ミクロ政治データ分析実習

第10回 データハンドリング（3）

そん じぇひょん
宋 財 泫
関西大学総合情報学部
2021/6/17 (updated: 2021-06-11)

{dplyr}: データの結合

データの結合: 行

`bind_rows()`を利用: 変数名が一致する必要がある

結合前

```
Data1
```

```
##   ID  Name Score
## 1  1 Aさん    77
## 2  2 Bさん    89
## 3  3 Cさん    41
```

```
Data2
```

```
##   ID  Name Score
## 1  4 Xさん    98
## 2  5 Yさん    78
```

結合

```
bind_rows(Data1, Data2)
```

```
##   ID  Name Score
## 1  1 Aさん    77
## 2  2 Bさん    89
## 3  3 Cさん    41
## 4  4 Xさん    98
## 5  5 Yさん    78
```

データの結合: 行

結合前のデータ識別変数の追加

- 結合するデータを`list()`でまとめ、`.id`引数を追加する
- `list()`の内部では`"識別変数の値" = 結合するデータ`と定義

例) 結合後、`Class`という列を追加し、元々`Data1`だった行は`"1組"`、`Data2`だった行には`"2組"`を格納する。

```
bind_rows(list("1組" = Data1, "2組" = Data2),
          .id = "Class")
```

```
##   Class ID  Name Score
## 1   1組  1 Aさん    77
## 2   1組  2 Bさん    89
## 3   1組  3 Cさん    41
## 4   2組  4 Xさん    98
## 5   2組  5 Yさん    78
```

データの結合: 列

`*_join()`: 結合に使う識別用の変数（**キー変数**）が必要（以下では`City`）

結合前

```
Data1
```

```
##    City Pop Area
## 1 Tokyo 927 2194
## 2 Osaka 148  828
## 3 Kochi  76 7104
```

結合前

```
Data2
```

```
##    City      Food
## 1 Kochi   Katsuo
## 2 Osaka Takoyaki
## 3 Tokyo    Ramen
```

結合

```
left_join(Data1, Data2, by = "City")
```

```
##    City Pop Area     Food
## 1 Tokyo 927 2194    Ramen
## 2 Osaka 148  828 Takoyaki
## 3 Kochi  76 7104   Katsuo
```

列結合に使う関数

識別子は両データに含まれているが、一致しないケースがある。

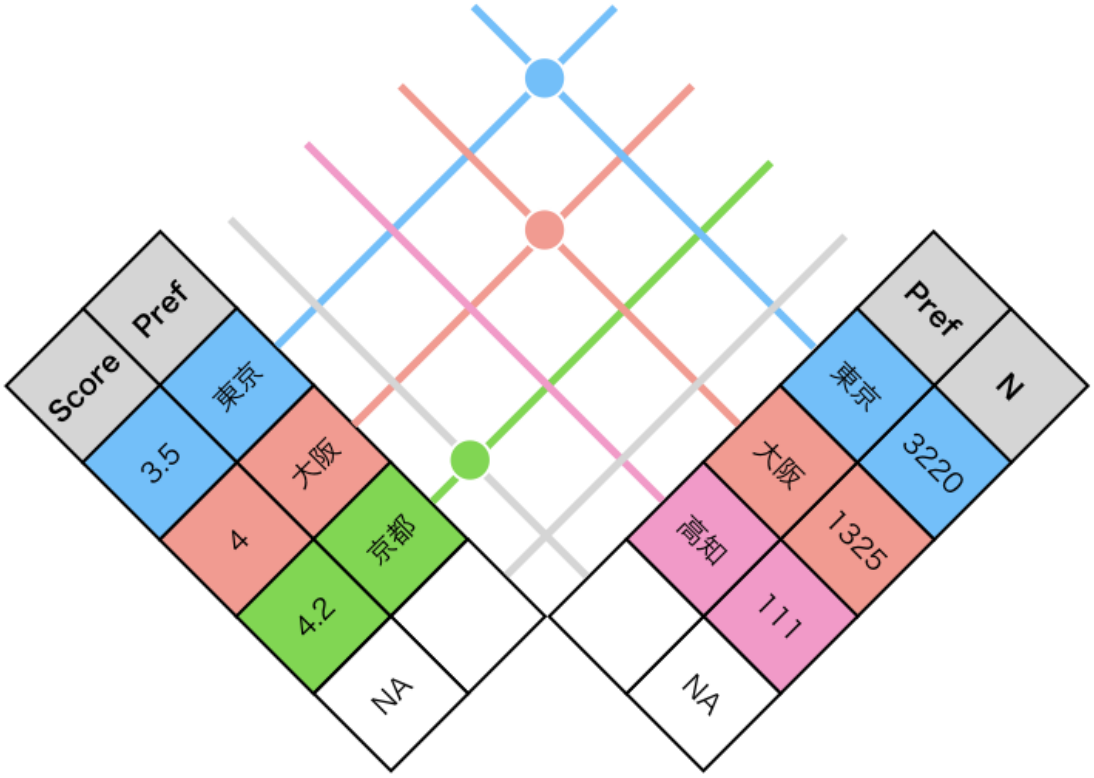
- どのデータの識別子を優先するか

1. `left_join()`
   - 左側のデータの識別子を優先する
   - 空欄は欠損値として埋められる
2. `right_join()`
   - 右側のデータの識別子を優先する
   - 空欄は欠損値として埋められる
3. `inner_join()`
   - 両データの識別子に共通する行のみを残して結合
4. `full_join()`
   - 両データの識別子に存在する行すべて結合
   - 空欄は欠損値として埋められる

left_join() の仕組み

`left_join(データ1, データ2, by = "識別用変数名")`

- データ1を温存する
- 欠損しているセルは欠損値（`NA`）で埋められる



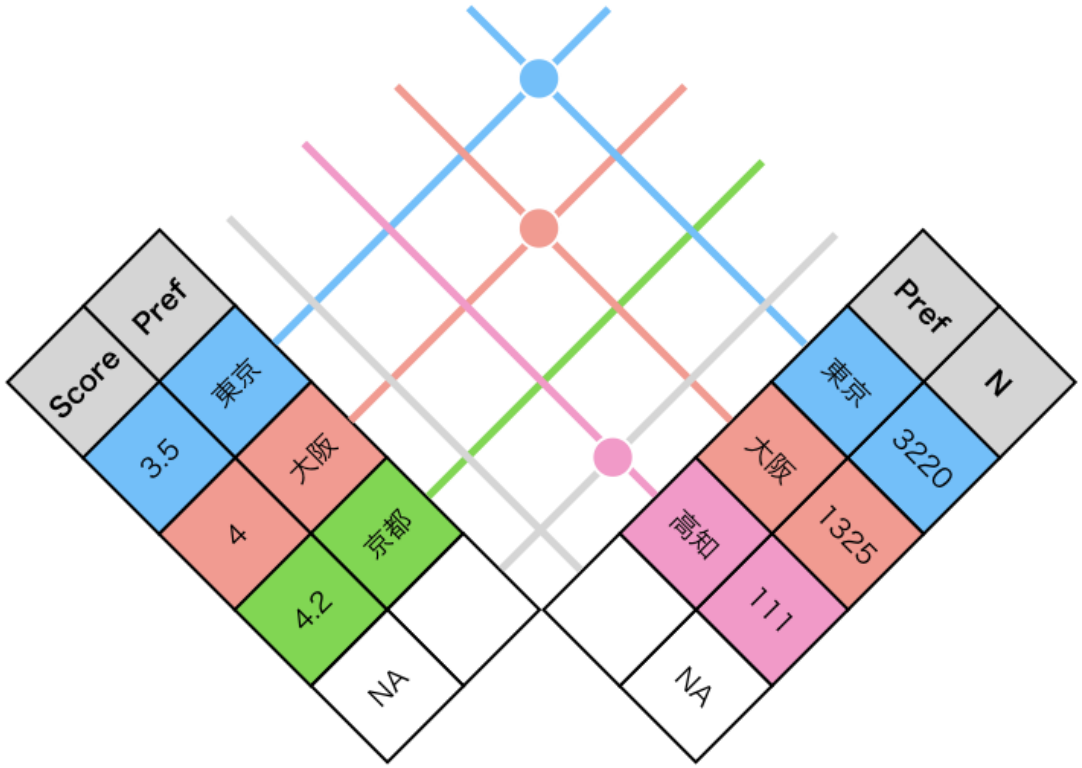
left_join()

Pref	Score	N
東京	3.5	3220
大阪	4	1325
京都	4.2	NA

right_join()の仕組み

`right_join(データ1, データ2, by = "識別用変数名")`

- データ2を温存する
- 欠損しているセルは欠損値（`NA`）で埋められる



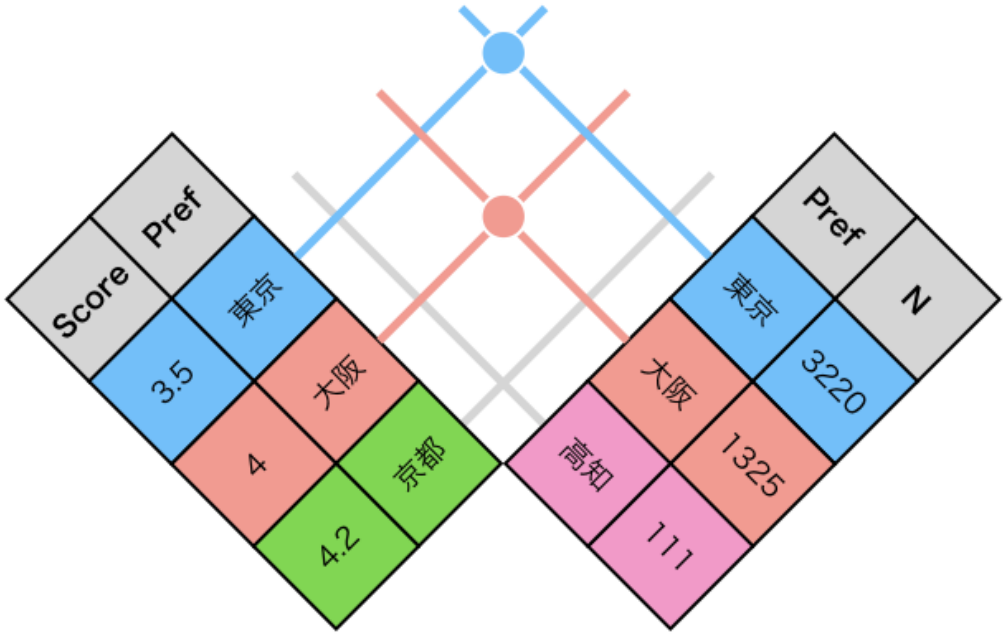
right_join()

Pref	Score	N
東京	3.5	3220
大阪	4	1325
高知	NA	111

inner_join() の仕組み

```
inner_join(データ1, データ2, by = "識別用変数名")
```

- データ1とデータ2で識別子が共通する行のみ結合



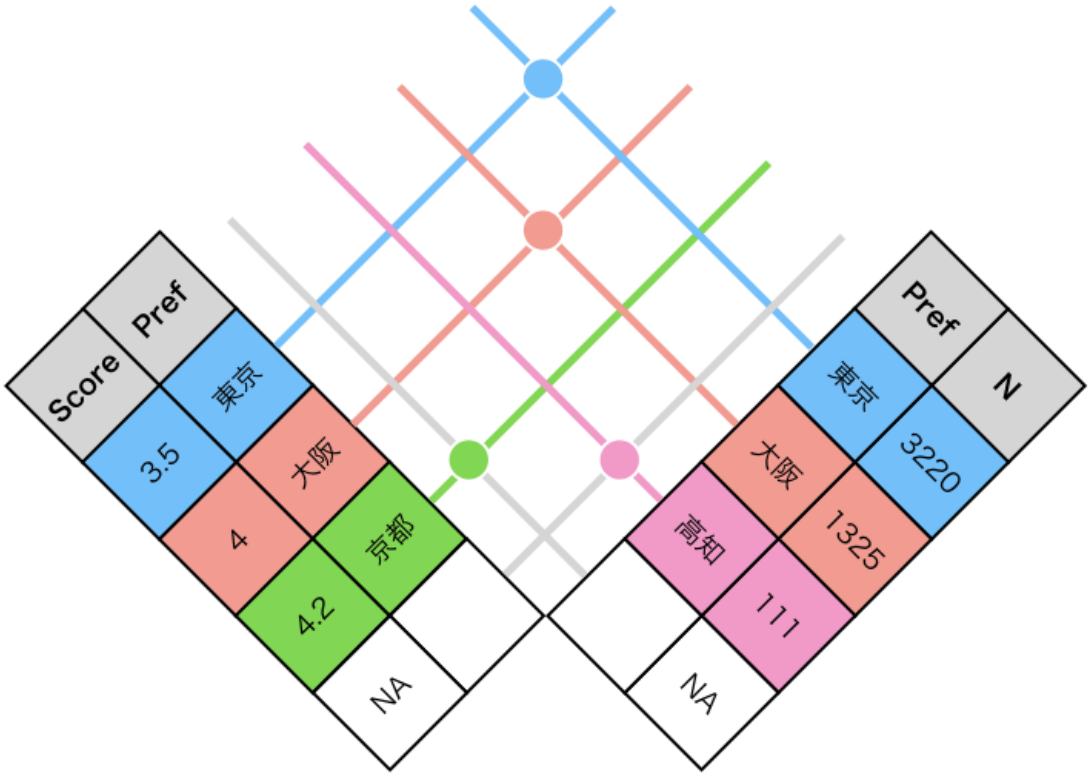
inner_join()

Pref	Score	N
東京	3.5	3220
大阪	4	1325

full_join()の仕組み

`full_join(データ1, データ2, by = "識別用変数名")`

- データ1とデータ2をすべて温存
- 欠損しているセルは欠損値（`NA`）で埋められる



full_join()

Pref	Score	N
東京	3.5	3220
大阪	4	1325
京都	4.2	NA
高知	NA	111

比較 (1)

```
df1 <- tibble(Pref  = c("東京", "大阪", "京都"),
              Score = c(3.5, 4, 4.2))
df2 <- tibble(Pref  = c("東京", "大阪", "高知"),
              N     = c(3220, 1325, 111))
```

```
df1
```

```
## # A tibble: 3 x 2
##   Pref  Score
##   <chr> <dbl>
## 1 東京    3.5
## 2 大阪    4
## 3 京都    4.2
```

```
df2
```

```
## # A tibble: 3 x 2
##   Pref      N
##   <chr> <dbl>
## 1 東京   3220
## 2 大阪   1325
## 3 高知    111
```

比較 (2)

`by = "識別用の変数名"` は複数用いることも可能（例: 都道府県名&年度で結合）

- `by = c("識別用の変数名1", "識別用の変数名2")`

```
left_join(df1, df2, by = "Pref")
```

```
## # A tibble: 3 x 3
##   Pref  Score     N
##   <chr> <dbl> <dbl>
## 1 東京    3.5  3220
## 2 大阪    4    1325
## 3 京都    4.2    NA
```

```
right_join(df1, df2, by = "Pref")
```

```
## # A tibble: 3 x 3
##   Pref  Score     N
##   <chr> <dbl> <dbl>
## 1 東京    3.5  3220
## 2 大阪    4    1325
## 3 高知   NA     111
```

```
inner_join(df1, df2, by = "Pref")
```

```
## # A tibble: 2 x 3
##   Pref  Score     N
##   <chr> <dbl> <dbl>
## 1 東京    3.5  3220
## 2 大阪    4    1325
```

```
full_join(df1, df2, by = "Pref")
```

```
## # A tibble: 4 x 3
##   Pref  Score     N
##   <chr> <dbl> <dbl>
## 1 東京    3.5  3220
## 2 大阪    4    1325
## 3 京都    4.2    NA
## 4 高知   NA     111
```

{tidyr}: 整然データ構造

整然データ構造とは

Tidy data: Hadley Wickhamが提唱した**データ分析に適したデータ構造**

- **整然データ**、簡潔データと呼ばれる
- パソコンにとって読みやすいデータ $\neq$ 人間にとって読みやすいデータ
- {tidyr}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
- 次回紹介する{ggplot2}は整然データを前提として開発されたパッケージ

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
  - 薬を飲む前: Control / 薬を飲んだ後: Treatment

雑然データ

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

処置有無

被験者名

数学成績

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 $\neq$ 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 $\times$ 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

地域		人口	平成22年	平成22年～27年の	平成22年～27年の	
		総数	組替人口	人口増減数	人口増減率	面積
都道府県名	都道府県・市区町村名	（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	1,362.9

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
  - 原則1・2に反するデータを整然データへ変換 (最も頻繁に使われる)
- `pivot_wider()`: Long型データからWide型データへ
  - 人間には雑然データの方が読みやすい場合がある（原則1の例）
- `separate()`: セルの分割（「年月日」から「年」、「月」、「日」へ）
  - 原則3に反するデータを整然データへ変換
- 原則4に反するデータは単位がずれている行を`filter()`などで除外

実習用データ

`Micro10.csv`: 日本、韓国、モンゴル、台湾の5日間COVID-19新規感染者数

- サポートページからダウンロード
- データ出典:Johns Hopkins University Center for Systems Science and Engineering

```
# プロジェクトフォルダー（作業ディレクトリ）内のDataフォルダー内の
# Micro10.csvを読み込み、COVID_dfという名で格納
COVID_df <- read_csv("Data/Micro10.csv")
COVID_df
```

```
## # A tibble: 4 x 7
##   Country  Population `2021/05/24` `2021/05/25` `2021/05/26` `2021/05/27`
##   <chr>         <dbl>        <dbl>        <dbl>        <dbl>        <dbl>
## 1 Japan     126476461         2743         3918         4485         4163
## 2 Korea      51269185          516          699          629          587
## 3 Mongolia    3278290          634          680          653          785
## 4 Taiwan     23816775          595          539          635          670
## # … with 1 more variable: 2021/05/28 <dbl>
```

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: 地域 × 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	Population	2021/05/24	2021/05/25	2021/05/26	2021/05/27	2021/05/28
Japan	126476461	2743	3918	4485	4163	3701
Korea	51269185	516	699	629	587	533
Mongolia	3278290	634	680	653	785	769
Taiwan	23816775	595	539	635	670	554

Wide型からLong型へ

- 整然な`COVID_df`の構造は?
  - 5列を1列にまとめるため、縦に長くなる
  - WideからLongへ

Country	Population	Date	New_Cases
Japan	126476461	2021/05/24	2743
Japan	126476461	2021/05/25	3918
Japan	126476461	2021/05/26	4485
Japan	126476461	2021/05/27	4163
Japan	126476461	2021/05/28	3701
Korea	51269185	2021/05/24	516
Korea	51269185	2021/05/25	699
Korea	51269185	2021/05/26	629
Korea	51269185	2021/05/27	587
Korea	51269185	2021/05/28	533

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
  - **注意:** 変数名が数字で始まったり、記号が含まれている場合、変数名を`` ` ``か`"`で囲む
  - 列名が日付の場合、数字で始まったり、記号（`/`や`-`など）が含まれるケースが多い

```
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からLongへ

- `cols = starts_with("2020")`もOK

```
COVID_Long <- COVID_df %>%
  pivot_longer(cols      = `2021/05/24`:`2021/05/28`,
               names_to  = "Date",
               values_to = "New_Cases")
COVID_Long
```

```
## # A tibble: 20 x 4
##    Country  Population Date       New_Cases
##    <chr>         <dbl> <chr>          <dbl>
##  1 Japan     126476461 2021/05/24      2743
##  2 Japan     126476461 2021/05/25      3918
##  3 Japan     126476461 2021/05/26      4485
##  4 Japan     126476461 2021/05/27      4163
##  5 Japan     126476461 2021/05/28      3701
##  6 Korea      51269185 2021/05/24       516
##  7 Korea      51269185 2021/05/25       699
##  8 Korea      51269185 2021/05/26       629
##  9 Korea      51269185 2021/05/27       587
## 10 Korea      51269185 2021/05/28       533
```

pivot_wider()：LongからWideへ

- Long型をWide型へ戻す関数
  - 人間にとっては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 = "New_Cases")
```

```
## # A tibble: 4 x 7
##   Country  Population `2021/05/24` `2021/05/25` `2021/05/26` `2021/05/27`
##   <chr>         <dbl>        <dbl>        <dbl>        <dbl>        <dbl>
## 1 Japan     126476461         2743         3918         4485         4163
## 2 Korea      51269185          516          699          629          587
## 3 Mongolia    3278290          634          680          653          785
## 4 Taiwan     23816775          595          539          635          670
## # … with 1 more variable: 2021/05/28 <dbl>
```

separate()：列の分割

`COVID_Long`の`Date`列を`Year`、`Month`、`Day`に分けたい

- 例) `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: 20 x 6
##    Country  Population Year  Month Day   New_Cases
##    <chr>         <dbl> <chr> <chr> <chr>     <dbl>
##  1 Japan     126476461 2021  05    24         2743
##  2 Japan     126476461 2021  05    25         3918
##  3 Japan     126476461 2021  05    26         4485
##  4 Japan     126476461 2021  05    27         4163
##  5 Japan     126476461 2021  05    28         3701
##  6 Korea      51269185 2021  05    24          516
##  7 Korea      51269185 2021  05    25          699
##  8 Korea      51269185 2021  05    26          629
##  9 Korea      51269185 2021  05    27          587
## 10 Korea      51269185 2021  05    28          533
## 11 Mongolia    3278290 2021  05    24          634
## 12 Mongolia    3278290 2021  05    25          680
## 13 Mongolia    3278290 2021  05    26          653
```

列の分割: 特定の記号がない場合（1）

例) `City_Data` の `City` 列が「都道府県名+市区町村」

- 「最初の3文字」と「残り」で分割することは出来ない（神奈川、和歌山、鹿児島）
- 任意の2文字の後に「都」、「道」、「府」、「県」が付くか、任意の3文字の後に「県」が付く箇所を見つけて分割
  - かなり複雑

```
## # A tibble: 4 x 2
##   City                 Pop
##   <chr>              <dbl>
## 1 北海道音威子府村     693
## 2 大阪府高槻市      347424
## 3 広島県府中市       36471
## 4 鹿児島県指宿市     38207
```

列の分割: 特定の記号がない場合（2）

正則表現（regular expression）の知識が必要

- テキスト分析に興味があるなら必須（前期・後期含めて、本講義では取り上げない）

```
City_Data %>%
    # 任意の2文字の後に「都道府県」のいずれかが来るか、
    # 任意の3文字の後に「県」が来たら、そこまでをブロック1、残りをブロック2とする
    # Cityの値を「ブロック1-ブロック2」に置換する
    mutate(City = str_replace(City, "^(.{2}[都道府県]|.{3}県)(.+)",
                              "\\1-\\2")) %>%
    # 「-」を基準に列を分割
    separate(col  = "City", into = c("Pref", "City"), sep  = "-")
```

```
## # A tibble: 4 x 3
##   Pref     City          Pop
##   <chr>    <chr>       <dbl>
## 1 北海道   音威子府村    693
## 2 大阪府   高槻市      347424
## 3 広島県   府中市       36471
## 4 鹿児島県 指宿市       38207
```

{tidyr}と{dplyr}の組み合わせ

{tidyr}と{dplyr}を組み合わせることも可能

- 例) 100万人当たりの新規感染者数を計算し、国ごとに平均値を計算

```
COVID_df %>%
  pivot_longer(cols      = `2021/05/24`:`2021/05/28`,
               names_to  = "Date",
               values_to = "New_Cases") %>%
   mutate(New_Case_per_1M = New_Cases / Population * 1000000) %>%
   group_by(Country) %>%
   summarise(New_Case_per_1M = mean(New_Case_per_1M))
```

```
## # A tibble: 4 x 2
##   Country  New_Case_per_1M
##   <chr>              <dbl>
## 1 Japan               30.1
## 2 Korea               11.6
## 3 Mongolia           215.
## 4 Taiwan              25.1
```

まとめ

今回の内容

よく分からない箇所は教科書を読み返す or 宋&TAに質問 (できれば、**LMSの質問コーナー**で)

- データの結合: 教科書第14.5章
- 整然データ構造: : 教科書第16章

データハンドリングに慣れるためには

- とりあえず、たくさんのデータをいじってみる
- たくさんのエラーメッセージに出会うこと
- パイプ（`%>%`）を使いすぎないように
  - 中級以上になると、自分が書いたコードの結果が予想できるため、たくさんのパイプを使っても問題は大きくない
  - 一方、初心者の場合、パイプを使いすぎず、2〜3回ごとに別途のオブジェクトとして保存したり、結果を確認していくこと
  - パイプが多すぎるとどこがエラーの原因かの特定が困難に（慣れたらすぐに見つかるが）

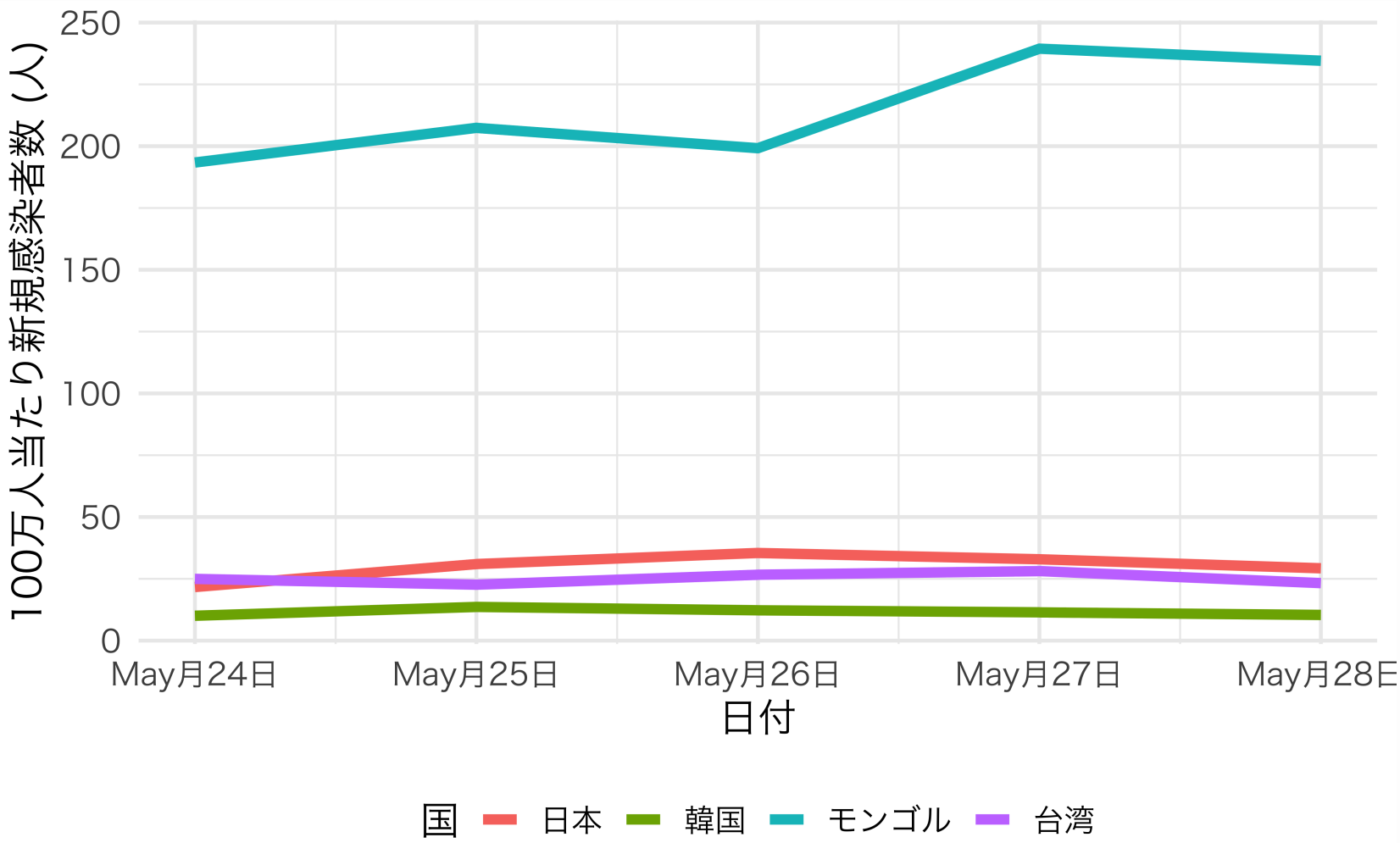
長過ぎるコードブロックの例

慣れたらこれくらいは長い方ではないが...

```
COVID_df %>%
  pivot_longer(cols      = `2021/05/24`:`2021/05/28`,
               names_to  = "Date",
               values_to = "New_Cases") %>%
   mutate(New_Case_per_1M = New_Cases / Population * 1000000,
          Country         = case_when(Country == "Japan"    ~ "日本",
                                      Country == "Korea"    ~ "韓国",
                                      Country == "Mongolia" ~ "モンゴル",
                                      TRUE                  ~ "台湾"),
          Country         = factor(Country, levels = c("日本", "韓国",
                                                       "モンゴル", "台湾")),
          Date            = as.Date(Date)) %>%
   ggplot(aes(x = Date, y = New_Case_per_1M,
              group = Country, color = Country)) +
   geom_line(size = 1.5) +
   labs(x = "日付", y = "100万人当たり新規感染者数（人）", color = "国") +
   scale_x_date(date_labels = "%b月%d日") +
   theme_minimal(base_size = 16) +
   theme(legend.position = "bottom")
```

来週以降の内容

次回からは{ggplot2}を利用したデータの可視化方法を紹介



課題

1. 今回講義用のプロジェクトを作成する。
2. LMSから**2つ**のデータ（`.csv`）、問題ファイル（`.Rmd`）とサンプルファイル（`.html`）をダウンロードし、プロジェクトのフォルダーに保存する。
    - **ファイル名は変更しない**こと
3. プロジェクトを開き、`.Rmd`ファイルを開く
4. サンプルファイルと同じ結果が得られるようにR Markdown文書を作成する。
5. 随時Knitし、結果を確認する。
    - Knitできないファイルは評価できない
6. `.Rmd`ファイルを関大LMSに提出する。
7. **期限は2021年6月19日（土）の23時59分とする。**
    - 時間に余裕を持って取り組むこと。期限直前に取り組み始めてPCトラブルがあっても期限延長はない
8. 答案は次回の講義までに公開する。