

# 情報処理応用演習

## 第1講: Rの基本的な使い方

SONG Jaehyun (同志社大学)

2020/8/21

# 本日の内容

1. 講義概要
2. R?
3. RStudio
4. 電卓としてのR
5. データの読み込み/書き込み
6. R Markdown

# 講義概要

# 講師紹介



## 宋財泫(そんじえひよん)

- 同志社大学文化情報学部 助教
- 博士（政治学）：神戸大学
- E-mail: jasong@mail.doshisha.ac.jp
- HP: <https://www.jaysong.net>

## 専門分野

- 政治行動論、政治学方法論

## 趣味

- ラーメン

# 講義内容

この授業で教えること

- Rを用いたデータ分析の教科書が読める基礎体力づくり
  - RとRStudioの導入
  - R Markdownの基礎
  - 電卓としてのR
  - 表形式データの読み込み/書き出し
  - 表形式データの操作
  - 可視化(グラフ)

この授業で教えないこと

- データ分析の知識
  - 統計学
  - 因果推論
  - 数学

# 講義内容

- 1日目 (8/21): RとRStudioの基本的な使い方
- 2日目 (8/24): 表形式データの操作、SONGの結婚記念日
- 3日目 (8/25): 可視化入門

## 1日目と2日目の間 (8/22, 23)

- SONG・矢内の参考資料を読む
- 必須: 第2, 6, 7, 8, 9章、第10.1, 10.2章
  - 2日目 (8/24)は以上の内容を熟知したと前提した上で講義を進める
- 推奨: 第12, 13, 14, 15, 16, 17, 18, 23章
  - 2日目 (8/24)と3日目 (8/25)の内容に該当する
  - 講義後に読んでもOK
  - 第23章は講義後に必ず読むこと

# 評価

## 平常点 (20%)

- 授業中の態度
- 積極的に質問する、ソンさんのミスを指摘する = 20点
- 黙ってソンさんの話を聞き、実習をこなす = 10点
- 講義中に歌う、焼きそばを食べる、喫煙・飲酒する = 0点

## 課題 (80%)

- データと結果物を事前に与える。
- 与えられたデータを用い、指定された結果を出すコードを提出
- 期限: 2020年8月30日 (予定)

# 教科書

SONGと矢内勇生先生(高知工科大学)で執筆中の資料

- 『私たちのR: ベストプラクティスの探究』
  - <https://www.jaysong.net/RBook/>
- ネット上で無料で閲覧可能

本講義との関係

講義前後に必ず読んでおくこと

- 1日目: 第2, 3, 4, 6, 7, 8, 9, 22章
- 2日目: 第12, 13, 14, 15章
- 3日目: 第16, 17章

# 参考書

- 以下の本は本講義の内容を一部カバー
  - 『はじめてのRStudio』：初心者向け
  - 『RユーザのためのRStudio[実践]入門』：本講義を終えてから読もう



# RとRStudioの導入

# RとRStudioの導入

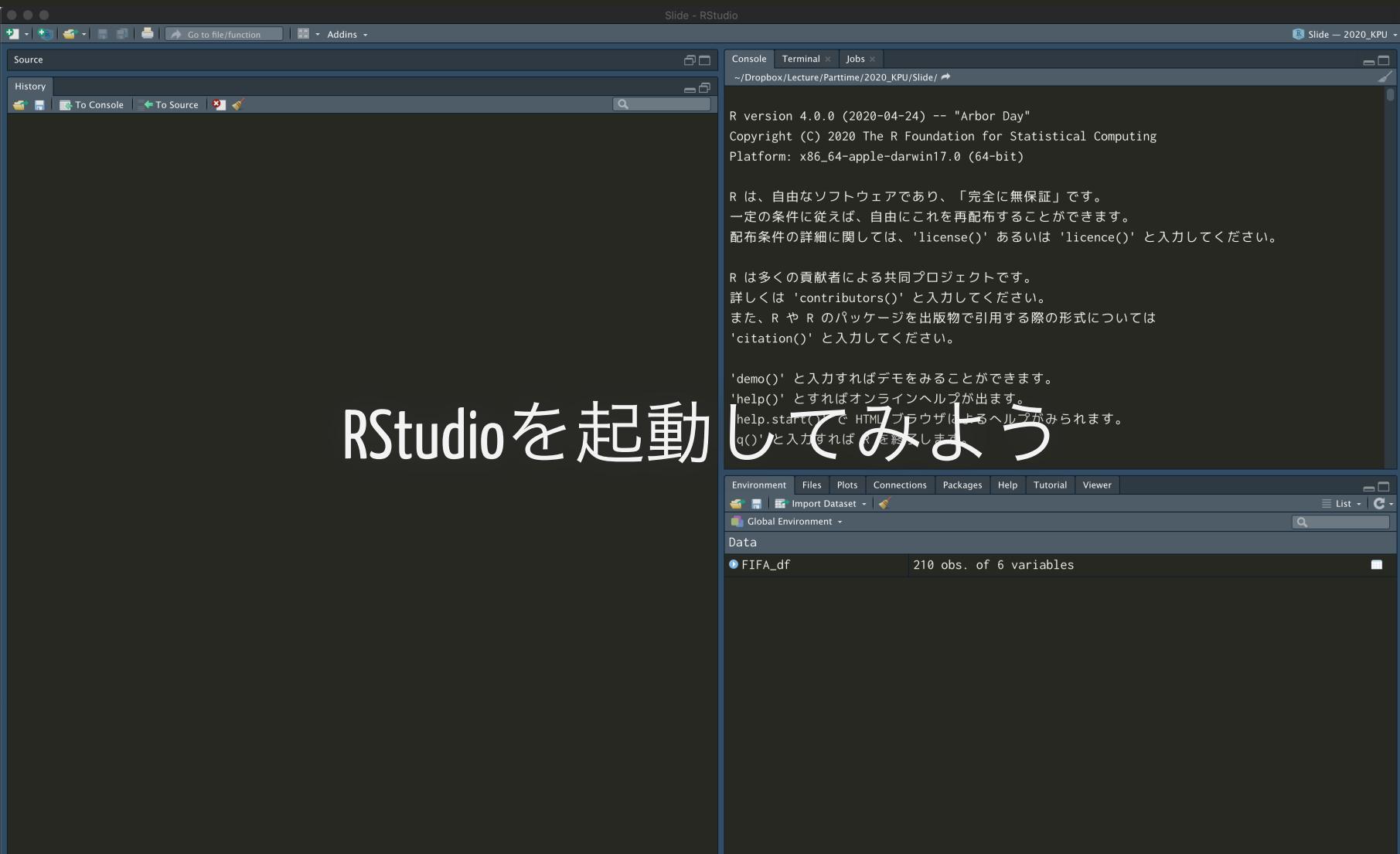
## 講義内

- 大学のPCの場合、インストール済み

## 講義外

- 自分のPCに導入する場合、以下の資料を参照
  - 矢内勇生先生(高知工科大学)の資料
  - RおよびRStudioのインストールについては、(多分)世界一詳しい資料
  - macOS、Linux、Windowsに対応

# RStudioを起動 してみよう



# RStudioをより使いやすく

## RStudio設定画面

- Tools -> Global Options

## オススメの設定

- 矢内先生の資料
  - 宋を含む、多くのRStudioユーザーの標準設定
  - macOS編の40ページ
  - Linux編の27ページ
  - Windows編の104ページ

# プロジェクトの生成

Rを使用する際は必ず、予めプロジェクトを生成しておく

- 簡単な計算をするだけなら、しなくても良い

プロジェクト生成の手順

1. Finderやエクスプローラーからプロジェクトに使うフォルダーを作成する
2. RStudioのFile -> New Project
3. Existing Directoryを選択
4. Project working directory:のBrowseをクリックし、1で作成したフォルダーを指定
5. 今後、プロジェクトを開く際はプロジェクトフォルダーのプロジェクトファイル (.Rproj)を開くだけでOK

# とりあえず、使ってみよう

- 簡単な計算ならConsoleに直接打ち込んでもOK
  - 電卓として使う場合
- ただし、ちゃんとした分析を行うなら全ての分析プロセスを記録として残しておく必要がある。
  - 記録を残さない => ○保方さん...
  - File -> New File -> R Scriptをクリックするとコードを書けるペイン(Pane)が生成される
  - 実行したいコードの行でCmd (⌘) + Return (macOS)、またはControl + Enter (Linux/Windows)を押すと、行の内容がConsoleへコピペ&実行される

# とりあえず、使ってみよう

- $1 + 1$ を計算してみよう
  - Sourceペイン上に $1+1$ と入力し、⌘ + Return

```
1 + 1
```

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

- $5^6$ を計算してみよう
  - Sourceペイン上に $5^6$ と入力し、⌘ + Return

```
5^6
```

```
## [1] 15625
```

# 数値演算子

| 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   |

# 電卓としてのR

- 3と5の足し算を行い、その結果の2乗を求めてみよう
- 任意の奇数を2で割った後の余り(剰余)を求めてみよう
- 任意の偶数を2で割った後の余り(剰余)を求めてみよう

```
(3 + 5)^2
```

```
## [1] 64
```

```
9 %% 2
```

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

```
12 %% 2
```

```
## [1] 0
```

# 代入(格納)

- 19861008 \* 2を計算してみよう
  - 19861008 \* 3を計算してみよう
  - 19861008 \* 4を計算してみよう
  - 19861008 \* 5を計算してみよう
  - 19861008 \* 6を計算してみよう
- ...
- 19861008 \* 100を計算してみよう

# 代入 (格納)

毎回、19861009のような意味深な数字を書くのは面倒くさい

- 19861008をXにすれば保存れば $X * 1$ 、 $X * 2$ 、...で済む
- Xに19861008を「格納」（代入）する( $<-$ )
- Xはオブジェクト(object)と呼ばれる

```
# 19861008をXという名のオブジェクトに格納する
X <- 19861008
X      # Xを出力する
```

```
## [1] 19861008
```

```
X * 3 # X * 3を計算し、出力する
```

```
## [1] 59583024
```

# 数列の作り方

- 19861008はXで置換出来たが、1から100まで掛け算を行うことも面倒
- まずは1から100までの公差1の等差数列を作成する
- seq(最小値, 最大値, by = 公差)
- 公差1の等差数列は最小値:最大値も可能

```
# 以下のコードは Y <- 1:100 と同じ
```

```
Y <- seq(1, 100, by = 1)
```

```
Y
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17  
## [19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35  
## [37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53  
## [55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71  
## [73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89  
## [91] 91 92 93 94 95 96 97 98 99 100
```

# R内の全てのデータはベクトル

- `X <- 3`の場合、`X`は長さ1の数値型ベクトル
- `Y <- 1:5`の場合、`Y`は長さ5の数値型ベクトル
- `Z <- c("A", "B", "C")`の場合、`Z`は長さ3の文字型ベクトル
  - `c()`はベクトルを作成する関数
  - 長さ1のベクトル、または`:`を使う場合、`c()`は不要

```
Z <- c("A", "B", "C")
Z           # Zの中身を出力
```

```
## [1] "A" "B" "C"
```

```
length(Z) # Yの長さを出力
```

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

```
class(Z) # Yのデータ型を出力
```

```
## [1] "character"
```

# ベクトル同士の計算

- 同じ長さのベクトル同士の場合、同じ位置の要素ごとに計算を行う

```
X <- c(1, 2, 3, 4) # 長さ = 4  
Y <- c(2, 3, 5, 7) # 長さ = 4  
X + Y
```

```
## [1] 3 5 8 11
```

- 長さが異なる場合、短い方は繰り返される (recycle)

```
Z <- c(1, 0) # 長さ = 2  
X * Z
```

```
## [1] 1 0 3 0
```

# 19861008

- 19861008 \* 1から\* 100まで

```
X <- 1:100          # 1から100までの公差1の等差数列をXに格納  
Y <- 19861008 * X # 19861008 * Xの結果をYに格納  
head(Y)            # Yの最初の要素6つを出力
```

```
## [1] 19861008 39722016 59583024 79444032 99305040 119166048
```

```
tail(Y)           # Yの最後の要素6つを出力
```

```
## [1] 1886795760 1906656768 1926517776 1946378784 1966239792 1986100800
```

- Yの23番目の要素の結果を出力: オブジェクト名[番号]

```
Y[23]
```

```
## [1] 456803184
```

# 色々計算してみよう

- $1^1, 2^2, \dots, 5^5$ を計算してみよう

```
# 方法1
```

```
X <- 1:5  
X^X
```

```
## [1] 1 4 27 256 3125
```

```
# 方法2
```

```
(1:5)^(1:5)
```

```
## [1] 1 4 27 256 3125
```

- $c(1, 2, 3, 4, 5)$ を2で割った後の余りは?

```
c(1, 2, 3, 4, 5) %% 2
```

```
## [1] 1 0 1 0 1
```

# 論理演算子(1)

- `X == Y`: XとYが同じならTRUE、以外はFALSEを出力
  - `>`, `>=`, `<=`, `<`, `!=`も可能

```
1 + 1 `==` 2
```

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

```
# 1, 2, 3, 4, 5は偶数か否か
c(1, 2, 3, 4, 5) %% 2 == 0
```

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

```
# 1^1, 2^2, ..., 5^5は100以上か
((1:5)^(1:5)) >= 100
```

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

# 論理演算子(2)

- &: かつ
- 例) 1:10で8未満の奇数か否かを判定

```
X <- 1:10  
(X < 8) & ((X %% 2) == 1)
```

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

- |: または
- 例) 1:10で3の倍数か、1を足すと3の倍数になるか否かを判定

```
X <- 1:10  
(X %% 3 == 0) | ((X + 1) %% 3 == 0)
```

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

# 論理演算子の使い道

-2:2から偶数のみ出力したい

- 方法1: TRUE、 FALSEを用いた要素の抽出

```
X <- -2:2 # X <- c(-2, -1, 0, 1, 2)と同じ  
X[c(TRUE, FALSE, TRUE, FALSE, TRUE)] # 1, 3, 5番目の要素を抽出
```

```
## [1] -2 0 2
```

- 方法2: 論理演算子を利用する
  - 偶数 = 2で割った場合余りが0

```
X[X %% 2 == 0]
```

```
## [1] -2 0 2
```

# Rの用語

# Rの用語

- 本講義で登場する重要な用語について
    - カッコ内は教科書において対応する章
    - 週末中に必ず読んでおくこと
1. オブジェクト（第10.1章）
  2. データ型（第8章）
  3. データ構造（第9章）
  4. 欠損値（第8章）
  5. 関数と引数（第10.1章）
  6. 改行と字下げ（第10.2章）

# オブジェクト

オブジェクト (object): メモリに割り当てられた何か

- `X <- Y` 演算子は `Y` を `X` という名でメモリに格納することを意味する
  - ベクトル
  - 行列 (1つのベクトルを2次元で表現したもの)
  - データフレーム (複数のベクトルを表のように並べたもの)
  - 関数など

```
X <- 1:5  
X * 2
```

- オブジェクトは `X` のみ?
  - `X`、`<-`、`1:5`、`*`、`2` 全てが オブジェクト
  - `<-` を使わない場合、一時的にメモリに割り当て、計算が終わったら削除される。

# データ型

## データ内タイプ

- 数値 (numeric)

```
X <- c(1, 2, 1, 2, 3)
```

- 文字 (character)

```
X <- c("A", "B", "A", "B", "C")
```

- 要因 (factor)

```
X <- factor(c("A", "B", "A", "B", "C"),  
            levels = c("A", "B", "C"))
```

- 複素数 (complex)
- 日付 (Date / DateTime)

# データ構造

- ベクトル
  - Rにおける最小単位
- 行列
  - 1つのベクトルを複数の行/列で表したもの
- データフレーム / tibble
  - 複数の縦ベクトルを並べたもの
  - tibbleはtidyverseで提供するデータフレームの拡張版
- リスト
  - ベクトル、行列、データフレームなど様々なデータ構造を内包するもの

# 欠損値

何らかの理由でデータが抜けている箇所

- 世論調査の場合、回答拒否や「分からない」で生じる
- 世界各国データの場合、データが入手困難な場合（北朝鮮など）
- 欠損値はNAで表す

```
X <- c(1, 2, NA, 4, 5, NA, 7)  
X
```

```
## [1] 1 2 NA 4 5 NA 7
```

- 分析の際、NAを含むデータは除去するか、統計的手法を用いて補完する
  - 補完 (multiple imputation) の方法は紹介しない

# 関数

**関数 (function):** 入力されたデータを、関数内部で決められた手順に沿って処理し、その結果を返す

- 関数名 (データ)
- 例) `sum()` 関数は () 内で与えた数値型ベクトルの総和を計算し返す

```
X <- c(2, 3, 5, 7, 11)
sum(X)
```

```
## [1] 28
```

**注意:** 複数のパッケージが同じ名前の関数名を提供している場合

- `filter()` や `select()` など、複数のパッケージで共通する関数
- パッケージ名::関数名() で、どのパッケージの関数かを指定
  - 例) `dplyr::select()`

# 引数

関数内で指定することで、関数の動き方を制御する

- `sum()`、`mean()`関数は欠損値が含まれているデータに対してNAを返す

```
X <- c(1, 2, NA, 4, 5, NA, 7)
sum(X)
```

```
## [1] NA
```

- `na.rm`の引数にTRUEを指定すると、自動的に欠損値を除外して計算を行う

```
sum(X, na.rm = TRUE)
```

```
## [1] 19
```

- 引数を書く順番は決められている
  - 順番通りに書くなら`na.rm` =などは不要だが、明示した方が良い
  - 引数の順番は`help(関数名)`で確認

# help(sum)の例

## Sum of Vector Elements

### Description

sum returns the sum of all the values present in its arguments.

### Usage

```
sum(..., na.rm = FALSE)
```

### Arguments

... numeric or complex or logical vectors.

na.rm logical. Should missing values (including NaN) be removed?

# 改行と字下げ

一行のコードが長くならないように適宜、改行・字下げをする

- 結果には影響を与えないが、コードが読みやすくなる
- 演算子の前後にスペース、, の後にスペースを付けるのは慣例 (^を除く)

```
# 悪い例
sum(c(97, NA, 97, 96, 108, NA, 99, 88, 85, 106, 85, 93, 81, 110, 95,
## [1] 1774
```

```
# 良い例
sum(c(97, NA, 97, 96, 108, NA, 99, 88, 85, 106, 85, 93, 81, 110,
      95, 112, 118, NA, 105, 103, 96),
na.rm = TRUE)
```

```
## [1] 1774
```

# コメント

コードの一部ではあるが、実行されないもの

- #で表記する
- #以降の内容は、当該行において無視される

```
sum(1:5)
```

```
## [1] 15
```

```
sum(1:5) # 1から5までの総和を計算する
```

```
## [1] 15
```

- コードを書く際、こまめにコメントを付けておくと読みやすくなる
  - 自分が書いたコードでも数日後に読んだら、よく分からぬ時が多い

# データの入出力と表データ

# パッケージの導入

パッケージ: データ分析に便利な自作関数の集合

- 2020年7月現在、CRANには約16000個のパッケージが登録されている。

パッケージの導入: `install.packages("パッケージ名")`

- 1回インストールしておけば、パソコンに残る
- 例: tidyverseパッケージのインストール

```
install.packages("tidyverse")
```

- CRANでなく、GitHubに登録されているパッケージ
  - `devtools:::install_github(レポジトリ名/パッケージ名)`
  - 事前にdevtoolsパッケージをインストールしておく必要あり
  - 例) SONGが公開したBalanceRパッケージの場合

```
devtools:::install_github("JaehyunSong/BalanceR")
```

# パッケージの読み込み

パッケージを使用するためには、読み込む必要がある

- `library(パッケージ名)`
- **Rを再起動するたびに読み込む必要がある**
- `tidyverse`: 本講義における最重要パッケージ

```
library(tidyverse)
```

# 表形式データの読み込み

1. データをプロジェクトフォルダー内に入れる
  - 下位フォルダー (Dataなど)を作成し、ここにまとめる
2. データのタイプに応じた関数を使用する
  - .csv: `read_csv()`関数
  - .xlsx: `read_xlsx()`関数
  - 以上の関数は `readr` パッケージが提供している関数であり、`readr` パッケージは `tidyverse` を読み込む際に同時に読み込まれる
  - 他にも `haven` パッケージを使うと、SPSS形式 (.sav)、Stata形式 (.dta)、オーブンドキュメント形式 (.ods) も読み込み可能

# 手作り表形式データ

表形式データは縦ベクトルを並べたもの

- `tibble()`内にベクトルを指定

```
ID      <- c(1, 2, 3, 4)
Name    <- c("Abe", "Xi", "Moon", "Chai")
Country <- c("Japan", "China", "Korea", "Taiwan")
```

```
New_Data <- tibble(ID, Name, Country)
```

```
New_Data
```

```
## # A tibble: 4 x 3
##       ID   Name   Country
##   <dbl> <chr>  <chr>
## 1     1 Abe    Japan
## 2     2 Xi     China
## 3     3 Moon   Korea
## 4     4 Chai   Taiwan
```

# 手作り表形式データ

`tibble()`内で直接ベクトルを作成することも可能

```
New_Data <- tibble(  
  ID      = c(1, 2, 3, 4),  
  Name    = c("Abe", "Xi", "Moon", "Chai"),  
  Country = c("Japan", "China", "Korea", "Taiwan")  
)
```

```
New_Data
```

```
## # A tibble: 4 x 3  
##       ID Name  Country  
##   <dbl> <chr> <chr>  
## 1     1 Abe   Japan  
## 2     2 Xi    China  
## 3     3 Moon  Korea  
## 4     4 Chai  Taiwan
```

# CSVフォーマット

- エクセル (.xlsx)と同様、表形式のファイル
  - メモ帳などでも閲覧、編集可能
- フォント、セルの色、枠線などのデータとして無駄な情報を含まない
  - エクセルより軽量
- データ分析における標準フォーマット
  - エクセルファイルを正確に開けるソフトはエクセルのみ
  - 一方、 .csv はどのソフトでも同じ表が得られる。
- 本講義で使うデータはすべて .csv

# 実習データ1

- [FIFA\\_Men.csv](#): FIFA男子サッカーランキング

Show 5 entries

Search:

| ID | Team           | Rank | Points | Prev_Points | Confederation |
|----|----------------|------|--------|-------------|---------------|
| 1  | Afghanistan    | 149  | 1052   | 1052        | AFC           |
| 2  | Albania        | 66   | 1356   | 1356        | UEFA          |
| 3  | Algeria        | 35   | 1482   | 1482        | CAF           |
| 4  | American Samoa | 192  | 900    | 900         | OFC           |
| 5  | Andorra        | 135  | 1082   | 1082        | UEFA          |

Showing 1 to 5 of 210 entries

Previous

1

2

3

4

5

...

42

Next

# データの読み込み

`read_csv("ファイル名")` 関数で読み込む

- ファイル名にはファイルが入っているフォルダーネームも含む

```
# Dataフォルダー内のFIFA_Men.csvを読み込み、FIFA_dfという名で格納  
FIFA_df <- read_csv("Data/FIFA_Men.csv")
```

# 中身の確認

オブジェクト名を入力するだけ

- 最初の10行が表示される
- 出力する行数を指定したい場合、`print(オブジェクト名, n = 行数)`
- `head()`: 最初の6行、`tail()`: 最後の6行

```
# FIFA_dfの最初の5行を出力
print(FIFA_df, n = 5)
```

```
## # A tibble: 210 x 6
##       ID Team          Rank Points Prev_Points Confederation
##   <dbl> <chr>        <dbl>   <dbl>      <dbl> <chr>
## 1     1 Afghanistan    149     1052      1052  AFC
## 2     2 Albania         66      1356      1356  UEFA
## 3     3 Algeria          35      1482      1482  CAF
## 4     4 American Samoa  192      900       900   OFC
## 5     5 Andorra          135     1082      1082  UEFA
## # ... with 205 more rows
```

# 文字化けしたら?

主にマルチバイト文字(日本語/中国語/韓国語)が含まれているデータで生じる

- `FIFA_Men.csv`はローマ字のみ

解決法: `locale`引数を追加する。

- Windowsの場合
  - 主にデータがUTF-8(世界標準)で記録されている場合に生じる

```
Data <- read_csv("ファイル名",
                  locale = locale(encoding = "utf8"))
```

- macOS/Linuxの場合
  - 主にデータがShift-JIS(Windows独自)で記録されている場合に生じる

```
Data <- read_csv("ファイル名",
                  locale = locale(encoding = "Shift_JIS"))
```

# 列の抽出 (1)

オブジェクト名\$列名で抽出

- ベクトルとして出力される
- 表データは縦方向のベクトルを横に並べたもの

```
# FIFA_dfのRank列
```

```
FIFA_df$Rank
```

```
## [1] 149 66 35 192 135 124 210 126 9 102 200 42 26 114 195 99 187
## [19] 87 1 170 84 168 189 75 49 148 3 208 191 59 59 149 78 173
## [37] 73 193 109 177 17 76 138 10 133 89 56 46 61 6 179 80 95
## [55] 16 184 184 158 63 51 69 4 145 205 104 153 146 110 163 58 2
## [73] 159 91 15 46 196 54 159 199 130 74 118 166 86 62 143 52 39
## [91] 173 33 70 93 13 48 28 97 118 107 116 40 115 147 96 188 137
## [109] 139 152 101 180 131 98 182 91 123 154 155 56 184 100 172 11 175
## [127] 64 183 43 106 136 117 170 14 156 122 151 112 31 68 36 44 82
## [145] 103 81 165 41 21 124 19 7 178 55 34 37 38 131 194 209 181
## [163] 50 20 29 202 118 157 32 64 141 196 71 168 8 206 139 176 167
## [181] 141 17 12 79 161 121 134 113 196 126 203 105 27 29 129 203 77
## [199] 71 5 207 22 85 163 25 94 23 144 88 111
```

# 列の抽出 (2)

オブジェクト名 [, 列名] で抽出

- 複数の列を抽出する場合

```
# FIFA_df の Team, Rank, Points 列  
# , の位置に注意  
Rank_df <- FIFA_df[, c("Team", "Rank", "Points")]  
print(Rank_df, n = 5)
```

```
## # A tibble: 210 x 3  
##   Team           Rank  Points  
##   <chr>          <dbl>  <dbl>  
## 1 Afghanistan    149    1052  
## 2 Albania         66     1356  
## 3 Algeria         35     1482  
## 4 American Samoa 192     900  
## 5 Andorra        135    1082  
## # ... with 205 more rows
```

# 列の抽出 (3)

オブジェクト名 [, 列番号] で抽出

```
# FIFA_dfの2, 3, 6番目列  
# , の位置に注意  
Rank_df <- FIFA_df[, c(2, 3, 6)]  
print(Rank_df, n = 5)
```

```
## # A tibble: 210 x 3  
##   Team           Rank Confederation  
##   <chr>        <dbl> <chr>  
## 1 Afghanistan    149  AFC  
## 2 Albania         66  UEFA  
## 3 Algeria          35  CAF  
## 4 American Samoa  192  OFC  
## 5 Andorra          135  UEFA  
## # ... with 205 more rows
```

# 行の抽出 (1)

オブジェクト[行番号, ]で抽出

```
# FIFA_dfの97, 102行目を出力  
# , の位置に注意  
FIFA_df[c(97, 102), ]
```

```
## # A tibble: 2 x 6  
##   ID Team          Rank Points Prev_Points Confederation  
##   <dbl> <chr>        <dbl>   <dbl>      <dbl> <chr>  
## 1    97 Japan         28     1500      1500  AFC  
## 2   102 Korea Republic  40     1464      1464  AFC
```

## 行の抽出(2)

オブジェクト [TRUE/FALSE, ] で抽出

- Rank列の値が10以下の行を抽出
  - FIFA\_df\$Rank <= 10で判定

```
Rank_Over10 <- FIFA_df$Rank <= 10  
Rank_Over10
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FA  
## [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FA  
## [25] FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FA  
## [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FA  
## [49] FALSE TRUE FALSE FA  
## [61] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FA  
## [73] FALSE FA  
## [85] FALSE FA  
## [97] FALSE FA  
## [109] FALSE FA  
## [121] FALSE FA  
## [133] FALSE FA  
## [145] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FA
```

# 行の抽出 (2)

```
FIFA_df[Rank_Over10, ]
```

```
## # A tibble: 10 × 6
##       ID Team      Rank Points Prev_Points Confederation
##   <dbl> <chr>     <dbl>   <dbl>        <dbl> <chr>
## 1     9 Argentina     9     1623        1623 CONMEBOL
## 2    20 Belgium       1     1765        1765 UEFA
## 3    28 Brazil        3     1712        1712 CONMEBOL
## 4    44 Colombia      10    1622        1622 CONMEBOL
## 5    50 Croatia       6     1642        1642 UEFA
## 6    62 England       4     1661        1661 UEFA
## 7    71 France        2     1733        1733 UEFA
## 8   152 Portugal      7     1639        1639 UEFA
## 9   175 Spain         8     1636        1636 UEFA
## 10   200 Uruguay       5     1645        1645 CONMEBOL
```

# []内に直接打ち込んでOK

```
FIFA_df[FIFA_df$Rank <= 10, ]
```

# 行の抽出: 応用

- Rankが50以下、かつConfederationが"AFC"の行

```
Temp_Vec <- (FIFA_df$Rank) <= 50 & (FIFA_df$Confederation == "AFC")
FIFA_df[Temp_Vec, ]
```

```
## # A tibble: 4 x 6
##       ID Team      Rank Points Prev_Points Confederation
##   <dbl> <chr>    <dbl>   <dbl>     <dbl> <chr>
## 1     12 Australia     42     1457     1457  AFC
## 2     92 IR Iran       33     1489     1489  AFC
## 3     97 Japan          28     1500     1500  AFC
## 4    102 Korea Republic  40     1464     1464  AFC
```

- []内に直接条件式を打ち込んでも良いが、一行が長くなりやすいので、別途のオブジェクトとして格納することを推奨

# 行の抽出: 応用

- Pointsが前より増加した行のみ
  - PointsがPrev\_Pointsより大きいかを判定

```
Temp_Vec <- FIFA_df$Points > FIFA_df$Prev_Points  
FIFA_df[Temp_Vec, ]
```

```
## # A tibble: 4 x 6  
##      ID Team      Rank Points Prev_Points Confederation  
##   <dbl> <chr>    <dbl>   <dbl>      <dbl> <chr>  
## 1     19 Belarus     87     1283        1280 UEFA  
## 2     96 Jamaica     48     1438        1437 CONCACAF  
## 3    137 Nicaragua   151     1051        1050 CONCACAF  
## 4    146 Panama      81     1305        1304 CONCACAF
```

# 要素の抽出

方法1: オブジェクト[行番号, 列番号]で抽出 (結果はデータフレーム形式)

```
# FIFA_dfの97行、2列目の要素を抽出  
FIFA_df[97, 2]
```

```
## # A tibble: 1 × 1  
##   Team  
##   <chr>  
## 1 Japan
```

方法2: オブジェクト\$列名[要素の位置]で抽出 (結果はベクトル形式)

```
# FIFA_dfのTeam列の97番目の要素を抽出  
FIFA_df$Team[97]
```

```
## [1] "Japan"
```

# もっと表をいじりたい

2日目の講義内容: `dplyr`と`tidyverse`パッケージの使い方

- 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
- Wide型とLong型の変換など

3日目は綺麗に整形した表を用いた可視化(グラフの作成)を中心

# データの書き出し

# データの書き出し

ファイルサイズの観点から.csvで保存することを推奨

- write\_csv()関数を使用
- Confederationが"AFC"の行のみを抽出し、AFC\_dfに格納
- AFC\_dfをDataフォルダーのAFC\_Men.csvという名で保存

```
AFC_df <- FIFA_df[FIFA_df$Confederation == "AFC", ]  
write_csv(AFC_df, "Data/AFC_Men.csv")
```

- AFC\_Men.csvをFIFA\_df2という名で読み込み、中身を確認してみよう

# R Markdown 基礎

# R Markdownとは

文章・コード・結果を1つのコードにまとめたもの

- R + Markdown
  - 比較的簡単な文法で構造化された文書の作成が可能
  - Rのコードとその結果を同時に載せることが可能
  - 本講義の最終課題もR Makrdownで提出する
- 出力形式は
  1. HTML (デフォルト)
    - インタラクティブな文書生成可能
  2. Microsoft Word
    - 非推奨
    - 考えた体裁と異なる体裁が得られるケースが多い
  3. PDF
    - 自分のPCに *LATEX* 環境が必要
- スライドの作成可能
  - 本講義のスライドもR Markdown (+xaringanパッケージ)で作成

# RMarkdownの例

```
---
```

```
title: "RMarkdownの例"
author: "Jaehyun Song"
date: "7/24/2020"
output: html_document
```

```
---
```

```
## 好きな食べ物
```

```
ソンさんが好きな食べ物は...
```

```
* ジャガイモのチヂミ
* カムジャタン
* ラーメン
```

```
```{r}
Favorite <- c("ジャガイモのチヂミ" = 80,
             "カムジャタン" = 95,
             "ラーメン" = 100)
barplot(Favorite)
```

```

## RMarkdownの例

Jaehyun Song

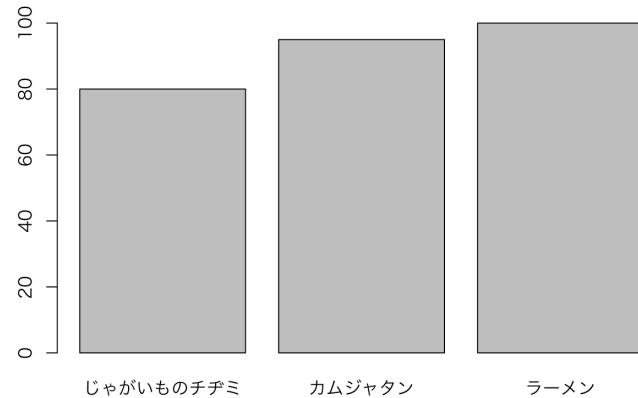
7/24/2020

### 好きな食べ物

ソンさんが好きな食べ物は...

- ジャガイモのチヂミ
- カムジャタン
- ラーメン

```
Favorite <- c("ジャガイモのチヂミ" = 80,
             "カムジャタン" = 95,
             "ラーメン" = 100)
barplot(Favorite)
```



# はじめてのR Markdown

## ファイルの作成

- File -> New File -> R Markdown.. を選択
  - TitleとAuthorには文書のタイトルと作成者
  - Default output formatはHTMLのままでOKの

## とりあえずKnit!

- とりあえず、Sourceペイン上段の  Knit をクリックしてみよう
  - Knitは「ニット」と読む
  - 最初のKnitの際、ファイルを保存する必要がある
  - Viewerペインにプレビューが出力
  - SourceペインとViewerペインの関係を対応させてみよう

# R Markdownの構成要素: ヘッダー

## ヘッダー (Header)

- RMakrdownの最上段に位置し、---と---の間
- 文書のタイトル、作成者、日付などの情報が含まれる
- 自由に修正可能
- output項目は中級者向け

```
---
```

```
title: "R Markdownの例"
author: "Jaehyun Song"
date: "7/24/2020"
output: html_document
```

```
---
```

# R Markdownの構成要素: チャンク

## チャンク (Chunk)

- Rコードの入力箇所
- 必ず ```{r} と ``` で囲む
  - チャンクの外にコードを書いても文章として認識する
- コードの出力->結果の出力の順番

ここはチャンクの外

ここに書いたものは文章として扱われる。

```{r}

ここはチャンク内

ここにコードを書く

ここに書いたものはコードと結果が出力される

```

ここに書いてのも文章

# チャンクオプション

- ``{r, オプション1, オプション2, ...}で指定
- オプションの例
  - コードを見せず、結果だけ表示: echo = FALSE
  - コードのみ表示し、計算は行わない: eval = FALSE
  - 出力される図を中央揃え: fig.align = "center"
  - 警告メッセージを隠す: warning = FALSE
  - 他にも数十のオプションの指定が可能
  - 詳しくは[R Markdown Cheatsheet](#)を参照

# Markdownの文法: 改行

- 文章の改行は空の行を挿入
- 単なる改行では改行されない

**Input:**

```
1行目  
2行目
```

**Output:**

```
1行目 2行目
```

**Input:**

```
1行目  
2行目
```

**Output:**

```
1行目  
2行目
```

# Markdownの文法: タイトル

文章の前に#を付ける

- #が多いほど文字サイズは小さくなる

```
# タイトル1
```

## タイトル1

```
### タイトル2
```

### タイトル2

# Markdownの文法: 項目

順序なしの場合: \* または -

**Input:**

- 項目1
  - 項目1-a
  - 項目1-b
- 項目2
- 項目3

**Output:**

- 項目1
  - 項目1-a
  - 項目1-b
- 項目2
- 項目3

順序付きの場合: 1.

**Input:**

1. 項目1
  - 1. 項目1-a
  - 2. 項目1-b
2. 項目2
3. 項目3

**Output:**

1. 項目1
  - 1. 項目1-a
  - 2. 項目1-b
2. 項目2
3. 項目3

# Markdownの文法: 強調

太字: \*\*と\*\*で囲む

Input:

```
**ラーメン**大好き!
```

Output: ラーメン大好き！

---

イタリック: \*と\*で囲む

Input:

```
*ラーメン*大好き!
```

Output: ラーメン大好き！

下線: <u>と</u>で囲む

Input:

```
<u>ラーメン</u>大好き!
```

Output: ラーメン大好き！

---

取り消し線: ~~と~~で囲む

Input:

```
~~ラーメン~~大好き!
```

Output: ~~ラーメン~~大好き！

# Markdownの文法: インラインコード

文章内にRコードの結果を埋め込む場合

- ・ `r` と ` ` の間にコードを挿入

**Input:**

```
```{r}
x = 5 # 円の半径
````
```

円の半径は`r x`、面積は`r 3.14 \* x^2`です。

**Output:**

```
x = 5 # 円の半径
```

円の半径は5、面積は78.5です。

# Markdownの文法: 文章内コメント

出力に影響されないコメントを入れる場合

- <!--と-->内に囲む
- <!--と-->の間の文章は出力されない
- チャンク内コメントはRと同様、#でOK

**Input:**

```
<!--  
これはコメントです。  
-->  
これは文章です。
```

**Output:**

これは文章です。

# Markdownの文法

ここで紹介したのはMarkdownのごく一部

- 水平線
- リンク
- 画像の埋め込み
- 表の作成など

詳しくは[R Markdown Cheat Sheet](#)を参照

- RStudioのHelp->Cheatsheetsからも確認可能
- ググれば日本語資料もたくさんある

# Rに慣れるためには...

- エラーメッセージをいっぱい出す
- そして、エラーの原因を考え、修正していく
- 出来る限り、色々試し、トラブル・失敗に出会う
- 成功しか経験しないと、トラブルに対応できなくなる
- エラーが出てもパソコンが壊れたり、爆発したりはしない