R で計量政治学入門

土井　翔平

2020-04-18

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。

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]  0.5100836 -1.9178627  0.3617162  1.6663065  0.6966553  0.9176074
##  [7] -0.9652590 -1.8262058  1.3781373  0.3726935
```

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] -0.634329890 -0.933581635  0.500597309 -0.237829713  1.134620433
##   [6]  0.157317447 -1.278441403 -0.169100490  0.953810676 -0.653200473
##  [11] -0.796310222  0.221745769  1.113032644  0.851603629  0.389395880
##  [16] -0.060482461  0.532422515  0.130012414  0.916426646 -1.130660001
##  [21]  1.519160421 -2.094265007  1.478230782  0.029008353 -1.062132403
##  [26]  0.011818349 -0.153622769 -0.322972427 -1.214824810 -0.752159414
##  [31]  0.598106395 -0.223620657 -0.077004346  1.051250545  0.017722334
##  [36]  0.331179280  0.368518002 -0.045472980  0.867805121 -0.767005798
##  [41]  0.154741473  0.004048953 -1.463517484  1.780155568 -0.626764309
##  [46] -0.989029395 -0.519081426  1.311618083  1.710519721  0.675993705
##  [51] -1.629270254  0.158816347 -1.152727667  2.876251126  0.303803216
##  [56] -0.020521977  0.542121676  0.529408201  0.393019694 -1.206772558
##  [61] -0.563868534 -1.578226881 -0.105160658 -0.066183955  1.165629714
##  [66] -2.146351536 -1.535515668  0.723146042  2.674476389  1.049104757
##  [71] -1.119754490 -1.001463143  0.989195459  0.356253324  0.568064365
##  [76] -1.683334863 -1.155942467 -0.857308916  1.127182373 -0.964649493
##  [81]  0.235484780 -1.904939252  0.492043196  0.711809763  0.026034740
##  [86] -0.137748542 -0.267288545 -0.373703486  0.980628346  0.067536281
##  [91] -0.112648097 -2.473260415 -0.218552691  0.027380917 -0.438420060
##  [96] -0.579576355 -0.878651361 -0.912948960  1.264377248 -1.464715491
```

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] -0.06682285
```

```
sd(x)
```

```
## [1] 1.010005
```

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] -0.06682285
```

- 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。

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.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] 0.04921945 1.03281027
```

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] 0.06185255
## [1] -0.0497178
## [1] -0.02262662
## [1] 0.1639038
```

```
## [1] -0.07830911
```

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]  0.04482331  0.15245047 -0.20079446  0.14338999  0.09254384
```

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。

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

```
## [1] 0.05292771
## [1] 0.05292771 0.06922472
## [1]  0.05292771  0.06922472 -0.01338469
## [1]  0.05292771  0.06922472 -0.01338469  0.24274790
## [1]  0.05292771  0.06922472 -0.01338469  0.24274790  0.20090668
```

- 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。

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

```
##  [1]  0.05292771  0.06922472 -0.01338469  0.24274790  0.20090668  0.06541486
##  [7]  0.14932941 -0.08364625  0.03719465  0.07494928
```

- 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]  0.36437285  0.08879437 -0.07453318 -0.08601525  0.03688836
```

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] "positive"
```

```
print(x)
```

```
## [1] 0.2608707
```

- 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 more than 0.5.ー"
```

```
print(x)
```

```
## [1] -2.039521
```

- & 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）。