R で計量政治学入門

土井　翔平

2020-04-19

目次

はじめに

本書は R による計量政治学の入門レベルの講義資料です。質問や間違いなどがありましたら、ご連絡を下さい。筆者のプロフィールはこちらをご覧ください。

想定する読者

本書は、データ分析や数学の前提知識やプログラミング経験のない社会科学系学部生を主たる読者として想定しています。

- やや高度と思われる箇所には * を付けているので、読み飛ばしても構いません。

なお、R や RStudio のインストールについてはR の分析環境を、基本操作についてはR プログラミング入門をご覧ください。

Tidyverse について

Tidyverseとは様々なデータ操作に関するパッケージ群を指します。本書では、可能な限り、R の標準関数を用いた表記と Tidyverse による表記を併記するようにします。しかし、筆者は Tidyverse に慣れているので、しばしば標準関数によるコードを省略します。

ウェブサイトの操作

本書は bookdown を用いて作成しています。ウェブサイトのナビゲーションバーでは、

- 三本線のボタンで目次の表示・非表示の切り替え
- 虫眼鏡のマークで単語検索
- A のマークで文字の大きさ、フォント、色のコントロール
- ダウンロードボタンで.pdf ファイルや.epub ファイルのダウンロード
- i のマークでキーボードによる操作方法の表示

が可能です。

第 I 部

データ・ハンドリング

1　データの読み込み

本章ではデータを読み込む方法について解説します。

```
library(tidyverse)
```

1.1　パッケージ付属のデータ

R は標準でいくつかのデータセットを持っており、またパッケージを読み込むと付属のデータセットも読み込みます。data() に何も入力せずに実行すると、データセットの一覧が表示されます。

```
data()
```

よく、使われるデータセットはフィッシャーのアヤメのデータセットで、iris という名前で保存されています。

```
head(iris)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

- head() は最初のいくつかの要素だけを表示する関数です（tail() は最後からいくつかを表示します）。

1.2 .csv ファイルの読み込み

付録 A R の分析環境

A.1 R のインストール

A.2 RStudio のインストール

A.3 再現可能な分析のために

A.3.1 R スクリプト

A.3.2 R プロジェクト

A.3.3 RStudio の設定 *

付録 B R プログラミング入門

R によるプログラミングの基本として、

- オブジェクト
- 関数
- パッケージ

について解説します。

大雑把に言えば、R ではオブジェクトとしてデータを読み込み、関数によってオブジェクト（＝データ）の処理や分析を行います。パッケージによって様々な関数を追加することで、処理や分析の幅を広げます。

B.1 関数

関数 (function) とは何かを入力すると、何かを出力するものです。例えば、

```
print("Hello, World.")
```

```
## [1] "Hello, World."
```

というコードは、"Hello, World." という文字列を print() という関数に入力し、その文字列を出力しています。

- R では、関数は関数名 () という形を取ります。
- 入力するものを入力引数 (input argument) 、出力するものを出力引数 (output argument) と呼んだりします。

次のように、入力引数も出力引数も 1 つとは限りません。

```
rnorm(n = 10, mean = 0, sd = 1)
```

```
##  [1] -0.6506469 -0.4342129  0.6211014 -0.1553590  0.4437120  1.1861590
```

```
## [7] -0.7941893 -0.1789547  0.9971317  2.1275700
```

さて、この関数は何をしているのでしょうか。R では、関数名の前に? をつけて実行することで、その関数のヘルプを見ることができます。

```
?rnorm
```

英語で関数の使い方が解説されていますが、rnorm(n = 10, mean = 0, sd = 1) は平均 0、標準偏差 1 の（標準）正規分布に従う乱数を 10 個だけ生じさせています。

入力引数は = で明示的に指定する場合、どのような順番でも構いません。

```
rnorm(mean = 0, sd = 1, n = 10)
```

入力引数を明示的に指定しない場合、ヘルプにある順番で入力します。以下の例は上述のものと同じです。

```
rnorm(10, 0, 1)
```

また、ヘルプで mean = 0, sd = 1 のように書かれている場合、デフォルトが定められています。実行者が入力引数を指定しない限り、デフォルト値が使用されます。したがって、以下の例もこれまでと同じコードです。

```
rnorm(10)
```

B.1.1 総称関数 *

総称関数 (generic function) とは、R において入力引数の種類に応じて挙動が変わる関数のことを指します。例えば、summary() という関数はデータフレームが入力引数の場合には記述統計を表示しますが、回帰分析の結果の場合は回帰表を出力します。

総称関数のヘルプを見る場合は、以下のように、関数名に . をつけて入力引数の種類を書きます。

```
?summary.data.frame
?summary.lm
```

B.2 オブジェクト

R では <-でオブジェクトを作成することができます。例えば、20 個の正規分布に従う乱数を x という名前のオブジェクトとして作成します。

```
x <- rnorm(20)
```

- RStudio では <-はショートカット Alt + -で入力できます。

実際に、乱数が x に格納されていることが分かります。

```
x
```

```
##  [1]  1.77021914 -0.76423743 -0.95449331 -1.44845346  1.40140007 -0.75364964
```

```
##  [7] -1.02204476 -0.54747674 -1.96223640  0.25511237  0.89058271 -0.29197463
## [13] -0.90106791  0.57716097  0.09611459  0.61669887 -0.79498937  0.16097148
## [19] -1.09397844  2.27349974
```

オブジェクトを入力引数とすることも可能です。x の平均と標準偏差を求めてみます。

```
mean(x)
```

```
## [1] -0.1246421
```

```
sd(x)
```

```
## [1] 1.115542
```

もちろん、出力引数を新しいオブジェクトにすることもできます。

```
x.mean <- mean(x)
x.mean
```

```
## [1] -0.1246421
```

- オブジェクトの名前にはアルファベットと数字、. と _ が使えます。
- ただし、数字は最初の文字としては使えません。

オブジェクトは上書きすることもできます。

```
x.mean <- mean(rnorm(20))
x.mean
```

```
## [1] 0.1532078
```

- 先ほどとは違う値に上書きされていることが分かります。

B.3 パッケージ

大雑把に言って、R によるデータ分析はデータをオブジェクトとして読み込み、いろいろな関数で処理を行うことで実行します。

つまり、関数が重要なのですが、R で標準に備わっている関数には限界があります。そこで、様々な研究者が関数を作成し、それをまとめたものをパッケージとして公開しています。

- 基本的に、CRANでパッケージは公開されます。

パッケージをインストールするには、install.packages() という関数にパッケージ名を入れて実行します。試しに、Tidyverseという幅広く使われているパッケージをインストールしてみます。

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

- RStudio の場合、Packages パネル（デフォルトの場合は右下）の中に Install というボタンがあり、そこにパッケージ名を入力していインストールすることも可能です。

" でパッケージ名を囲まないとエラーになります。

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

```
## Error in install.packages(tidyverse): object 'tidyverse' not found
```

インストールしたパッケージに対して再び install.packages() を行うと、最新版にアップデートされます。

- RStudio の場合、Packages パネルに Update というボタンがあり、アップデートできるパッケージを自動検索してくれます。

パッケージはインストールしただけでは使用することはできず、library() で読み込む必要があります。

```
library(tidyverse)
```

- この場合は" で囲む必要はありません。
- インストールは一回で十分です。

RStudio であれば Packages パネルにインストール済みのパッケージ一覧があるので、パッケージ名をクリックすると含まれる関数一覧を見ることができます。

- 同様のものはCRANでも pdf 形式で見ることができます。
- 一部のソフトウェアはJournal of Statistical Softwareなどで論文が公開されています。

B.3.1 Tidyverse とは *

Tidyverse とは広義には R におけるデータ処理を行うためのパッケージを開発するプロジェクトであり、狭義にはそこで開発されたパッケージの一部を指します。具体的には、

- ggplot2
- dplyr
- tidyr
- readr
- purrr
- tibble
- stringr
- forcats

になります。

パッケージとしての tidyverse を読み込むことで、上記のパッケージを読み込んでいます。

なお、プロジェクト全体としては、上記のもの以外にも多くのパッケージが開発されています。

付録 C　R プログラミング中級 *

R による、より高度な作業のために

- 代表的なオブジェクトのクラス
- オリジナルの関数の作成
- ループや条件分岐

などを学びます。

C.1　オブジェクトのクラス

オブジェクトの種類をクラスと呼びます。`class()` にオブジェクトを入力するとクラスが分かります。

最も使われるのは数値 ()numeric, real) です。

```
class(1)
```

```
## [1] "numeric"
```

厳密には数値と整数 (integer) は異なりますが、気にしないといけない局面は少ないと思います。

```
class(2L)
```

```
## [1] "integer"
```

他には、文字列 (character) や

```
class("Hello, World.")
```

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

論理値 (logical) などもあります。

```
class(TRUE)
```

```
## [1] "logical"
```

論理値は主に条件式が満たされるかどうかを示します。

```
1 == 1
```

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

```
0 > 2
```

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

ちなみに、TRUE は数値としての 1、FALSE は 0 にもなります。

```
TRUE + 1
```

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

```
FALSE * 2
```

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

また、因子型 (factor) と呼ばれるクラスもあります。カテゴリカル変数と言ったほうが分かりやすいかもしれません。

```
x <- factor(1)
x
```

```
## [1] 1
## Levels: 1
```

```
class(x)
```

```
## [1] "factor"
```

X の中身は 1 ですが、数値ではなくカテゴリーになっているので、数値として操作することはできません。

```
x + 1
```

```
## Warning in Ops.factor(x, 1): '+' not meaningful for factors
```

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

R ではベクトルには特別なクラスは付与されていません。ベクトルは c() に中身を入力して作成します。

```
x <- c(1,3,5)
x
```

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

行列 (matrix) はクラスとして存在します。

```
x <- matrix(c(1,3,5,7), 2, 2)
x
```

```
##      [,1] [,2]
## [1,]    1    5
## [2,]    3    7
```

```
class(x)
```

```
## [1] "matrix"
```

他に、データフレーム (data.frame) やリスト (list) などもあります。

クラスを確認するときは、is.*() の形をとる関数を使います。

```
is.numeric(1)
```

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

```
is.character(1)
```

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

クラスを変更するときは、as.*() のような関数を使います。

```
as.factor(1)
```

```
## [1] 1
## Levels: 1
```

```
as.character(1)
```

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

- 必ずしも全てのクラスが任意のクラスに変換できるわけではありません（例えば、文字列から数値など）。

C.2 関数の作成

R で関数を自作する際は function(){} という関数を使います。

- () の中に入力引数を記述します。
- {} の中に処理内容を記述し、最後に return() で出力引数を指定します。

例えば、数値ベクトルを入力引数として、平均と標準偏差を出力引数とする関数を作成します。

```
mean_sd <- function(x) { # 入力引数の名前を x としておきます。
  mean.x <- mean(x) # 平均を計算します。
  sd.x <- sd(x) # 標準偏差を計算します。
  return(c(mean.x, sd.x)) # 出力引数を指定します。
}
```

実際に実行してみます。

```
x <- rnorm(100)
mean_sd(x)
```

```
## [1] -0.2604071  1.1366948
```

C.3 ループ

ループとは同一の処理を複数回実行することを指します。例えば、100 個の標準正規分布に従う乱数の平均を 5 回求める処理は次のようになります。

```
for (i in 1:5) {
  print(mean(rnorm(100)))
}
```

```
## [1] -0.06236995
## [1] 0.2073594
## [1] -0.05392085
## [1] -0.08990549
## [1] 0.07726384
```

for ループとは () の中の in のあとのベクトルの第 1 要素から順番に i に代入して繰り返しています。そのことは、次の例から解ると思います。

```
head(letters)
```

```
## [1] "a" "b" "c" "d" "e" "f"
```

- letters とはアルファベットのベクトルです。

```
for (i in head(letters)) {
  print(i)
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
## [1] "e"
## [1] "f"
```

- for ループとは別に、特定の条件が満たされるまで繰り返される while ループもあります。

ループ処理の結果を格納するには少しテクニックが必要です。100 個の乱数の平均を 5 回取ったものを x として保存したいとします。

まず、x を NULL オブジェクトとして作成します。

```
x <- NULL
x
```

```
## NULL
```

- NULL とは空っぽのオブジェクト（0 という数値や空白という文字ではない）です。

先程のループ処理の中で、計算した平均を c() で x にくっつけていきます。

```
for (i in 1:5) {
  x <- c(x, mean(rnorm(100)))
}
x
```

```
## [1]  0.002596383 -0.061913105  0.180748913  0.007591530  0.078634594
```

無事、5 個の平均値が x に保存されていることがわかります。

実際に for ループの中で何が起こっているかは、次のコードで解ると思います。

```
x <- NULL
for (i in 1:5) {
  x <- c(x, mean(rnorm(100)))
  print(x)
}
```

```
## [1] 0.01769138
## [1] 0.01769138 0.04974248
## [1]  0.01769138  0.04974248 -0.10868319
## [1]  0.01769138  0.04974248 -0.10868319 -0.09374632
## [1]  0.01769138  0.04974248 -0.10868319 -0.09374632  0.04271317
```

- ループが一周するたびに、前回の x に新しい要素が付け加わり、新しい x として保存されています。

NULL オブジェクトを使ったループ結果の保存でよくあるミスは、やり直す際に NULL でリセットするのを忘れることです。例えば、同じコードをもう一度実行しましょう。

```
for (i in 1:5) {
  x <- c(x, mean(rnorm(100)))
}
x
```

```
##  [1]  0.01769138  0.04974248 -0.10868319 -0.09374632  0.04271317  0.09595001
##  [7]  0.01613145 -0.10123466  0.16745623  0.10335703
```

- x に 10 個の平均値が入っています。

このようなミスを避ける方法の一つは、全体を関数として作成することです。

```r
multi_mean <- function() {
  x <- NULL
  for (i in 1:5) {
    x <- c(x, mean(rnorm(100)))
  }
  return(x)
}
x <- multi_mean()
x
```

```
## [1] -0.1289799  0.1694192 -0.0138531 -0.1463846  0.1965181
```

C.4 条件分岐

条件分岐とは、特定の条件の場合に特定の動作を行うようにすることです。例えば、正の場合 positive、負の場合 negative と出力するコマンドは次のようになります。

```r
x <- rnorm(1)
if (x > 0) {
  print("positive")
} else {
  print("negative")
}
```

```
## [1] "positive"
```

```r
print(x)
```

```
## [1] 0.263542
```

- if(){} の () の中に条件式を書き、{} の中に処理内容を書きます。
- それ以外の条件は else で示します。

条件式は 3 つ以上でも構いません。

```r
x <- rnorm(1)
if (x > -0.5) {
  print("x is less than -0.5.")
} else if (x >= -0.5 & x <= 0.5) {
  print("x is between -0.5 and 0.5.")
} else {
  print("x is more than 0.5. ー")
}
```

```
## [1] "x is less than -0.5."
```

```
print(x)
```

```
## [1] 0.3311985
```

- & は「かつ」を意味します。
- 「または」は | を使います。
- >= は $\geq$ を意味します。
- 「同じ値である」は == を使います（= ではない点に注意）。

C.5 練習問題：フィボナッチ数列

フィボナッチ数列とは以下の条件を満たす数列です。

$$
\begin{aligned}
F_0 &= 0 \\
F_1 &= 1 \\
F_n &= F_{n-1} + F_{n-2} \quad n \geq 2
\end{aligned}
$$

例えば、

$$F_2 = 1, F_3 = 2, F_4 = 3, F_5 = 5, F_6 = 8, ...$$

となります。

フィボナッチ数列の第 n 項を（解析解を使わずに）求める関数を作成してみて下さい。

また、$F_n \geq m$ となるような n を求める関数を作成してみて下さい。

C.6 練習問題：モンテカルロ・シミュレーション

モンテカルロ・シミュレーション（モンテカルロ法）とは乱数を用いて近似解を求める手法です。

例えば、円周率 π の近似解は以下のように求めることができます。

1. 0 以上 1 未満の一様分布から n 個の乱数 x_i と n 個の乱数 y_i を発生させます $(i = 1, 2, ..., n)$。
2. 原点と (x_i, y_i) の距離が 1 以下である回数を計算し n_1 とします。
3. 円周率の近似解として $\hat{\pi} = 4 \times n_1/n$ を得ます。

モンテカルロ・シミュレーションによる円周率の近似解を求める関数を作成してみて下さい。

また、モンテカルロ・シミュレーションによる円周率の近似解を m 回求めて、その平均値や標準偏差が n によってどのように変化するか検討してみて下さい。

付録 D　R Markdown 入門

動作環境

```
sessionInfo()
```

```
## R version 3.6.3 (2020-02-29)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.4 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] forcats_0.5.0   stringr_1.4.0   dplyr_0.8.5     purrr_0.3.4
## [5] readr_1.3.1     tidyr_1.0.2     tibble_3.0.0    ggplot2_3.3.0
## [9] tidyverse_1.3.0
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.0.0 xfun_0.13        haven_2.2.0      lattice_0.20-41
##  [5] colorspace_1.4-1 vctrs_0.2.4      generics_0.0.2   htmltools_0.4.0
##  [9] yaml_2.2.1       rlang_0.4.5      pillar_1.4.3     withr_2.1.2
## [13] glue_1.4.0       DBI_1.1.0        dbplyr_1.4.2     modelr_0.1.6
## [17] readxl_1.3.1     lifecycle_0.2.0  munsell_0.5.0    gtable_0.3.0
## [21] cellranger_1.1.0 rvest_0.3.5      evaluate_0.14    knitr_1.28
## [25] fansi_0.4.1      highr_0.8        broom_0.5.5      Rcpp_1.0.4.6
```

```
## [29] backports_1.1.6  scales_1.1.0     jsonlite_1.6.1   fs_1.4.1
## [33] hms_0.5.3        digest_0.6.25    stringi_1.4.6    bookdown_0.18
## [37] grid_3.6.3       cli_2.0.2        tools_3.6.3      magrittr_1.5
## [41] crayon_1.3.4     pkgconfig_2.0.3  ellipsis_0.3.0   xml2_1.3.1
## [45] reprex_0.3.0     lubridate_1.7.8  assertthat_0.2.1 rmarkdown_2.1
## [49] httr_1.4.1       rstudioapi_0.11  R6_2.4.1         nlme_3.1-144
## [53] compiler_3.6.3
```