R で計量政治学入門

土井　翔平

2020-04-18

目次

はじめに 1

第 I 部　データ・ハンドリング 2

1 データの読み込み 2
1.1 使用するパッケージ 2

付録 A　R の環境構築 2

付録 B　R プログラミング入門 2
B.1 関数 2
B.2 オブジェクト 3
B.3 パッケージ 4
B.4 R スクリプト 5
B.5 R プロジェクト 5
B.6 関数の作成 5
B.7 ループ 6
B.8 条件分岐 8
B.9 練習問題：フィボナッチ数列 9
B.10 練習問題：モンテカルロ・シミュレーション 9

動作環境 10

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。

R や RStudio のインストールについてはR の環境構築を、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。

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。

1.1　使用するパッケージ

```
library(tidyverse)
```

付録 A　R の環境構築

付録 B　R プログラミング入門

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] -0.8331044 -0.7309096 -0.4894784  0.4881421 -0.6218853 -0.7934651
##  [7]  0.6577110  0.6371939  0.7022634 -1.0800785
```

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。x

```
rnorm(10)
```

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、100 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(100)
```

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

```
x
```

```
##   [1] -0.517361520 -1.455387414  0.821016180  0.333605967  0.651434049
##   [6] -1.501008900 -0.014855765  0.447378744 -0.297534041  0.494646784
##  [11] -1.767282811  1.032372135 -0.687838087 -1.703715358  0.131227738
##  [16] -0.686595769  0.171029683  0.191001956 -0.119933429  1.007889187
##  [21] -0.801638540  1.025438385  0.210437711 -0.244863002 -0.513489079
##  [26]  1.660765742  0.515620066 -0.461772887 -1.528595764  0.885853408
##  [31]  0.743839922  0.050017167  0.884489742  0.319596910  0.697098852
##  [36] -1.073938097  2.084973856  0.999887474 -2.066161905 -0.488081071
##  [41]  0.765713260 -0.790802069 -0.669135617 -1.561738371  1.878666375
##  [46]  0.039346710 -0.519496868  0.213101325 -1.076810499 -2.694418310
##  [51]  1.007949015 -0.026456341 -1.407199440  0.382300935  0.042891840
##  [56]  1.498469621  1.315473927 -1.847780192  0.918504391  1.373817246
##  [61]  1.053382991  2.223008479  0.127003622  0.232235657 -0.674348200
##  [66] -1.296855321  0.708563953 -0.119832247 -0.236172296 -0.469442022
##  [71] -0.039545631 -0.332874693  0.535556422 -0.417675525  0.847156097
##  [76]  0.298682105  0.370557088  0.070012222  1.171802489  1.961270998
```

```
##  [81]  0.133071162 -0.129005427  0.502058157 -0.506058958  0.223770433
##  [86] -0.979688903 -0.797500792 -0.052707505  0.398869079  0.710285192
##  [91]  0.678543554 -1.012239450  0.626761044  2.881473815  1.354093183
##  [96]  1.418564731  2.117708421 -0.167081298  0.003465883 -0.731508756
```

実際に、乱数が x に格納されていることが分かります。

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] 0.1095732
```

```
sd(x)
```

```
## [1] 1.020188
```

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] 0.1095732
```

- 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(100))
```

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CRAM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

B.4 R スクリプト

B.5 R プロジェクト

B.6 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] -0.08496779  0.90721023
```

B.7 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] 0.1320865
## [1] -0.02878453
## [1] -0.07223823
## [1] -0.1101577
## [1] -0.007778733
```

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] -0.03359415 -0.03433155 -0.13509150  0.01220645  0.12029171
```

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] 0.111222
## [1] 0.11122199 0.03246351
## [1]  0.11122199  0.03246351 -0.01666441
## [1]  0.11122199  0.03246351 -0.01666441 -0.11080168
## [1]  0.11122199  0.03246351 -0.01666441 -0.11080168  0.11804347
```

- 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]  0.11122199  0.03246351 -0.01666441 -0.11080168  0.11804347 -0.03910354
##  [7]  0.03258709 -0.11114005 -0.01832329  0.26131473
```

- 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。

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

```
## [1]  0.02371186 -0.07317963  0.12399518  0.11874866 -0.03259594
```

B.8 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。

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

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

```
print(x)
```

```
## [1] -0.3383114
```

- 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。

```
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] 0.3721476
```

- & 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）。

B.9 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。

B.10 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。

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_0.3.3
## [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      broom_0.5.5      Rcpp_1.0.4.6     backports_1.1.6
## [29] scales_1.1.0     jsonlite_1.6.1   fs_1.4.1         hms_0.5.3
## [33] digest_0.6.25    stringi_1.4.6    bookdown_0.18    grid_3.6.3
```

```
## [37] cli_2.0.2         tools_3.6.3       magrittr_1.5      crayon_1.3.4
## [41] pkgconfig_2.0.3   ellipsis_0.3.0    xml2_1.3.1        reprex_0.3.0
## [45] lubridate_1.7.8   assertthat_0.2.1  rmarkdown_2.1     httr_1.4.1
## [49] rstudioapi_0.11   R6_2.4.1          nlme_3.1-144      compiler_3.6.3
```