2 Eritrea             -7
## 3 Equatorial Guinea  -6
## 4 Cameroon            -4
## 5 Congo (Brazzaville) -4
## 6 Egypt               -4
## 7 Morocco              -4
## 8 Sudan                -4
```

# 行のソート

- dfからアフリカのみを抽出し、Polity\_Scoreが低い行を上位に
- Polity\_Scoreが同点の場合、PPP\_per\_capitaが高い行を上位に
- CountryとPolity\_Score, PPP\_per\_capita列のみ残す
- Polity\_ScoreはPolityに、PPP\_per\_capitaはPPPと名前を変更

```
df %>%
  filter(Continent == "Africa") %>%
  arrange(Polity_Score, desc(PPP_per_capita)) %>%
  select(Country, Polity = Polity_Score, PPP = PPP_per_capita)
```

```
## # A tibble: 54 × 3
##   Country          Polity     PPP
##   <chr>            <dbl>    <dbl>
## 1 Eswatini         -9     8634.
## 2 Eritrea          -7     1860.
## 3 Equatorial Guinea -6    19458.
## 4 Egypt             -4    11198.
## 5 Morocco           -4     7554.
## 6 Sudan              -4     4063.
## 7 Cameroon          -4     3506.
```

# これまでの内容: データの抽出

いずれも元のデータの一部 (subset)が結果として出力される

- 列を抽出する
- 列を並び替える
- 行を抽出する
- 行を並び替える

これから的内容: データの拡張

- 人口を面積で割って人口密度を計算し、新しい列として追加する
- G7, G20, OECDいずれかに所属しているか否かを表す新しい変数を作成する

これから的内容: データの要約

- 大陸ごとに平均一人当たりGDPを計算する
- 大陸ごとに民主主義の度合いを計算する

# dplyr: データの拡張

# 新しい列の追加追加: mutate()

- データ %>% mutate(新しい変数名 = ベクトル)
  - ベクトルの長さはデータの行数と一致する必要あり
- 例) IDという列に1からnrow(df)の数列を入れ、Countryの前へ
  - nrow(データ)はデータの行数を出力する関数
  - .afterまたは.beforeを指定しないと、最後列へ

```
df %>%
  mutate(ID = 1:nrow(df), .before = Country) %>%
  print(n = 5)
```

```
## # A tibble: 186 x 19
##   ID Country Population Area GDP PPP GDP_per_capita PPP_per_ca
##   <int> <chr>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1 Afghanistan 38928346 6.53e5 1.91e4 82737. 491. 2
## 2 2 Albania      2877797 2.74e4 1.53e4 39658. 5309. 13
## 3 3 Algeria      43851044 2.38e6 1.70e5 496572. 3876. 11
## 4 4 Andorra       77265 4.70e2 3.15e3    NA 40821.
## 5 5 Angola        32866272 1.25e6 9.46e4 218533. 2879. 6
## # ... with 181 more rows, and 11 more variables: G7 <dbl>, G20 <dbl>,
## # OECD <dbl>, HDI_2018 <dbl>, Polity_Score <dbl>, Polity_Type <chr>/,79
```

# mutate()の仕組み

データ %>% mutate(新しい変数名 = ベクトル)

- ベクトルはデータの行数と一致するものなら何でも良い
- df\$Population / df\$Areaの場合、データの行数と一致するベクトルが出力される
  - 計算式を入れることが可能(こちらが主な使い方)

```
# PopulationをAreaで割った値をDensity列に格納し、Areaの後に
```

```
df %>%
  mutate(Density = Population / Area, .after = Area)
```

```
## # A tibble: 186 x 19
##   Country Population     Area Density      GDP      PPP GDP_per_capita
##   <chr>        <dbl>    <dbl>    <dbl>    <dbl>    <dbl>          <dbl>
## 1 Afghan...    38928346 6.53e5    59.6  1.91e4  8.27e4         491.
## 2 Albania     2877797  2.74e4   105.   1.53e4  3.97e4        5309.
## 3 Algeria     43851044 2.38e6    18.4  1.70e5  4.97e5        3876.
## 4 Andorra      77265  4.70e2   164.   3.15e3  NA            40821.
## 5 Angola       32866272 1.25e6   26.4  9.46e4  2.19e5        2879.
## 6 Antigu...    97929  4.40e2   223.   1.73e3  2.08e3       17643.
```

# mutate()の応用

```
# %>%が多すぎても可読性が落ちるので適宜、オブジェクトとして保存しながら進める
df2 <- df %>%
  filter(Population >= 10000000, Continent == "Asia") %>%
  mutate(Density = Population / Area, .after = Area)
```

```
df2 %>%
  arrange(desc(Density)) %>%
  select(Country:PPP)
```

```
## # A tibble: 25 x 6
##   Country     Population      Area Density     GDP     PPP
##   <chr>        <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 Bangladesh  164689383  130170    1265.  302571.  734108.
## 2 Taiwan      23816775   35410     673.   589391   1099030
## 3 South Korea 51269185   97230     527.   1642383.  2046766.
## 4 India       1380004385  2973190    464.   2875142.  9058692.
## 5 Philippines 109581078  298170    368.   376796.   887474.
## 6 Japan        126476461  364555    347.   5081770.  5247581.
## 7 Sri Lanka   21413249   62710     341.   84009.   287228.
## 8 Vietnam      97338579   310070    314.   261921.   742462.
## 9 Pakistan     220892340  770880    287.   278222.  1030292.
```

# mutate()の応用

先進国か否かを表すDeveloped列を追加

- G7, G20, OECDのいずれかに加入している場合、「先進国」と定義
- G7, G20, OECD加入有無は0/1 -> この合計をDevelopedとして格納
- Developedが1以上なら"Developed Country"、0なら"Etc"とする
  - ifelse() または case\_when() 関数を使用

```
# Country, GDP, G7, G20, OECD列のみ抽出し、df3に保存
df3 <- df %>%
  select(Country, GDP, G7:OECD)
df3
```

```
## # A tibble: 186 x 5
##   Country           GDP     G7     G20   OECD
##   <chr>       <dbl>  <dbl>  <dbl>  <dbl>
## 1 Afghanistan    19101.     0      0      0
## 2 Albania        15278.     0      0      0
## 3 Algeria         169988.    0      0      0
## 4 Andorra          3154.     0      0      0
## 5 Angola          94635.    0      0      0
```

# mutate()の応用

```
df3 %>%
  mutate(Developed = G7 + G20 + OECD)

## # A tibble: 186 x 6
##   Country           GDP     G7     G20    OECD Developed
##   <chr>        <dbl>  <dbl>  <dbl>  <dbl>      <dbl>
## 1 Afghanistan    19101.     0     0     0       0
## 2 Albania        15278.     0     0     0       0
## 3 Algeria        169988.    0     0     0       0
## 4 Andorra         3154.     0     0     0       0
## 5 Angola          94635.    0     0     0       0
## 6 Antigua and Barbuda 1728.    0     0     0       0
## 7 Argentina       449663.    0     1     0       1
## 8 Armenia          13673.    0     0     0       0
## 9 Australia        1392681.   0     1     1       2
## 10 Austria         446315.   0     0     1       1
## # ... with 176 more rows
```

# mutate()の応用

```
df3 %>%
  mutate(Developed = G7 + G20 + OECD,
        Developed = ifelse(Developed >= 1, "Developed Country",
                           "Etc"))

## # A tibble: 186 x 6
##   Country           GDP     G7     G20     OECD Developed
##   <chr>       <dbl>  <dbl>  <dbl>  <dbl>   <chr>
## 1 Afghanistan    19101.     0      0      0 Etc
## 2 Albania        15278.     0      0      0 Etc
## 3 Algeria         169988.    0      0      0 Etc
## 4 Andorra          3154.     0      0      0 Etc
## 5 Angola           94635.    0      0      0 Etc
## 6 Antigua and Barbuda  1728.     0      0      0 Etc
## 7 Argentina        449663.    0      1      0 Developed Country
## 8 Armenia           13673.     0      0      0 Etc
## 9 Australia         1392681.   0      1      1 Developed Country
## 10 Austria          446315.    0      0      1 Developed Country
## # ... with 176 more rows
```

# ifelse() と case\_when()

## ifelse() の使い方 (if\_else() もOK)

```
ifelse(条件式, TRUEの場合の返り値, FALSEの場合の返り値)
```

# 例

```
ifelse(Developed >= 1, "Developed Country", "Etc")
```

## case\_when() の使い方

- 条件が複数の場合には推奨
- 返り値として欠損値は指定できない

```
case_when(条件1 ~ 条件1がTRUEの場合の返り値,  
         条件2 ~ 条件2がTRUEの場合の返り値,  
         ...,  
         TRUE ~ その他の場合の返り値)
```

# 例

```
case_when(Developed >= 1 ~ "Developed Country",  
          TRUE ~ "Etc")
```

# mutate()の応用

```
df4 <- df %>%
  select(Country, GDP, PPP, Continent)
```

# mutate()の応用

```
df4 %>%
  mutate(Continent_JP = recode(Continent,
                                "Africa"   = "アフリカ",
                                "America"  = "アメリカ",
                                "Asia"     = "アジア",
                                "Europe"   = "ヨーロッパ",
                                .default   = "オセアニア"))
```

```
## # A tibble: 186 x 5
##   Country           GDP     PPP Continent Continent_JP
##   <chr>        <dbl>   <dbl>   <chr>    <chr>
## 1 Afghanistan      19101.   82737. Asia     アジア
## 2 Albania          15278.   39658. Europe   ヨーロッパ
## 3 Algeria          169988.  496572. Africa   アフリカ
## 4 Andorra          3154.     NA      Europe   ヨーロッパ
## 5 Angola           94635.   218533. Africa   アフリカ
## 6 Antigua and Barbuda 1728.    2083.   America  アメリカ
## 7 Argentina         449663.  1036721. America  アメリカ
## 8 Armenia           13673.   38446. Europe   ヨーロッパ
## 9 Australia         1392681. 1275027. Oceania オセアニア
```

# recode():値の置換

- `.default`は「その他の場合」を意味する

```
recode(基になる変数名,  
       "基の値1" = "新しい値1",  
       "基の値2" = "新しい値2",  
       ....,  
       .default   = "新しい変数")
```

# dplyr: データの要約

# データ要約の例

記述統計量(要約統計量): 変数の性質を表す数値を計算すること

- 平均値: `mean()`
- 中央値: `median()`
- 分散: `var()`
- 標準偏差: `sd()`
- 分位数: `quantile()`
- 四分位範囲: `IQR()`
- 最小値: `min()`
- 最大値: `max()`
- 尖度: `e1071::kurtosis()`
- 歪度: `e1071::skewness()`
- その他

# データの要約: summarise() 関数

```
summarise(新しい変数名 = 関数(変数名))
```

- 新しい変数名は記述統計量が格納される任意の変数名
- 全く新しいデータが出力される
  - これまで既存のデータの一部を抽出したり、列を追加するなど、原型が残っていた

```
df %>%
  summarise(Pop_mean = mean(Population),
            Pop_sd   = sd(Population),
            Area_mean = mean(Area),
            Area_sd   = sd(Area))
```

```
## # A tibble: 1 x 4
##   Pop_mean     Pop_sd Area_mean     Area_sd
##       <dbl>     <dbl>     <dbl>     <dbl>
## 1 41737773. 151270298. 696069. 1872412.
```

# データの要約: summarise() 関数

ただし、単にある変数の記述統計料を計算するだけなら、`summarise()`使わない方が早い

```
mean(df$Population) # dfのPopulation列の平均値
```

```
## [1] 41737773
```

```
sd(df$Population) # dfのPopulation列の標準偏差
```

```
## [1] 151270298
```

```
mean(df$Area) # dfのArea列の平均値
```

```
## [1] 696069.2
```

```
sd(df$Area) # dfのArea列の標準偏差
```

```
## [1] 1872412
```

# データの要約: group\_by() 関数

- グループごとに記述統計量を計算したい
  - group\_by() を使用
  - 例) dplyrを使わず、大陸ごとにGDPの平均値を計算

```
mean(df$GDP[df$Continent == "Africa"])
```

```
## [1] 44941.25
```

```
mean(df$GDP[df$Continent == "America"])
```

```
## [1] 804002.1
```

(省略)

```
mean(df$GDP[df$Continent == "Oceania"])
```

```
## [1] 407528.6
```

# データの要約: group\_by() 関数

- group\_by(グループ変数名): グループごとの計算を行う

```
# 1. 大陸 (Continent)ごとにGDPの平均値と最小値、最大値を計算
# 2. GDPの平均値が高い順でソート
df %>%
  group_by(Continent) %>%
  # GDPが欠損している国もあるので、na.rm = TRUEを指定
  summarise(GDP_mean = mean(GDP, na.rm = TRUE),
            GDP_min   = min(GDP, na.rm = TRUE),
            GDP_max   = max(GDP, na.rm = TRUE)) %>%
  arrange(desc(GDP_mean))
```

```
## # A tibble: 5 x 4
##   Continent GDP_mean GDP_min   GDP_max
##   <chr>        <dbl>    <dbl>     <dbl>
## 1 America     804002.    596.  21544825
## 2 Asia        762373.   1674.  14762792.
## 3 Europe      458979.   1638.  3845630.
## 4 Oceania     407529.   5536.  1392681.
## 5 Africa       44941.    429.   448120.
```

# 便利な関数:n()

- n(): グループに属するケース数を計算
  - グループごとに国数とOECD加盟国数を計算し、加盟国の割合を計算する

```
df %>%
  group_by(Continent) %>%
  summarise(OECD      = sum(OECD),
            N_Countries = n()) %>%
  mutate(OECD_Prop = OECD / N_Countries)
```

```
## # A tibble: 5 x 4
##   Continent  OECD N_Countries OECD_Prop
##   <chr>      <dbl>     <int>      <dbl>
## 1 Africa        0         54        0
## 2 America       5         36       0.139
## 3 Asia          3         42       0.0714
## 4 Europe        27        50       0.54
## 5 Oceania       2          4       0.5
```

# dplyr: データの結合

# データの結合

## 行の結合と列の結合

- 行の結合: `bind_rows()` 関数
- 列の結合: `bind_cols()`、`*_join()` 関数

# 行の結合

列数と列名の一致を確認すること (列の順番は一致しなくてもOK)

```
# サンプルデータを作成する
data1 <- df %>%
  filter(Continent == "Oceania") %>%
  select(Country:GDP, Continent)

data2 <- df %>%
  filter(Continent == "Asia") %>%
  select(Country:GDP, Continent)

dim(data1) # data1のサイズ: 4行5列
```

```
## [1] 4 5
```

```
dim(data2) # data2のサイズ: 42行5列
```

```
## [1] 42 5
```

# 行の結合

```
# data1とdata2を結合しbinded_dataという名のオブジェクトとして格納  
binded_data <- bind_rows(data1, data2)  
dim(binded_data) # binded_dataのサイズ: 46行5列
```

```
## [1] 46 5
```

```
print(binded_data, n = 7)
```

```
## # A tibble: 46 x 5  
##   Country           Population      Area      GDP Continent  
##   <chr>              <dbl>     <dbl>     <dbl>   <chr>  
## 1 Australia        25499884  7682300 1392681. Oceania  
## 2 Fiji             896445    18270     5536. Oceania  
## 3 New Zealand     4842780   263820   206929. Oceania  
## 4 Papua New Guinea 8947024   452860   24970. Oceania  
## 5 Afghanistan     38928346  652860   19101. Asia  
## 6 Bahrain          1701575     760     38574. Asia  
## 7 Bangladesh       164689383 130170   302571. Asia  
## # ... with 39 more rows
```

# 列の結合(1)

```
# data1を列単位で2つに分割  
data1_a <- data1 %>%  
  select(Country, Population)  
  
data1_b <- data1 %>%  
  select(Area, GDP)
```

data1\_a

```
## # A tibble: 4 x 2  
##   Country           Population  
##   <chr>              <dbl>  
## 1 Australia        25499884  
## 2 Fiji             896445  
## 3 New Zealand     4842780  
## 4 Papua New Guinea 8947024
```

data1\_b

```
## # A tibble: 4 x 2  
##   Area            GDP  
##   <dbl>          <dbl>  
## 1 7682300  1392681.  
## 2 18270      5536.  
## 3 263820    206929.  
## 4 452860    24970.
```

# 列の結合: bind\_cols()

- 同じ行数を持つデータを単純結合

```
binded_data1 <- bind_cols(data1_a, data1_b)  
binded_data1
```

```
## # A tibble: 4 x 4  
##   Country           Population      Area       GDP  
##   <chr>              <dbl>     <dbl>     <dbl>  
## 1 Australia        25499884  7682300 1392681.  
## 2 Fiji             896445    18270    5536.  
## 3 New Zealand     4842780   263820   206929.  
## 4 Papua New Guinea 8947024   452860   24970.
```

# 列の結合

以下の場合は?

- `bind_cols()`で結合すると大混乱が生じる

データ1

```
## # A tibble: 3 x 2
##   Country     Population
##   <chr>        <dbl>
## 1 Australia    25499884
## 2 Fiji         896445
## 3 New Zealand 4842780
```

データ2

```
## # A tibble: 3 x 2
##   Country           Area
##   <chr>            <dbl>
## 1 Australia       7682300
## 2 Fiji            18270
## 3 Papua New Guinea 452860
```

# 列の結合: \*\_join() 関数群

- 結合のキーとなる変数が存在する場合使用
  - スライド前頁の場合、Country

```
# 実習用データを作成

# スライド前頁のデータ1
data1_c <- df %>%
  filter(Country %in% c("Australia", "Fiji", "New Zealand")) %>%
  select(Country, Population)

# スライド前頁のデータ2
data1_d <- df %>%
  filter(Country %in% c("Australia", "Fiji", "Papua New Guinea")) %>%
  select(Country, Area)
```

# %in%演算子

「`X %in% Y`」の場合

- `X`が`Y`に含まれているか否かをTRUE/FALSEで判定
  - 一般的に`X`は長さ1、`Y`は長さ2以上のベクトル

`Country`の値が"`Australia`"、"`Fiji`"、"`New Zealand`"のいずれかと一致する行を抽出し、`Country`と`Population`列のみ残す

```
df %>%
  filter(Country %in% c("Australia", "Fiji", "New Zealand")) %>%
  select(Country, Population)
```

```
## # A tibble: 3 x 2
##   Country     Population
##   <chr>        <dbl>
## 1 Australia    25499884
## 2 Fiji         896445
## 3 New Zealand 4842780
```

# 列の結合: 4つの考え方

- キー変数は常に一致するとは限らない
  - データ1内の国: オーストラリア、フィジー、ニュージーランド
  - データ2内の国: オーストラリア、フィジー、パプアニューギニア
- どのデータを基準に結合するか
  - `left_join()`: データ1の国で結合
  - `right_join()`: データ2の国で結合
  - `inner_join()`: データ1と2に共通する国のみ結合
  - `full_join()`: データ1と2両方の国で結合
- 結合後、存在しない値は欠損値 (NA)となる

# 列の結合: left\_join()

データ1の国が残る

- 使い方: `left_join(データ1, データ2, by = "キー変数名")`

```
left_join(data1_c, data1_d, by = "Country")
```

```
## # A tibble: 3 x 3
##   Country     Population     Area
##   <chr>        <dbl>      <dbl>
## 1 Australia    25499884 7682300
## 2 Fiji         896445    18270
## 3 New Zealand 4842780     NA
```

# 列の結合: right\_join()

データ2の国が残る

- 使い方: `right_join(データ1, データ2, by = "キー変数名")`

```
right_join(data1_c, data1_d, by = "Country")
```

```
## # A tibble: 3 x 3
##   Country           Population     Area
##   <chr>              <dbl>      <dbl>
## 1 Australia        25499884  7682300
## 2 Fiji             896445    18270
## 3 Papua New Guinea     NA    452860
```

# 列の結合: inner\_join()

データ1と2に共通する国が残る

- 使い方: `inner_join(データ1, データ2, by = "キー変数名")`

```
inner_join(data1_c, data1_d, by = "Country")
```

```
## # A tibble: 2 x 3
##   Country   Population     Area
##   <chr>       <dbl>      <dbl>
## 1 Australia  25499884 7682300
## 2 Fiji        896445    18270
```

# 列の結合: full\_join()

データ1と2両方の国が残る

- 使い方: `full_join(データ1, データ2, by = "キー変数名")`

```
full_join(data1_c, data1_d, by = "Country")
```

```
## # A tibble: 4 x 3
##   Country           Population     Area
##   <chr>              <dbl>      <dbl>
## 1 Australia        25499884  7682300
## 2 Fiji             896445    18270
## 3 New Zealand     4842780      NA
## 4 Papua New Guinea       NA    452860
```

# 列の結合: 複数のキー

複数のキーを指定することも可能: `by = c("キー1", "キー2")`

`Press_df`の中身

```
## # A tibble: 4 x 4
##   Country Year PressRank
##   <chr>    <dbl>
## 1 Japan     2019     67
## 2 Japan     2020     66
## 3 Korea     2019     41
## 4 Korea     2020     42
```

`PPP_df`の中身

```
## # A tibble: 4 x 4
##   Country Year     PPP
##   <chr>    <dbl>
## 1 Japan     2019 39763
## 2 Japan     2020 40085
## 3 Korea     2019 39060
## 4 Korea     2020 39764
```

```
left_join(Press_df, PPP_df, by = c("Country", "Year"))
```

```
## # A tibble: 4 x 4
##   Country Year PressRank     PPP
##   <chr>    <dbl>     <dbl>   <dbl>
## 1 Japan     2019     67  39763
## 2 Japan     2020     66  40085
## 3 Korea     2019     41  39060
## 4 Korea     2020     42  39764
```

# (もう一回) 表形式データの作り方

`tibble()`関数を使用

- `data.frame()`関数もOK。しかし、`tibble`の方が機能的に豊富

```
Press_df <- tibble(  
  Country = c("Japan", "Japan", "Korea", "Korea"),  
  Year    = c(2019, 2020, 2019, 2020),  
  PressRank = c(67, 66, 41, 42)  
)  
Press_df
```

```
## # A tibble: 4 x 3  
##   Country  Year PressRank  
##   <chr>    <dbl>     <dbl>  
## 1 Japan     2019      67  
## 2 Japan     2020      66  
## 3 Korea     2019      41  
## 4 Korea     2020      42
```

# tidyverse: 整然データ構造

# 整然データ構造とはなにか

## 整然データ (Tidy data)

- Tidy data: Hadley Wickhamが提唱した「データ分析に適した」データ構造
  - 3日目はggplot2を用いた可視化を解説するが、ggplot2は整然データを前提として開発されたパッケージ
  - 整然データ、簡潔データと呼ばれる
- tidyverseパッケージは雑然データを整然データへ変形するパッケージ

## 4つの原則

1. 1つの列は、1つの変数を表す
2. 1つの行は、1つの観測を表す
3. 1つのセルは、1つの値を表す
4. 1つの表は、1つの観測単位をもつ

# 原則1: 1列1変数

- 1列には1つの変数のみ
  - 3人の被験者に対し、薬を飲む前後の数学成績を測定した場合

雑然データ

Name	Control	Treatment
Hadley	90	90
Song	80	25
Yanai	100	95

被験者名

数学成績

処置有無

整然データ

Name	Treat	Math_Score
Hadley	Control	90
Hadley	Treatment	90
Song	Control	80
Song	Treatment	25
Yanai	Control	100
Yanai	Treatment	95

被験者名 処置有無 数学成績

# 原則2: 1行1観察

- 1観察 ≠ 1値
  - 観察: 観察単位ごとに測定された値の集合
  - 観察単位: 人、企業、国、時間など
- 以下の例の場合、観察単位は「人 × 時間」

雑然データ

Name	Control	Treatment
Hadley	90	90
Song	80	25
Yanai	100	95

Yanaiの投薬前の数学成績&投薬後の数学積積

整然データ

Name	Treat	Math_Score
Hadley	Control	90
Hadley	Treatment	90
Song	Control	80
Song	Treatment	25
Yanai	Control	100
Yanai	Treatment	95

Yanaiの投薬後の数学成績

# 原則3: 1セル1値

- この原則に反するケースは多くない
- 例外) 1セルに2020年8月24日という値がある場合
  - 分析の目的によっては年月日を全て異なるセルに割り当てる必要もある
  - このままで問題とならないケースも

雑然データ

Name	Treat	Math_Score
Hadley	Control, Treatment	90
Song	Control	80
Song	Treatment	25
Yanai	Control	100
Yanai	Treatment	95

# 原則4: 1表1単位

- 政府統計: 日本を代表する雑然データ
  - データの中身は良いが、構造が...
  - 表に「国」、「都道府県」、「市区町村」、「行政区」の単位が混在

都道府県名	都道府県・市区町村名	人口	平成22年	平成22年～27年の	平成22年～27年の	面積
		総数	組替人口	人口増減数	人口増減率	
(人)	(人)	(人)	(%)	(km2)		
全国	全国	127,094,745	128,057,352	-962,607	-0.8	377,970.7
北海道	北海道	5,381,733	5,506,419	-124,686	-2.3	83,424.3
北海道	札幌市	1,952,356	1,913,545	38,811	2.0	1,121.2
北海道	札幌市 中央区	237,627	220,189	17,438	7.9	46.4
北海道	札幌市 北区	285,321	278,781	6,540	2.3	63.5
北海道	札幌市 東区	261,912	255,873	6,039	2.4	56.9
北海道	札幌市 白石区	209,584	204,259	5,325	2.6	34.4
北海道	札幌市 豊平区	218,652	212,118	6,534	3.1	46.2
北海道	札幌市 南区	141,190	146,341	-5,151	-3.5	657.4
北海道	札幌市 西区	213,578	211,229	2,349	1.1	75.1
北海道	札幌市 厚別区	127,767	128,492	-725	-0.6	24.3
北海道	札幌市 手稲区	140,999	139,644	1,355	1.0	56.7
北海道	札幌市 清田区	115,726	116,619	-893	-0.8	59.8
北海道	函館市	265,979	279,127	-13,148	-4.7	677.8
北海道	小樽市	121,924	131,928	-10,004	-7.6	243.8
北海道	旭川市	339,605	347,095	-7,490	-2.2	747.6
北海道	室蘭市	88,564	94,535	-5,971	-6.3	80.8
北海道	釧路市	174,742	181,169	-6,427	-3.5	1,362.8

# 原則4: 1表1単位

## 雑然データ

Name	Treat	Math_Score
Hadley	Control	90
Hadley	Treatment	90
Song	Control	80
Song	Treatment	25
Yanai	Control	100
Yanai	Treatment	95
Total		80

単位：個人

単位：クラス

# tidyrパッケージ

雑然データから整然データへ変形をサポートするパッケージ

- `pivot_longer()`: Wide型データからLong型データへ
- `pivot_wider()`: Long型データからWide型データへ
- `separate()`: セルの分割（「年月日」から「年」、「月」、「日」へ）



# 実習用データ

- COVID19\_Asia.csv
- 日本、韓国、中国、台湾、香港の5つ間のCOVID-19新規感染者数
  - 変数名が数字で始まったり、記号が含まれている場合、`変数名` で表記

```
COVID_df <- read_csv("Data/COVID19_Asia.csv")  
COVID_df
```

```
## # A tibble: 5 x 6  
##   Country `2020/06/27` `2020/06/28` `2020/06/29` `2020/06/30` `2020/07/01`  
##   <chr>     <dbl>      <dbl>      <dbl>      <dbl>      <dbl>  
## 1 Japan      100        93        86       117        100  
## 2 Korea      62         42        43        50         50  
## 3 China      17         12        19         3         10  
## 4 Taiwan      0          0         0         0         0  
## 5 Hong Kong  1          2         4         2         1
```

# このデータの問題点

- 観察単位は? 測定した変数は?
  - 観察単位: 国 (?) × 時間
  - 変数: 新規感染者数
- 新規感染者数が5列にわたって格納されている

Country	2020/06/27	2020/06/28	2020/06/29	2020/06/30	2020/07/01
Japan	100	93	86	117	130
Korea	62	42	43	50	54
China	17	12	19	3	5
Taiwan	0	0	0	0	0
Hong Kong	1	2	4	2	28

# Wide型からLong型へ

- 整然なCOVID\_dfの構造は?
  - 5列を1列にまとめたため、縦に長くなる
  - WideからLongへ

Country	Date	Positive
Japan	2020/06/27	100
Japan	2020/06/28	93
Japan	2020/06/29	86
Japan	2020/06/30	117
Japan	2020/07/01	130
Korea	2020/06/27	62
Korea	2020/06/28	42
Korea	2020/06/29	43
Korea	2020/06/30	50
Korea	2020/07/01	54

# pivot\_longer(): Wide to Long

- colsはdplyr::select()と同じ使い方
  - c()で個別の変数名を指定することも、:やstarts\_with()を使うこともOK

```
データ %>%  
  pivot_longer(cols      = 変数が格納されている列,  
               names_to  = "元の列名が入る変数名",  
               values_to = "変数の値が入る変数名")
```

# pivot\_longer(): Wide to Long

- `cols = starts_with("2020")` もOK

```
COVID_Long <- COVID_df %>%
  pivot_longer(cols      = `2020/06/27`:`2020/07/01`,
               names_to  = "Date",
               values_to = "Positive")
```

COVID\_Long

```
## # A tibble: 25 x 3
##   Country Date     Positive
##   <chr>    <chr>     <dbl>
## 1 Japan    2020/06/27     100
## 2 Japan    2020/06/28      93
## 3 Japan    2020/06/29      86
## 4 Japan    2020/06/30     117
## 5 Japan    2020/07/01     130
## 6 Korea    2020/06/27      62
## 7 Korea    2020/06/28      42
## 8 Korea    2020/06/29      43
## 9 Korea    2020/06/30      50
## 10 Korea   2020/07/01      54
## # ... with 15 more rows
```

# pivot\_wider(): Long to Wide

- Long型をWideへ戻す関数
  - 人間にとてはLong型よりWide型の方が読みやすいケースも多い
  - 1列に2つの変数が入っている場合もある

```
COVID_Long %>%
  pivot_wider(names_from = "Date",
              values_from = "Positive")
```

```
## # A tibble: 5 x 6
##   Country   `2020/06/27` `2020/06/28` `2020/06/29` `2020/06/30` `2020/07/01`
##   <chr>       <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
## 1 Japan        100        93        86       117        120
## 2 Korea         62         42        43        50         50
## 3 China         17         12        19         3         3
## 4 Taiwan        0          0         0         0         0
## 5 Hong Kong     1          2         4         2         1
```

# separate():列の分割

COVID\_LongのDate列をYear、Month、Dayに分けたい

- Date列を/を基準に分割する

```
データ %>%
  separate(col = "分割する列名",
          into = c("分割後の列名1", "分割後の列名2", ...),
          sep = "分割する基準")
```

# separate():列の分割

```
COVID_Long %>%  
  separate(col = "Date",  
          into = c("Year", "Month", "Day"),  
          sep = "/")
```

```
## # A tibble: 25 x 5  
##   Country Year Month Day  Positive  
##   <chr>    <chr> <chr> <chr>    <dbl>  
## 1 Japan    2020  06    27     100  
## 2 Japan    2020  06    28      93  
## 3 Japan    2020  06    29      86  
## 4 Japan    2020  06    30     117  
## 5 Japan    2020  07    01     130  
## 6 Korea    2020  06    27      62  
## 7 Korea    2020  06    28      42  
## 8 Korea    2020  06    29      43  
## 9 Korea    2020  06    30      50  
## 10 Korea   2020  07    01      54  
## # ... with 15 more rows
```

# tidyrについて

ここで紹介したtidyrの機能はごく一部

- より詳しく知りたい場合、[SONG・矢内の資料第15章](#)を参照
- 今どきのRにおいて整然データの概念は重要
  - 多くのパッケージが整然データを前提に提供されている