ミクロ政治データ分析実習

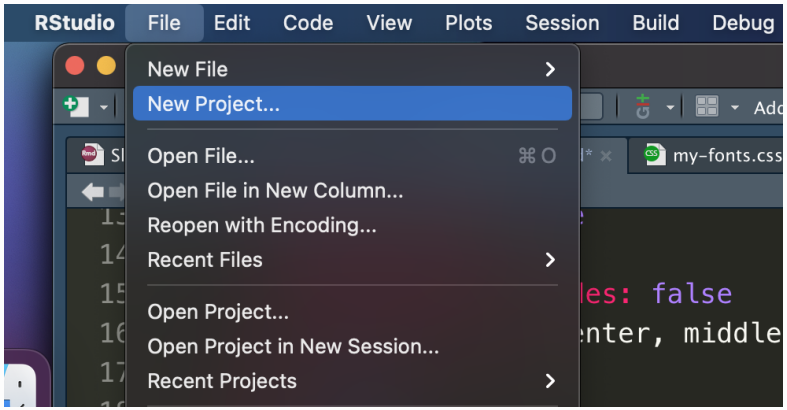
第3回 Rの基本的な操作

そん じぇひょん
宋 財 泫
関西大学総合情報学部
2021/4/22 (updated: 2021-04-08)

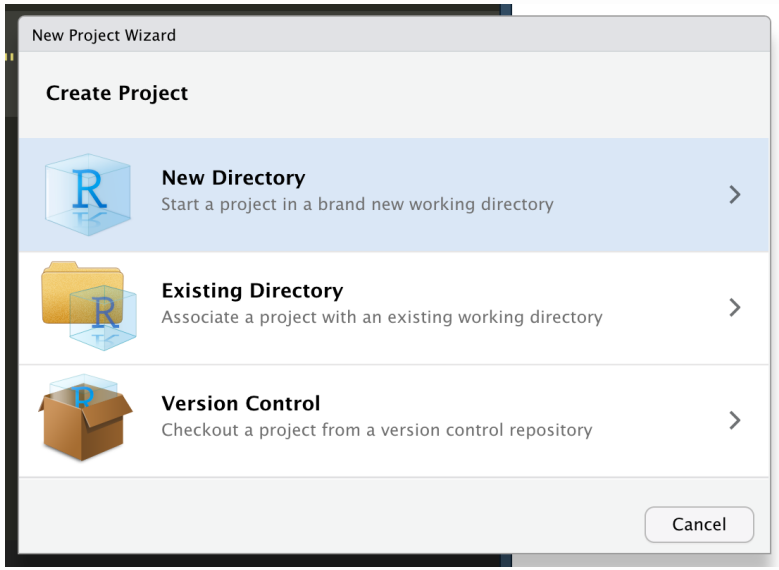
プロジェクト生成

プロジェクトの作成（１）

Step 1: File > New Project

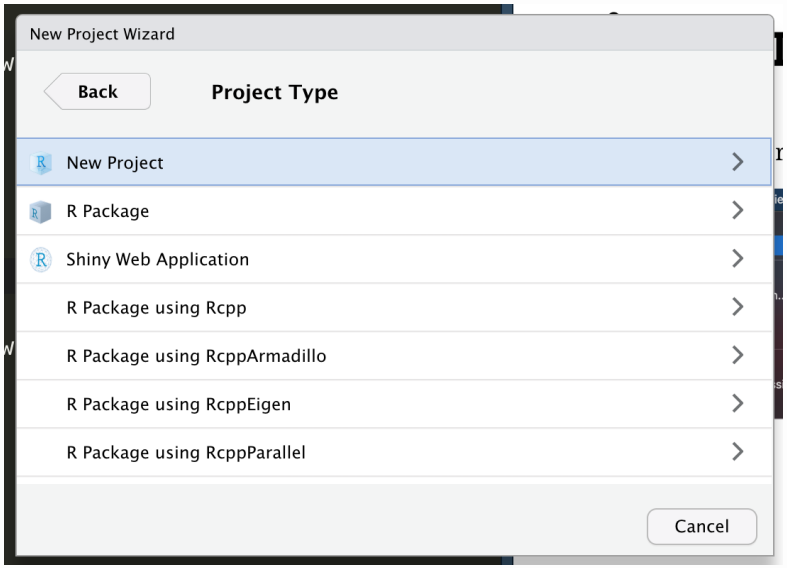


Step 2: New Directoryを選択

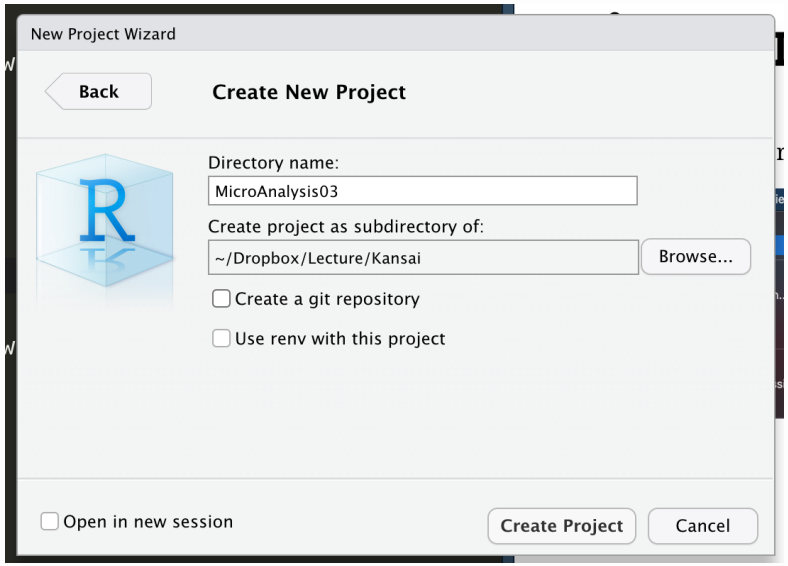


プロジェクトの作成（2）

Step 3: New Projectを選択



Step 4: プロジェクト名、パス指定



- プロジェクト名は必ず英数字のみ
  - `_`はOK
- Browse..をクリックし、プロジェクトフォルダを置く場所を指定
  - 指定後、左側のパスが英数字のみで構成されているか確認

プロジェクトを開く: ローカル版

- 一旦、RStudioを閉じる。
- Step 4で指定したフォルダ内にプロジェクト名のフォルダがあるかを確認
- プロジェクト名フォルダ内に`プロジェクト名.Rproj`があるはず
  - 拡張子（`.Rproj`）が表示されない場合、  アイコンで識別可
  - 今後のために拡張子は常に表示されるように設定しておこう
  - macOSの場合: Googleで「macOS 拡張子 表示」
  - Windowsの場合: Googleで「windows 拡張子 表示」
- `プロジェクト名.Rproj`をダブルクリックすれば、RStudioが起動される。
  - RStudioを先に起動し、File > Open Project...も可
- Consoleペインに以下のように入力

```
# 現在の作業フォルダの絶対パスを出力
getwd()
```

```
## [1] "/Users/jaehyunsong/Dropbox/Lecture/Kansai/2021/02_MicroAnalysis/2021"
```

プロジェクトを開く: クラウド版

- 一旦、RStudioを閉じる。
  - 右上の  をクリックし、タブを閉じる
- Jupyter hub画面からStep 4で指定したプロジェクト名のフォルダがあるかを確認
- RStudioの起動（New > RStudio）
- File > Open Project...
- プロジェクトのフォルダ内の`.Rproj`ファイルを選択し、Open
- Consoleペインに以下のように入力

```
# プロジェクト名をMicroAnalysis03にした場合
getwd()
```

```
## [1] "/home/jovyan/MicroAnalysis03"
```

電卓としてのRと演算子

R Scriptの作成

- Step 1: File > New File > R Script
  - Cmd + Shift + NもOK（Windowsの場合Cmdの代わりにCtrl）
- Step 2: 左下のHistoryペインを隠す（ペイン右上の  をクリック）
- Step 3: Sourceペインに以下のように入力する

```
1 + 2 + 3
```

- Step 4: Step 3で入力した行に移動し、Cmd + Return (Ctrl + Enter)
  - Cmd + Returnを押すと、カーソルが位置する行の内容がConsoleへ送られ、実行される。
  - Consoleペインに以下のように出力されればOK

```
> 1 + 2 + 3
[1] 6
```

- Rを終了する際に、作成したスクリプトを保存しておく
  - File > Save

電卓としてのR

```
1 + 5  # 足し算
```

```
## [1] 6
```

```
3 - 10 # 引き算
```

```
## [1] -7
```

```
19 * 2 # 掛け算
```

```
## [1] 38
```

```
13 / 7 # 割り算
```

```
## [1] 1.857143
```

```
5^3    # 5の3乗
```

```
## [1] 125
```

Rの基本的な演算子

- 結果として何らかの**数字**が返される
- 累乗（`^`）を除き、演算子の前後にはスペースを入れるのがRの流儀
  - 必須ではないが、コードが読みやすくなる

演算子	意味	例	結果
`+`	和	`2 + 5`	7
`-`	差	`2 - 8`	-6
`*`	積	`7 * 3`	21
`/`	商	`16 / 5`	3.2
`^`、`**`	累乗（べき乗）	`2^3` または `2 ** 3`	8
`%%`	剰余 (モジュロ)	`18 %% 7`	4
`%/%`	整数商	`18 %/% 7`	2

論理演算子

- 真（`TRUE`）か偽（`FALSE`）の値を返す演算子

```
3 > 2 # 3は2より大きいか
```

```
## [1] TRUE
```

```
5 <= 10 # 5は10以下か
```

```
## [1] TRUE
```

```
2 + 3 == 1 # 2 + 3は1に等しいか
```

```
## [1] FALSE
```

```
2 + 2 != 2 * 2 # 2 + 3は2 * 2と等しくないか
```

```
## [1] FALSE
```

論理演算子まとめ

- 「等しい」は`=`でなく、`==`であることに注意すること

演算子	意味	例	結果
`x < y`	`x`は`y`より小さい	`3 < 1`	`FALSE`
`x <= y`	`x`は`y`と等しいか、小さい	`2 <= 2`	`TRUE`
`x > y`	`x`は`y`より大きい	`6 > 5`	`TRUE`
`x >= y`	`x`は`y`と等しいか、大きい	`4 >= 5`	`FALSE`
`x == y`	`x`と`y`は等しい	`(2 + 3) == (4 + 1)`	`TRUE`
`x != y`	`x`と`y`は等しくない	`((2 * 3) + 1) != (2 * (3 + 1))`	`TRUE`

論理演算子: ANDとOR

`&`: AND演算子

- `&`を挟む左右の**両側**が`TRUE`の場合のみ`TRUE`を返す

```
(2 + 3 == 5) & (1 * 2 == 3) # TRUE and FALSEだから
```

```
## [1] FALSE
```

`|`: OR演算子

- `|`を挟む左右の**片側、あるいは両側**が`TRUE`の場合のみ`TRUE`を返す

```
(2 + 3 == 5) | (1 * 2 == 3) # TRUE or FALSEだから
```

```
## [1] TRUE
```

基礎演算: 練習

03:00

1. $\frac{1234 \times 4321}{3}$
2. $\frac{1234 \times 4321}{3}$の余り
   - $(1234 \times 4321) \bmod 3$とも表記される
3. 2×3 と $2 + 3$ は等しいか
4. $1009 \bmod 3$ は0か
5. 5^5 は100以上、かつ1000未満であるか

基礎演算: 答え

```
1234 * 4321 / 3
```

```
## [1] 1777371
```

```
1234 * 4321 %% 3
```

```
## [1] 1234
```

```
2 * 3 == 2 + 3
```

```
## [1] FALSE
```

```
1009 %% 3 == 0
```

```
## [1] FALSE
```

```
(5^5 >= 100) & (5^5 < 1000)
```

```
## [1] FALSE
```

ベクトルの操作

考えてみよう

- 123454321 × 2を計算してみよう
- 123454321 × 3を計算してみよう
- 123454321 × 4を計算してみよう
- ...

効率的な方法は?

- `123454321` をコピーし貼り付けながら計算を繰り返す
- `123454321` に `x` という名前を付けて、`x * 1` のように表記する

Rにおけるベクトル

Rにおけるデータの最小単位

- 同じデータ型（数値、文字列など）
  - 数値と文字列が混在するベクトルは作成不可
- 長さは1以上
  - `1` や `"Cat"` は長さ1のベクトル
- `c()` 関数で作成
  - `()` の中にベクトルの要素を入力する

ベクトルの格納

- `<-` 演算子で格納
  - `ベクトル名` という名前のオブジェクト（object）が生成される
- 格納しない場合、作業環境に保存されず、出力のみ

```
ベクトル名 <- c(要素1, 要素2, ...)
```

ベクトルの作成と格納

例1

- 要素が1, 2, 3, 4, 5の長さ5の数値型ベクトル
- ベクトルを`my_vector1`という名前で作業環境に格納

```
my_vector1 <- c(1, 2, 3, 4, 5)
```

例2

- 要素が"Cat", "Lion", "Tiger"の長さ3の文字型ベクトルを`my_vector2`という名で格納
  - 文字は必ず`"`か`'`で囲む

```
my_vector2 <- c("Cat", "Lion", "Tiger")
```

オブジェクトの表示

現在の作業環境におけるオブジェクトリスト

1. Environmentペインで確認
2. Consoleペインで`ls()`を入力

```
ls()
```

```
## [1] "my_vector1" "my_vector2"
```

オブジェクトの出力

- オブジェクト名のみ入力
  - `print(オブジェクト名)`もOK

```
my_vector1
```

```
## [1] 1 2 3 4 5
```

- Rを再起動すると作業環境が初期化されるため、もう一回作成する必要がある。
  - コードを記録し、残すことが重要

比較

```
123454321 * 2
```

```
## [1] 246908642
```

```
123454321 * 3
```

```
## [1] 370362963
```

```
123454321 * 4
```

```
## [1] 493817284
```

```
123454321 * 5
```

```
## [1] 617271605
```

```
x <- 123454321
x * 2
```

```
## [1] 246908642
```

```
x * 3
```

```
## [1] 370362963
```

```
x * 4
```

```
## [1] 493817284
```

```
x * 5
```

```
## [1] 617271605
```

等差数列ベクトルの作り方

公差1の等差数列

- `:`演算子を利用

```
c(1, 2, 3, 4, 5, 6, 7)
```

```
## [1] 1 2 3 4 5 6 7
```

```
1:7
```

```
## [1] 1 2 3 4 5 6 7
```

その他の等差数列

- `seq()`関数の使用

```
c(1, 3, 5, 7, 9)
```

```
## [1] 1 3 5 7 9
```

```
# 1から9まで、公差2の等差数列
 seq(1, 9, by = 2)
```

```
## [1] 1 3 5 7 9
```

内蔵ベクトル

Rがデフォルトで提供しているベクトル

- `ls()`でも表示されないが、Rに内蔵されている

ローマ字（大文字）

```
LETTERS
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
## [18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

ローマ字（小文字）

```
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

ベクトル演算（1）

長さ2以上のベクトルと長さ1のベクトル同士

- 長い方のベクトルの各要素と長さ1ベクトルの値間の演算

```
# my_vector1の各要素に10をかける
my_vector1 * 10
```

```
## [1] 10 20 30 40 50
```

```
# my_vector1の各要素を2で割った余りが0か否か
my_vector1 %% 2 == 0
```

```
## [1] FALSE  TRUE FALSE  TRUE FALSE
```

ベクトル演算（2）

長さ2以上のベクトル同士

- 同じ位置の要素同士で演算

```
my_vector3 <- c(1, 3, 5, 7, 9)
my_vector1 + my_vector3
```

```
## [1]  2  5  8 11 14
```

ベクトルの長さが一致しない場合

- 短い方のベクトルがリサイクルされる（=ベクトル・リサイクル）
- 警告（warning）が表示される場合もある

```
my_vector4 <- c(0, 1)
my_vector1 * my_vector4
```

```
## [1] 0 2 0 4 0
```

ベクトル抽出

1. 長さ5のベクトルから`i`番目の要素を抽出: `ベクトル名[i]`
2. 長さ5のベクトルから`i`, `j`, `k`番目の要素を抽出: `ベクトル名[c(i, j, k)]`
3. 長さ5のベクトルから`i`番目から`j`番目の要素を抽出
   - `ベクトル名[i:j]`
   - `ベクトル名[seq(i, j, by = 1)]`
4. 長さ5のベクトルから偶数番目の要素なら
   - `ベクトル名[seq(1, 5, by = 2)]`
5. 長さ5のベクトル内の個々の要素の出力有無を`TRUE`と`FALSE`で指定
   - 2番目と5番目の要素のみ出力
   - `ベクトル名[c(FALSE, TRUE, FALSE, FALSE, TRUE)]`
   - `TRUE`は`T`、`FALSE`は`F`で略せるが、非推奨
   - `[]`の中に論理式を入れることも可能

ベクトル抽出: 要素の位置指定

```
my_vector1[2] # 2番目の要素
```

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

```
my_vector1[c(2, 4)] # 2、4番目の要素
```

```
## [1] 2 4
```

```
my_vector1[1:3] # 1番目から3番目の要素
```

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

```
my_vector1[seq(1, 5, by = 2)] # 1、3、5番目の要素
```

```
## [1] 1 3 5
```

ベクトル抽出: 論理式の利用

```
my_vector1[c(TRUE, FALSE, FALSE, FALSE, TRUE)] # 1、5番目の要素
```

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

```
my_vector1[my_vector1 %% 2 == 1] # 2で割って余りが1の要素
```

```
## [1] 1 3 5
```

参考

```
my_vector1 %% 2 == 1
```

```
## [1]  TRUE FALSE  TRUE FALSE  TRUE
```

ベクトル操作: 練習

08:00

1. 要素が39, 49, 100, 73, 54, 57, 61, 9, 27, 65の長さ10のベクトルを作成し、`my_vector5`と名付ける
2. Consoleペインで`length(my_vector5)`を入力してみる
    - `length()`はベクトルの長さを返す関数
3. 7番目の要素を抽出する
4. 7, 10番目の要素を抽出する（`c()`使用）
5. 3番目から7番目の要素を抽出する（`:`使用）
6. 1、4、7、10番目の要素を抽出する（`seq()`使用）
7. 奇数のみ抽出する（論理式利用）
    - 奇数**番目**の要素でなく、要素が奇数であるものを抽出
8. 3の倍数のみ抽出する（論理式利用）
    - 3の倍数 = 3で割り切れる要素

ベクトル操作: 答え

```
# 問1
my_vector5 <- c(39, 49, 100, 73, 54, 57, 61, 9, 27, 65)
```

```
length(my_vector5) # 問2
```

```
## [1] 10
```

```
my_vector5[7] # 問3
```

```
## [1] 61
```

```
my_vector5[c(7, 10)] # 問4
```

```
## [1] 61 65
```

```
my_vector5[3:7] # 問5
```

```
## [1] 100  73  54  57  61
```

```
# 問6
my_vector5[seq(1, 10, by = 3)]
```

```
## [1] 39 73 61 65
```

```
# 問7
my_vector5[my_vector5 %% 2 == 1]
```

```
## [1] 39 49 73 57 61  9 27 65
```

```
# 問8
my_vector5[my_vector5 %% 3 == 0]
```

```
## [1] 39 54 57  9 27
```

データの読み込み

表形式データの種類

- 表形式のデータを扱うソフトはそれぞれのデータタイプを持つことが多い
- 例
  - Excel（`.xlsx`）
  - SPSS（`.sav`）
  - Stata（`.dta`）
  - Open Office / Libre Office（`.ods`）
  - など
- ソフトの種類と関係なく使用可能なデータタイプもある
  - Comma Separated Value（`.csv`）<- 業界標準
  - Tab Separated Value（`.tsv`）
  - など
- 表形式以外のデータもある
  - json、地図データなど

データの保存場所

- プロジェクトXで使用されるデータはプロジェクトXのフォルダに入れておく
- 出来れば、プロジェクトのフォルダないに`Data`などの下位フォルダを作成し、そこにデータを入れておく

.csvファイルとは

- Comma Separated Valueの略
  - 列がカンマ（`,`）で区切られているファイル
- データ分析における標準的なデータフォーマット
  - Excel（`.xlsx`）、SPSS（`.sav`）、Stata（`.dta`）形式などより軽量
  - 拡張子は`.csv`
- R内蔵の`read.csv()`、または{tidyverse}の`read_csv()`関数で読み込み

.csvファイルの読み込み（１）

- {tidyverse}パッケージを読み込む

```
pacman::p_load(tidyverse) # または、library(tidyverse)
```

- プロジェクトフォルダの下位フォルダDataにある`Micro02_01.csv`を読み込み、`MyData1`という名前で格納する。
  - 各列がどのようなデータ型なのかを自動に判定し、その結果も出力される。

```
MyData1 <- read_csv("Data/Micro02_01.csv")
```

```
##
## ── Column specification ────────────────────────────────────────
## cols(
##   ID = col_double(),
##   Pref = col_character(),
##   Zaisei = col_double(),
##   Over65 = col_double(),
##   Under30 = col_double(),
##   Jimin = col_double(),
##   Minshin = col_double(),
```

.csvファイルの読み込み（2）

- データの中身を確認する

```
MyData1
```

```
## # A tibble: 47 x 13
##       ID Pref   Zaisei Over65 Under30 Jimin Minshin Komei Kyosan Ishin
##    <dbl> <chr>   <dbl>  <dbl>   <dbl> <dbl>   <dbl> <dbl>  <dbl> <dbl>
##  1     1 北海道  0.419   29.1    24.7  32.8    30.6 13.4    3.43 11.4
##  2     2 青森県  0.332   30.1    23.9  40.4    24.6 12.8    3.82  8.92
##  3     3 岩手県  0.341   30.4    24.5  34.9    22.4  8.61   5.16 11.2
##  4     4 宮城県  0.596   25.8    27.3  36.7    25.4 13.4    3.97  9.99
##  5     5 秋田県  0.299   33.8    21.4  43.5    22.7 11.2    5.17  7.56
##  6     6 山形県  0.342   30.8    24.8  42.5    21.5 11.8    4.3   7.6
##  7     7 福島県  0.509   28.7    25.2  33.8    28.3 11.0    3.43 10.4
##  8     8 茨城県  0.633   26.8    26.6  40.6    19.0 15.0    6.67 10.1
##  9     9 栃木県  0.622   25.9    26.8  38.8    21.6 12.4   10.9   7
## 10    10 群馬県  0.603   27.6    26.6  42.1    19.3 13.8    5.61 10
## # … with 37 more rows, and 3 more variables: Shamin <dbl>, Region2 <dbl>,
## #   Region6 <dbl>
```

.xlsxファイルの読み込み（１）

- {readxl}パッケージを読み込む

```
# {xlsx}のインストールと読み込み
pacman::p_load(readxl)
```

- プロジェクトフォルダの下位フォルダDataにある`Micro02_02.xlsx`を読み込み、`MyData2`という名前で格納する。

```
MyData2 <- readxl::read_excel("Data/Micro02_02.xlsx")
```

- もし、エクセルファイルが複数のシートで構成され、`n`番目のシートを読み込む場合、

```
MyData2 <- readxl::read_excel("ファイルのパス", sheet = n)
```

.xlsxファイルの読み込み（2）

- データの中身を確認する

```
MyData2
```

```
## # A tibble: 47 x 13
##       ID Pref   Zaisei Over65 Under30 Jimin Minshin Komei Kyosan Ishin
##    <dbl> <chr>   <dbl>  <dbl>   <dbl> <dbl>   <dbl> <dbl>  <dbl> <dbl>
##  1     1 北海道  0.419   29.1    24.7  32.8    30.6 13.4    3.43 11.4
##  2     2 青森県  0.332   30.1    23.9  40.4    24.6 12.8    3.82  8.92
##  3     3 岩手県  0.341   30.4    24.5  34.9    22.4  8.61   5.16 11.2
##  4     4 宮城県  0.596   25.8    27.3  36.7    25.4 13.4    3.97  9.99
##  5     5 秋田県  0.299   33.8    21.4  43.5    22.7 11.2    5.17  7.56
##  6     6 山形県  0.342   30.8    24.8  42.5    21.5 11.8    4.3   7.6
##  7     7 福島県  0.509   28.7    25.2  33.8    28.3 11.0    3.43 10.4
##  8     8 茨城県  0.633   26.8    26.6  40.6    19.0 15.0    6.67 10.1
##  9     9 栃木県  0.622   25.9    26.8  38.8    21.6 12.4   10.9   7
## 10    10 群馬県  0.603   27.6    26.6  42.1    19.3 13.8    5.61 10
## # … with 37 more rows, and 3 more variables: Shamin <dbl>, Region2 <dbl>,
## #   Region6 <dbl>
```

文字化けの場合

- masOSとWindows間の文字コードの違い
  - 世界標準はmacOS, Linuxなどが採用しているUTF-8
- 英数字のみのデータは問題ないケースが多いが、マルチバイト文字（日本語、韓国語、中国語など）では問題が深刻
  - Excel形式でも使用OSによって文字化けが生じる場合も

文字化けの例 (`Pref`列)

```
## # A tibble: 47 x 11
##      ID Pref      Zaisei Over65 Under30   LDP   DPJ Komei Ishin   JCP   SDP
##   <dbl> <chr>      <dbl>  <dbl>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     1 "\x96k\x…  0.419   29.1    24.7  32.8  30.6 13.4   3.43 11.4   1.68
## 2     2 "\x90\xc…  0.332   30.1    23.9  40.4  24.6 12.8   3.82  8.92  3.41
## 3     3 "\x8a\xe…  0.341   30.4    24.5  34.9  22.4  8.61  5.16 11.2   5.29
## 4     4 "\x8b{\x…  0.596   25.8    27.3  36.7  25.4 13.4   3.97  9.99  3.62
## 5     5 "\x8fH\x…  0.299   33.8    21.4  43.5  22.7 11.2   5.17  7.56  5.12
## 6     6 "\x8eR\x…  0.342   30.8    24.8  42.5  21.5 11.8   4.3   7.6   5.2
## 7     7 "\x95\x9…  0.509   28.7    25.2  33.8  28.3 11.0   3.43 10.4   3.24
## # … with 40 more rows
```

文字化けの場合: ケース１

自分のPCがWindows、かつ文字化けが生じた場合

- `read.csv()`を使用した場合

```
My_Data <- read.csv("ファイルのパス", fileEncoding = "UTF-8")
```

- `read_csv()`を使用した場合

```
My_Data <- read_csv("ファイルのパス", locale = locale(encoding = "UTF-8"))
```

- その他言語（簡体字、ハングル、アラビア文字など）のファイルを読み込む際、文字化けが生じた場合、宋に相談

文字化けの場合: ケース２

自分のPCがmacOS、かつ文字化けが生じた場合

- `read.csv()`を使用した場合

```
My_Data <- read.csv("ファイルのパス", fileEncoding = "Shift_JIS")
```

- `read_csv()`を使用した場合

```
My_Data <- read_csv("ファイルのパス", locale = locale(encoding = "Shift_JIS"))
```

- その他言語（中国語、韓国語、アラブ語など）のファイルを読み込む際、文字化けが生じた場合、宋に相談

表形式データの書き出し

`write_csv()`関数を利用

- データを加工した場合、加工済みのデータを保存しておくと、次回からは加工なしで分析を始められる
- `write_csv()`は{tidyverse}を読み込んでおくと使用可能
  - `write.csv()`というR内蔵関数もある
- `write_csv()`の使い方

```
write_csv(オブジェクト名, "保存するファイル名")
```

- **例:** 作業環境内にある表形式データ`My_Data`をプロジェクトフォルダ内の`Data`フォルダに`MyData.csv`という名で保存する。

```
write_csv(My_Data, "Data/MyData.csv")
```

- オブジェクト名は`"`で囲まず、パスやファイル名は`"`か`'`で囲む

関数について詳しく知りたい

- 関数のヘルプを確認する
- Console上に`?関数名`を入力（`()`は不要）

```
# write_csv()関数のヘルプを出力する
?write_csv
```

- Helpペインに関数の説明、使い方、必要な引数、返される情報、例などが出力される。
- すべて英語
- 日本語の情報が欲しい場合、メジャーな関数はググっても出る。

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 宋に質問

- プロジェクト生成: 教科書 第7章
- Rの演算子: 教科書 第7章
- ベクトル操作: 教科書 第7章
- データの入出力: 教科書 第8章

今週の課題

復習

- 教科書第7章と第8章を読む
- コードを直接打ち込みながら結果を確認すること

ブラインドタッチの練習（続き）

ブラインドタッチに自信のない履修者は引き続き、ブラインドタッチの練習

- 毎日欠かさず練習すれば2週間でできるようになる。
- Rの操作はほとんどキーボードで行われるので、ブラインドタッチが出来ない場合、生産性が急落する。