第 12 章: 函式寫作

12: Function Writing

R 程式語言其中的一項能力是允許使用者建立自己的 函式物件 (function object) 或 函式 (function). R 系統的大多數函式都是 R 系統的一部分, 如 `mean()`, `var()` 等等. 這些函式都是用 R 程式語言寫成的, 在本質上和使用者撰寫的沒有太大差別. 通過 R 的函式撰寫與編輯過程, 擴展 R 在程式語言在使用上的功能性與便利性.

12.1 函式撰寫與編輯

一個 R 基本函式 通常是通過下面類似的語句形式定義:

```
1 > function.name(arg.1, arg.2, arg.3 = value.3, ...)
```

函式中的引數分別爲:

- `arg.1, arg.2` 爲 必要引數 (required argument).
  爲使用者必須輸入引數值.
- `arg.3 = value.3` 爲 選擇引數 (optional argument).
  選擇引數 附有一個 `=` (等號), 例如, `arg.3 = value.3`. 表示必須使用的引數 `arg.3`, 但不一定必須輸入到函式 `function.name()` 之內, 若使用者沒有輸入引數值, 則函式會直接使用 R 的內部自動設定的引數值 `value.3`.

- `...` 爲 省略引數 (ellipsis argument).
  不一定須要存在函式中, 通常是將省略引數傳送到 函式 `function.name()` 所使用到的其他函式.
- 函式 `function.name()` 運算的最終結果 (數值, 物件), 是函式回傳的物件.
- 函式 `args(function.name)` 可以查看函式的必要引數以及引數內部自動設定值. 引數並沒有特定的型態, 任何物件都可能成爲一個引數, 必要引數名字應該在寫在內部自動設定值引數之前.

一個使用者自建 函式, 寫法與 R 基本內建函式類似, 通常是通過下面類似的語句形式定義:

```
> function.name <- function(arg.1, arg.2, ...){
      do expression1
      do expression2
      code to be executed ...
      }
```

函式內的運算式常常是一個 R 用 大括號, `{expr}`, 圍成區塊的運算式. 其中的主要引數分別爲:

- 大括號: `{expr}`, 圍成區塊的運算式內容, 由 R 一個或多個運算式敘述所組成, 各運算式之間用 換行 或 分號 分開.
- 運算式 `expression1`, `expression2` ...
- `arg.i`, $i = 1, 2, \ldots$, 提供 運算式 `expression` 進行運算或計算使用.
- `arg.i`, $i = 1, 2, \ldots$, 需要由逗號 (`,`) 分隔.
- `arg.i` 可以在使用時輸入.
- 若使用者未輸入 `arg.i` 時, 可以使用引數指令 `arg.i = arg.def.i`, 讓 R 內部的引數值會成爲 `arg.def.i`, 而成爲 選擇引數 (optional argument) 或 自動設定值 (default values).
- 函式傳回的值, 就是函式內區塊運算式的最後的結果.

- 輸入函式指令 function.name, 且不帶括號使用, 則 R 顯示函式定義, 而不是執行函式.

函式撰寫與編輯完成後, 可以在 R 的任何地方以

```
> function.name(arg.1, arg.2, ...)
```

的形式使用, 在 R 中使用函式名稱 (function.name), 不帶括號, 則 可以顯示函式之定義. 在函式運算式中, 新生定義之變數 (或物件) 是函式的一部分, 當執行函式的時候, 新定義之變數 (或物件) 才存在. 在函式運算式中, 新生定義之變數 (或物件) 在一函式內部儲存區隔, 因此, 使用可以定義函數內部的新變數 (或物件) 時, 與函式之外 R 內已有的變數 (或物件) 可以有相同的名稱, 但是它無法影響 R 原有的物件儲存.

首先撰寫一個簡單的自製函式 (homemade function) 例子, 用來計算 x 的函數值域 (當然還有其他更簡單的方法得到一樣的結果), 並使用自製函式.

$$f(x) = x^3 + x, \tag{12.1.1}$$

```
> ## first simple homemade function
> f.hm <- function(x){x^3+x}
> x <- 1:5
> f.hm(x)
[1]   2  10  30  68 130
> f.hm(x/3)
[1] 0.3703704 0.9629630 2.0000000 3.7037037 6.2962963
> f.hm(x/pi)
[1] 0.3505614 0.8946320 1.8257211 3.3373377 5.6229912
```

另一個簡單的函式例子,

$$\frac{df(x)}{dx} = 3x^2 + 1, \tag{12.1.2}$$

```
> ## homemade function: derviative
> f.dev.hm <- function(x){3*x^2 + 1}
> x <- 1:5
> f.dev.hm(x)
[1]  4 13 28 49 76
> f.dev.hm(x/3)
```

```
[1] 1.333333 2.333333 4.000000 6.333333 9.333333
> f.dev.hm(x/pi)
[1] 1.303964 2.215854 3.735672 5.863417 8.599089
```

對於一個已有的函式, 在 R 中可以使用函式 `fix(function.name)` 函式來修改, 如:

```
> fix(f.dev.hm)
```

R 會開啟一個編輯視窗, 顯示函式的內容, 修改後再關閉編輯視窗, 修改就完成了.

一般函式多須反覆編輯與測試, 在 R 指令提示符號 (>) 之下, 輸入或編輯自製函式, 通常會不方便修改, 一般編輯函式, 是先用文字編輯軟體輸入或編輯自製函式的定義, 儲存成純文字檔案, 檔案應該僅包含文字和空白沒有其他的型態資訊, 例如儲存到了 `"C:\RData\fhmdev.r"`, 然後再用函式 `source()` 呼叫進入 R 工作環境視窗內使用或執行. 任何 R 程式都可以用這種方式編好, 儲存成純文字檔案, 再輸入R工作環境內. 修改自製函式, 也可以在文字編輯軟體內反覆編輯.

```
> source("C:/Rdata/fhmdev.r")
```

12.1.1 函式之引數 Arguments

函式基本的之引數型態包含 必要引數 (required argument), 爲使用者必須輸入引數值. 選擇引數 (optional argument), 可有自動內設值, 不用不一定必須由使用者輸入到函式之內, 若使用者沒有輸入引數值, 則函式會直接使用 R 的內部自動設定的引數值. 省略引數 (ellipsis argument), 不一定須要存在函式中, 通常是將省略引數傳送到 函式 `function.name()` 內, 作爲其他函式的引數.

函式內引數並沒有特定的型態, 任何物件都可能成爲一個引數, 在函式的引數列 `(arg.1, arg.2, ...)` 中, 必要引數 應該在寫在內部自動設定值引數之前. 查看函式的 必要引數 以及引數內部自動設定值, 可以使用函式 `args(function.name)`. 例如,

```
> ## args(): arguments
> args(var)
function (x, y = NULL, na.rm = FALSE, use)
NULL
```

```
> args(f.hm)
function (x)
NULL
```

使用一個函式時, 若這個函式需要使用者輸入引數, 有 3 種常見方式:

```
> f.name(arg.1 = value.1, arg.2 = value.2, ..., arg.6 = value.6, ... )
> f.name(value.1, value.2, value.3, ...)
> f.name(arg.2 = value.2, arg.6 = value.6, ..., arg.4 = value.4, ... )
```

1. `arg.i = value.i`: 依序寫出引數名與引數值輸入.
2. 若使用者只輸入引數值 `value.i`, 而省略引數名字, 則必須依照函式定義引數時的順序, 依序輸入.
3. 以 `arg.i = value.i`, $i = 1, 2, \ldots$, 的方式輸入時, 使用者可以不用考慮引數在函式定義引數時的順序, 可以任一的順序輸入.

例如, 使用以下的方式定義的函式 `f.name()`, 內有 3 個必要引數, `data.frame`, `group.vec`, `x.vec`, 而 `df.object`, `y`, `x` 等爲 R 工作環境內的存在的物件,

```
> f.name <- function(data.frame, group.vec, x.vec) {
                    do expression1
                    do expression2
                    code to be executed
        }
```

好幾種方式可以使用函式, 例如下面所有的使用函式方式都是相同的.

```
> ## (1) arg.i = value.i
> f.use2 <- f.name(df.object = df.object, group.vec = y, x.vec = x)
> #
> ## (2) only argument values.i
> f.use1 <- f.name(df.object, y, x)
> #
> ## (3) arg.i = value.i
> f.use3 <- f.name(x.vec = x, group.vec = y, data.frame = df.object)
```

```
> ## input arguments
> x.vec <- 1:5
> x.vec[3] <- NA
> mean(x.vec)
[1] NA
> #
```

```
> help(mean)
> ## mean(x, trim = 0, na.rm = FALSE, ...)
> mean(x = x.vec, trim = 0.1, na.rm = TRUE)
[1] 3
> mean(x.vec, 0.1, TRUE)
[1] 3
> mean(na.rm = TRUE, x = x.vec, trim = 0.1)
[1] 3
> mean(x = x.vec, na.rm = TRUE, trim = 0.1)
[1] 3
> mean(FALSE, x.vec)
Error in mean.default(FALSE, x.vec) :
  'trim' must be numeric of length one
```

許多時候, 函式的自選引數會有內部自動設定值, 一般會設定一些常用的預設值, 如果這些引數的預設值適合使用者之需要, 使用者可以省略輸入這些有自動設定值的引數. 例如, 函式 `f.name()` 以下面的方式定義

```
> f.name <- function(data.frame, group.vec, x.vec = c(1:5)) {
                    do expression1
                    do expression2
                    code to be executed
        }
```

輸入函式 `f.name()` 的引數時, 可以省略輸入 `x.vec`, 因爲引數 `x.vec` 有自動設定值 (= c(1:5)), 若引數 `x.vec` 的自動設定值不適合使用者之需要, 則使用者可以改變引數 `x.vec` 的自動設定值.

```
> f.use1 <- f.name(df.object, y) # use default value
> f.use2 <- f.name(df.object, y, c(11:15)) # change
> f.use3 <- f.name(df.object, group.vec = y,
                   x.vec = c(11:15)) # change
> f.use4 <- f.name(x.vec = c(11:15), group.vec = y,
                   data.frame = df.object) # change
```

```
> ## arg. default value
> x.vec <- 1:100
> x.vec[3] <- NA
> mean(x.vec) # default na.rm = FALSE
[1] NA
> mean(x = x.vec, na.rm = TRUE) # defalut trim = 0
[1] 50.9798
> mean(na.rm = TRUE, x = x.vec, trim = 0.2)
[1] 51
```

引數的自動設定值, 可以是任何運算式, 甚至是函式本身所帶有的其他函式之引數, 例如, 引數 na.rm 爲 R 系統內許多函式, 如 mean(), var() 等之引數, na.rm = TRUE 可以將資料物件的缺失值計算, 自動移除缺失值 (NA). 下列定義函式 fun.desc(), 其中的自選引數 na.del = TRUE 之自動設定值爲 TRUE, 可以將資料物件的缺失值計算, 自動移除缺失值 (NA).

```
> ## optional argument and efault value
> fun.desc <- function(x.vec, na.del = TRUE, graph = TRUE){
    if(graph = =  TRUE){boxplot(x.vec)}
    x.num <- length(x.vec)
    x.mis <- sum(is.na(x.vec))
    x.use <- x.num - x.mis
    x.mean <- mean(x.vec, na.rm = na.del)
    x.med <- median(x.vec, na.rm = na.del)
    x.var <- var(x.vec, na.rm = na.del)
    x.min <- min(x.vec, na.rm = na.del)
    x.max <- max(x.vec, na.rm = na.del)
    x.rgn <- x.max - x.min
    round(cbind(x.num, x.mis, x.use,
          x.mean, x.med, x.var,
          x.min, x.max, x.rgn), digit = 2)
    }
> #
> x.use.vec <- rexp(20, rate = 0.001)
> x.use.vec[c(4, 13)] <- NA
>
> fun.desc(x.use.vec, na.del = FALSE)
     x.num x.mis x.use x.mean x.med x.var x.min x.max x.rgn
[1,]    20     2    18     NA    NA    NA    NA    NA    NA
> fun.desc(x.use.vec, graph = FALSE)
     x.num x.mis x.use x.mean  x.med    x.var x.min   x.max   x.rgn
[1,]    20     2    18 859.57 576.39 674617.4 72.71 2686.95 2614.24
> fun.desc(x.use.vec)
     x.num x.mis x.use x.mean  x.med    x.var x.min   x.max   x.rgn
[1,]    20     2    18 859.57 576.39 674617.4 72.71 2686.95 2614.24
> #
> ## compare R summary()
> summary(x.use.vec)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  72.71  246.60  576.40  859.60 1314.00 2687.00       2
```

可省略之引數或變數之選擇, (...), 稱爲 省略引數, (ellipsis), 通常 ellipsis 引數被放置在函式定義內引數列的最後一個, 當作函式的引數, 使用者可以輸入其中一些選擇引數, 也可以省略, 但不會影響函式的運算.

```
> f.name <- function(data.frame, group.vec, x.vec = c(1:5), ...) {
                    do expression1
                    do expression2
                    code to be executed
      }
```

12.2 函式作用域 Scope

在函式內部的變數可以分爲 3 類: 形式參數 (formal parameter), 局部變數 (local variable) 和 自由變數 (free variable). 形式參數 是出現在函數的引數列中的引數, 它們的值與函式實際的引數值 結合 (binding), 進而成形式參數. 局部變數 由函式內部的運算式之值結合成的. 既不是 形式參數 又不是 局部變數 的變數是 自由變數. 自由變數 如果被指派數值將會變成 局部變數. 考慮以下的函式定義:

```
> f.name <- function(x.arg) {
         y.local <- 2*x.arg
         print(x.arg)
         print(y.local)
         print(z.free)
      }
```

在這個函式 `f.name` 中, 引數 `x.arg` 是 形式參數, 由使用者輸入; `y.local` 是 局部變數, 由函式內部的運算式指派; `z.free` 是 自由變數.

任何在函式內部的 普通指派值 (assign) 都是 局部 (local) 且暫時的, 當退出函式時都會遺失. 若一個 變數物件 能夠在函式內部與外部都能使用, 稱爲 總體變數 (global variable). 如果想在一個函式裏面改變 總體變數 (global variable) 的指派值或者成爲 永久指派, 使用者可以在函式內採用 "強迫指派" (super-assignment), 函式內的指派符號爲 `< <-` 或者採用函式 `assign()` 與 引數 `env = .GlobalEnv`. 函式內的局部變數僅是在函式內作用, 對函式內的局部變數給與指派值, 成爲局部變數值, 當函式結束執行後, 局部變數值就刪除, 不影響原來相同名稱之總體變數值.

```
> ## scope
> x.global <- 20
> f.hm <- function(x){
    x.local <- sqrt(20)
```

```
    x.global <- x.local # x.golbal changed locally
    y.global < <-  10     # global variable assignment
    cat("local variable x.local = ", x.local, "\n")
    cat("local variable x.global = ", x.global, "\n")
    }
> ## local and global
> f.hm(x.global)
local variable x.local = 4.472136
local variable x.global = 4.472136
> x.global
[1] 20
> y.global
[1] 10
> x.local
Error: object 'x.local' not found
> #
> ## scope
> x.global <- 20
> f.hm <- function(x){
     x.local <- sqrt(20)
     assign("x.global", x.local, env = .GlobalEnv)  # force assignment
     assign("y.local", x.local)
     z.local < <-  x.local   # force assignment
     cat("local variable x.local = ", x.local, "\n")
     cat("local variable y.local = ", y.local, "\n")
     cat("local variable z.local = ", z.local, "\n")
     cat("local variable x.global = ", x.global, "\n")
     }
> # local and global
> f.hm(x.global)
local variable x.local = 4.472136
local variable y.local = 4.472136
local variable z.local = 4.472136
local variable x.global = 4.472136
> x.global # value did not changed
[1] 4.472136
> x.local  # not found
Error: object 'x.local' not found
> y.local  # not found
Error: object 'y.local' not found
> z.local  # found
[1] 4.472136
```

首先考慮下列 R 函式:

```
## lexical scope
> cube.fun <- function(Zvar) {
      sq.fun <- function() {Zvar*Zvar}
      Zvar*sq.fun()
    }
```

在上述 R 函式 cube.fun() 的引數列中, 變數 Zvar, 在第一次出現爲 cube.fun() 的 形式參數, 可以將 Zvar 設定成爲另一子函式 sq.fun() 內的 自由變數, Zvar, 且也可以在另一個子函式內成爲 形式參數 或 局部變數. 變數進行如此的操作稱爲 詞彙作用域 (lexical scope). 上述函式 sq.fun() 中的自由變數 Zvar, 不是子函式 sq.fun() 中的引數. 因此 Zvar 在 子函式 sq.fun() 中是自由變數. 在 R 內, 當子函式 sq() 定義的時候, 它會動態結合函式 cube.fun() 的 引數 Zvar, Zvar 是函式 cube.fun() 之引數 (形式參數), 這是 R 的 詞彙作用域. 在 R 和 S-PLUS 裏面解析不同點在於 S-PLUS 先搜索總體變數 Zvar, 而 R 在 函式 cube.fun 使用 變數 Zvar 時, 首先尋找函式 cube.fun 環境建立的引數或變數 Zvar.

```
> ## lexical scope
> cube.fun <- function(Zvar) {
    sq.fun <- function() {Zvar*Zvar}
    Zvar*sq.fun()
  }
> cube.fun(Zvar = 3)
[1] 27
> ## lexical scope power function
> make.power <- function(n) {
             pow <- function(x) {x^n}
             pow
      }
> cube <- make.power(3)
> square <- make.power(2)
> #
> cube(4)
[1] 64
> square(4)
[1] 16
```

12.3 回傳或輸出資料函式

自建函式常常需要回傳複雜資料物件, 供其它函式使用, 例如, 求解最大概似函數, 同時回傳參數估計與變異數矩陣, 可先將複雜資料物件個別成分建構成 列表物件 (list object), 函式回傳列表物件, 例如使用指令 list.name <- fun.name(arg) 回傳列表物件 list.name, 再利用 list$var 使用列表物件 list.name 內的個別

成分 (component) var.

```
> ## return as list
> f.hm <- function(num){
      set.seed(100)
      x.vec <- rexp(num, rate = 0.001)
      y.mat <- matrix(c(rexp(num*5, rate = 0.01)), nrow = num)
      list(x.vec = x.vec, y.mat = y.mat)
      }
> f.list <- f.hm(3)
> f.list$x.vec
[1] 924.2116 723.8372 104.6449
> f.list$y.mat
          [,1]       [,2]     [,3]     [,4]      [,5]
[1,] 309.73623   9.311719 19.43265 202.3192 38.058106
[2,]  62.48052 174.839077 52.51022 112.3247  7.162231
[3,] 117.44293  24.999295 33.80434 113.1048 42.160769
```

使用函式 cat() 與 print() 可將自建函式產生的結果傳送到螢幕視窗. 函式 cat() 也可將結果傳送到外部一個檔案.

```
> cat(... , file = "", sep = " ", fill = FALSE,
      labels = NULL, append = FALSE)
```

其中的主要引數分別爲:

- file: 儲存資料輸出檔案名.
- sep: 指定資料值的分隔.
- fill: 控制列印的寬度.

```
> ## 'fill' and label lines:
> paste(letters, 100*c(1:26))
 [1] "a 100"  "b 200"  "c 300"  "d 400"  "e 500"  "f 600"  "g 700"  "h 800"
 [9] "i 900"  "j 1000" "k 1100" "l 1200" "m 1300" "n 1400" "o 1500" "p 1600"
[17] "q 1700" "r 1800" "s 1900" "t 2000" "u 2100" "v 2200" "w 2300" "x 2400"
[25] "y 2500" "z 2600"
> paste0("{", 1:10, "}:")
 [1] "{1}:"  "{2}:"  "{3}:"  "{4}:"  "{5}:"  "{6}:"  "{7}:"  "{8}:"  "{9}:"
[10] "{10}:"
> cat(paste(letters, 100*c(1:26)), fill = TRUE,
     labels = paste0("{", 1:10, "}:"))
{1}: a 100 b 200 c 300 d 400 e 500 f 600 g 700 h 800 i 900 j 1000 k 1100
{2}: l 1200 m 1300 n 1400 o 1500 p 1600 q 1700 r 1800 s 1900 t 2000 u 2100
{3}: v 2200 w 2300 x 2400 y 2500 z 2600
```

函式 print() 爲一般的通用函式, 且列印結果會依賴 print() 的 引數 與 物件之類型 (class) 的不同而有所不同. 另外, 使用函式 write.table() 也可輸出資料, 但使用 write.table() 時, 最好同時設定輸出格式, 詳見輔助文件 help(write.table).

```
> ## cat() return
> f.hm <- function(num){
      set.seed(1)
      x.vec <- rexp(num, rate = 0.001)
      cat(x.vec, "\n")
      cat(x.vec, sep = ",", "\n")
      cat(x.vec, sep = "\t", fill = 50)
      cat(x.vec, sep = "\n")
      cat(x.vec, file = "C:/RData/xveccat.dat", sep = "\n")
      write.table(x.vec, file = "C:/RData/xvecwr.dat",
                  sep = ",", row.names = FALSE)
      x.df <- data.frame(xvec = x.vec, yvec = x.vec)
      write.table(x.df, file = "C:/RData/xvecdf.dat",
                  sep = ",", row.names = FALSE)
      }
> f.hm(5)
755.1818 1181.643 145.7067 139.7953 436.0686
755.1818,1181.643,145.7067,139.7953,436.0686,
755.1818        1181.643        145.7067        139.7953        436.0686
755.1818
1181.643
145.7067
139.7953
436.0686
> #
> ## check new files in "C:/RData/"
```

12.4 輸入資料函式: readline()

若自建函式需使用者由螢幕視窗輸入資料, 可以使用函式 readline(), 在任一自建函式中, 容許使用者由螢幕視窗輸入資料. 除非輸入少數資料, 否則較少使用由螢幕視窗輸入資料.

```
> ## readline()
> f.hm.read <- function(){
      n.chr <- readline("Enter a integer: ")
      num <- as.numeric(n.chr) # readline treat as characters
      set.seed(num)
      x.vec <- rexp(num, rate = 0.001)
```

```
      cat(x.vec, "\n")
      }
> f.hm.read()
Enter a integer: 3
1730.627 615.0328 1232.917
```

12.5 函式工作環境

函式 在 R 程式語言的正式參照是 closure 封閉, lexical closure (詞彙封閉) 或 function closures (函式封閉), 是由函式和與其相關的 R 參照環境 (environment) 組合而成的物件, 是指函式內產生的局部變數或自由變數是隨著函式而一同存在, 參照環境 (environment) 的形成, 是由所函式創建的物件或變數組成. 在 R command prompt > (提示符號) 下建立的函式, 在參照環境爲 top level (最高階層), 呈現爲 R_GlobalEnv (<environment: R_GlobalEnv>, 但容易與 R .GlobalEnv 混淆. ls() 顯示在 R 環境內的物件, 若無指定參照環境的階層, ls() 顯示 top level 內的物件.

```
> ## function environment
> rm(list = ls()) # remove all objects
> w <- 5
> f <- function(x) {
    d <- 6
    h <- function() {
      return(d*(w+x))
    }
    return(h())
  }
> environment(f)
<environment: R_GlobalEnv>
> ls()
[1] "f" "w"
> ls.str()
f : function (x)
w :  num 5
```

在 C 語言函式內不能再定義另一個函式, 在 R 語言函式內可以再定義另一個函式, 例如, 上述 函式 f() 內定義另一個函式 h() 與 物件 d. 函式 h() 與 物件 d 都是

函式 f() 內的局部物件 (local object). R 的 階層作用域 (hierarchical scope) 顯示物件 w 相對 函式 f() 爲 總體變數, 也是相對 函式 h() 爲 總體變數, 物件 d 相對 函式 h() 爲 總體變數, 物件 x 相對 函式 h() 爲 局部變數.

有好幾種方法可以制定 R 環境選項, 例如, 在每個工作目錄下, 可以有 R 特有的一個初始化文件, 修改初始化檔案, 還有就是利用函式 .First 和 .Last 制定或改變 R 的工作環境選項. 初始化檔案的路徑可以由 環境變數 (environmental variable), R_PROFILE, 設定, 如果該變數沒有設定, 則是自動使用在 R 安裝目錄下面的子目錄 etc, 使用其中的檔案 Rprofile.site. 這個檔案包括每次執行 R 時一些自動運行的指令與工作環境選項設定.

若使用者想在不同工作目錄下, 有不同的工作環境選項設定, 可以在工作目錄下放置或編輯 第二個制定檔案 .Rprofile, 這個檔案 .Rprofile 可以放在任何目錄之下, 如果 R 在該目錄下面工作或執行, 這個檔案就會被載入. .Rprofile 這個檔案允許使用者定制不同的工作環境選項, 允許在不同的工作目錄下, 設定不同的起始自動運行指令的指令與工作環境選項設定. 如果在起始工作目錄下, 沒有 .Rprofile 這個檔案, R 會在主目錄下, 搜尋 .Rprofile 檔案 並且使用.

在這兩個檔案 Rprofile.site, .Rprofile 或在檔案 .RData 中, 任何呼叫 .First() 的函式, 都有特定的狀態. 它會在 R 視窗開啟時自動執行並且初始化工作環境. R 在這些檔案的執行順序是 Rprofile.site, .Rprofile, .RData, 然後是 .First(). 執行排序在後面文件檔案中之定義, 會蓋掉排序在前面檔中的定義. 在統計分析中進行統計模擬, 常常需要改變列印之有效數字, 避免列印成科學表示符號數字, 增加記憶體使用限制, 載入常用統計套件等特殊需求, 都可在 .First() 更改. 例如, 下面的定義將 提示符號 (prompt symbol) 改爲 $, 以及設置其他常用的環境選項設定.

```
> .First <- function() {
      rm(list = ls(all = TRUE))
      memory.limit(size = 2000)
      ## custom numbers and printout
      options(object.size = 10000000000, digits = 6, scipen = 100, length = 999,
          memory = 3200483647, contrasts = c("contr.treatment", "contr.poly"))
```

```
7       library(MASS)              # use a package
8       setwd("C:/RData/")         # change working directory
9        }
```