歡迎閱讀
R 錦囊妙計第二版

R 是一個用於統計、圖形和統計程式設計的強大工具。每天有成千上萬的人使用它進行重要的統計分析。它也是一個自由、開源的系統，是許多聰明、勤奮工作者集體智慧結晶。目前 R 已有超過 10,000 個可用的附加套件，R 是目前所有商業統計軟體的強勁競爭對手。

但剛開始接觸 R 可能有時也會令人感到無所適從。因為，即使是簡單的任務，R 並未直觀地呈現如何完成任務。一旦您跨越學習門檻，就能輕易地完成簡單的任務。不過，學習“如何”使用 R 的過程，有時可能會讓人抓狂。

這本書內容盡是引導讀者如何使用（how-to）R 的實務錦囊，每一個錦囊能解決一個特定的問題。錦囊也包括對解決方案的快速介紹，然後進行解決方案的細節討論，讓您瞭解它是如何運作。我們知道這些錦囊是實用的，而且在實務上是可行的，因為我們自己也經常使用它們。

錦囊的範圍很廣。從基本任務開始，然後介紹資料的輸入和輸出、一般統計、繪圖和線性回歸分析等。任何與 R 有關的重要工作都涉及以上大部分或全部領域。

如果您是初學者，那麼這本書會讓您起步更快。如果您已是一個中級使用者的程度，這本書將有助於擴展您的視野和強化您的記憶（“例如：我要如何再做一次 Kolmogorvo-Smirnov 檢定？”）。

這本書不是關於 R 的指導手冊，儘管您會透過學習錦囊學到一些東西。這也不是一本 R 參考手冊，但它確實包含了很多有用的資訊。它不是一本關於 R 語言程式設計的書，儘管許多錦囊都能應用在撰寫實務上的 R 腳本。

最後，這本書不是統計學入門書。本書中許多錦囊都假設您熟悉基本的統計程序，並且只想知道 R 中如何完成這些統計運算功能。

實務錦囊

本書中大多數錦囊都使用一個或兩個 R 函式來解決特定的問題。需要提醒各位的是，我們不詳細描述函式的所有功能；相反地，我們只提供足夠解決指定問題的描述。幾乎每個提及的函式都有超出書中描述之外的更多功能，而且，其中一些功能還很驚人。我們強烈建議您閱讀函式的說明頁面，您將會學到一些有價值的內容。

每個錦囊的設計，都是為讀者提供一種解決特定問題的方法。當然，每個問題都可能有幾種合理的解決方案。如果我們知道同一個問題存在多個解法時，我們通常選擇最簡單的那一個。對於任何您想進行的任務，您都可能自行發現更多備選解決方案。此時，請記得這是一本錦囊，不是經典大全。

特別是，R 有成千上萬個可下載的附加套件，其中許多實現了可互相替代的演算法和統計方法。本書專注於基礎發行版本的核心功能和幾個重要的套件組成的 *tidyverse*。

tidyverse 的發起者和核心維護者 Hadley Wickham 用最簡潔的方法定義了 tidyverse（*http://bit.ly/2Rh2tq1*）：

> tidyverse 是一組和諧工作的套件，因為它們共用了資料表示方法和 API 設計。`tidyverse` 套件被設計成只需要一個命令，就能輕鬆安裝和載入核心套件。若想瞭解 tidyverse 中的所有套件以及它們可如何被組合使用，請見 *R for Data Science*（*http://r4ds.had.co.nz*）。

專業術語說明

每個錦囊的目標都是如何快速地解決一個問題，為避免冗長乏味的論述，我們有時會用雖然正確但不精準的術語簡化描述。以**泛型函式**（*generic function*）為例，我們將 `print(x)` 和 `plot(x)` 稱為泛型函式，因為它們能適當地處理多種類型的 *x*，所以函式適用於多種型態的 *x*。電腦科學家們會對我們的術語表達頗有微詞，因為嚴格地來說，這

些不僅僅是“函式”；它們同時也是動態調度的多型方法。但是，如果我們仔細分析每一個這樣的技術細節，關鍵的解決方案將被淹沒在技術細節中。所以我們選擇稱它們為函式，以確保易讀性。

另一個例子取自統計學，是關於統計假設檢定在語義上複雜性。堅持使用嚴格的機率論術語將會模糊檢定於實務應用的焦點；因此，在描述每一個統計檢定時，我們會使用更加口語化的語言。請參閱第 9 章的介紹，瞭解更多關於錦囊中呈現假設檢定的方式。

我們的目標是淺顯易讀，而不是形式化的寫作，讓更多的讀者瞭解 R 的強大功能。我們的術語偶爾是非正式的，希望各領域的專家們能夠諒解。

軟體和平台說明

雖然 R 的基礎發行版本會頻繁且依預排計劃一直更新，但是語言定義和核心實現卻是相當穩定的。這本書中的錦囊應該與任何最近發佈的基礎發行版本都能相容。

有些錦囊在不同平台上會有使用上的差異，我們已經仔細地標注了這些差異。這些有差異的錦囊主要是處理軟體問題錦囊，比如安裝和設定。據我們所知，所有其他的錦囊都同時適用於 R 的所有三個主要平台：Windows、macOS 和 Linux/Unix。

其他資源

如果您想要更深入地挖掘 R 相關資訊，這裡有一些進一步閱讀的建議：

網路

所有關於 R 的內容都源自於 R 專案網站（*http://www.r-project.org*）。在那裡，您可以下載關於 R 的資源，例如平台、附加元件套件、文件、原始程式碼以及許多其他資源。

除了 R 專案網站之外，我們建議使用 R 專用的搜尋引擎，比如 Sasha Goodman 建立的 RSeek（*http://rseek.org*）。當然，您也可以使用一般的搜尋引擎，比如 Google，只是“R”關鍵字搜尋會帶出太多無關的東西。有關搜尋網路的更多資訊，請參見錦囊 1.11。

閱讀部落格是學習 R 的好方法，也是跟上最新發展的好方法。令人驚訝的是，這樣的部落格有很多，所以我們推薦以下兩個部落格：Tal Galili 建立的 R-bloggers（*http://www.r-bloggers.com/*）和 PlanetR（*http://planetr.stderr.org/*）。透過訂閱他們的 RSS feed，您將收到來自許多網站有趣又實用的文章通知。

第二章

基礎知識

本章的錦囊主題介於解決問題和教學指南之間。是的，它們能解決常見的問題，同時解決方案也展示了大多數 R 程式碼（本書中的程式碼也包括在其中）中使用的常見技術和慣用術語。如果您是 R 的新手，我們建議您通讀這一章來熟悉這些用語。

2.1 在螢幕上列印東西

問題

要顯示變數或運算式的值。

解決方案

如果您單純在命令提示符中輸入變數名或運算式，R 將列印它的值。使用 `print` 函式來列印任何物件的值。使用 `cat` 函式生成客製化格式的輸出。

討論

讓 R 列印東西很容易——只要在命令提示符處輸入：

```
pi
#> [1] 3.14
sqrt(2)
#> [1] 1.41
```

當您輸入如下方的運算式時，R 會對運算式進行運算，然後在背後呼叫 print 函式。所以前面的例子和這個是一樣的：

```
print(pi)
#> [1] 3.14
print(sqrt(2))
#> [1] 1.41
```

print 的美妙之處在於，它知道如何格式化列印任何 R 值，包括結構化的值，如矩陣（matrix）和串列（list）：

```
print(matrix(c(1, 2, 3, 4), 2, 2))
#>      [,1] [,2]
#> [1,]    1    3
#> [2,]    2    4
print(list("a", "b", "c"))
#> [[1]]
#> [1] "a"
#>
#> [[2]]
#> [1] "b"
#>
#> [[3]]
#> [1] "c"
```

這很有用，因為您總是能使用 print 函式查看您的資料，即使對於複雜的資料結構，也不需要編寫特殊的列印邏輯。

但 print 函式有一個明顯的限制：它一次只列印一個物件。試圖 print 多個物件，將會得到這個令人一怔的錯誤訊息：

```
print("The zero occurs at", 2 * pi, "radians.")
#> Error in print.default("The zero occurs at", 2 * pi, "radians."):
#>     invalid 'quote' argument
```

print 多個物件的唯一方法是一次列印一個，但像這樣的寫法又不會是您想要的：

```
print("The zero occurs at")
#> [1] "The zero occurs at"
print(2 * pi)
#> [1] 6.28
print("radians")
#> [1] "radians"
```

cat 函式是 print 的另一種選擇，它允許您將多個物件結合成一個連續的輸出：

```
cat("The zero occurs at", 2 * pi, "radians.", "\n")
#> The zero occurs at 6.28 radians.
```

注意，cat 預設情況下會在每個元件的輸出間放置一個空格，您必須使用分行符號（\n）來換行。

cat 函式也可以列印簡單的 vector：

```
fib <- c(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
cat("The first few Fibonacci numbers are:", fib, "...\n")
#> The first few Fibonacci numbers are: 0 1 1 2 3 5 8 13 21 34 ...
```

使用 cat 可以對輸出進行更多的控制，在它 R Script 中生成給其他人使用的輸出時特別有用。然而，一個嚴重的限制是它不能印出複合資料結構，比如矩陣和串列。試著 cat 它們只會產生另一個錯誤訊息：

```
cat(list("a", "b", "c"))
#> Error in cat(list("a", "b", "c")): argument 1 (type 'list') cannot
#>     be handled by 'cat'
```

參見

控制輸出格式見錦囊 4.2。

2.2 設定變數值

問題

您希望將值保存在變數中。

解決方案

使用運算子（<-），而且不需要先宣告變數：

```
x <- 3
```

討論

只把 R 當成計算機般使用無法滿足您的需求，很快您就會想要定義變數並在其中保存值。這可以減少打字，節省時間，並使您的工作更有效率。

不需要在 R 中宣告或建立變數，只需為名稱做給值動作，R 就會建立變數：

```
x <- 3
y <- 4
z <- sqrt(x^2 + y^2)
print(z)
#> [1] 5
```

注意，賦值運算子由一個小於字元（<）和一個連字號（-）組成，它們之間沒有空格。

當您像這樣在命令提示符號中定義一個變數時，該變數將被保存在您的工作區中。工作區保存在電腦的主記憶體中，也可以保存到磁碟上。這個變數定義將保留在工作區中，直到您刪除它。

R 是一個**動態型別語言**（*dynamically typed language*），這代表我們可以隨意改變變數的資料類型。我們可以將 x 設定為數值變數，就像剛才顯示的那樣，然後馬上用字串值覆蓋它。R 具有這樣的彈性：

```
x <- 3
print(x)
#> [1] 3

x <- c("fee", "fie", "foe", "fum")
print(x)
#> [1] "fee" "fie" "foe" "fum"
```

有時候，在一些 R 函式中，您會看到特別不一樣的賦值運算子：

```
x <<- 3
```

這賦值運算子會強制賦值給全域變數而不是區域變數。但是，這個有點超出了本討論的範圍。

本著充分揭露資訊的精神，我們將說明 R 另外兩種形式的賦值述句。一個等號（=）可以當成賦值運算子使用。右向設定運算子（->）可用於任何可以使用左向設定運算子（<-）的地方（但參數是相反的）：

```
foo <- 3
print(foo)
#> [1] 3

5 -> fum
print(fum)
#> [1] 5
```

我們建議您避免使用這兩種賦值述句，因等號賦值很容易與等號判斷混淆。右向賦值在某些情況可能很有用，但對於不習慣看到它的人來說，可能會感到困惑。

參見

參見錦囊 2.4、2.14 和 3.3，也可參見 `assign` 函式的說明頁面。

2.3 列出變數

問題

您想知道工作區中定義了哪些變數和函式。

解決方案

請使用 `ls` 函式，使用 `ls.str` 取得關於每個變數的更多細節。您還可以在 RStudio 的 Environment 窗格中看到您的變數和函式，如圖 2-1 所示。

討論

`ls` 函式顯示工作區中物件的名稱：

```
x <- 10
y <- 50
z <- c("three", "blind", "mice")
f <- function(n, p) sqrt(p * (1 - p) / n)
ls()
#> [1] "f" "x" "y" "z"
```

注意，`ls` 回傳一個字串 vector，其中每個字串都是一個變數或函式的名稱。當您的工作區為空時，`ls` 回傳一個空 vector，不過，以下這個情況將產生令人困惑的輸出：

```
ls()
#> character(0)
```

這是 R 表示 `ls` 回傳一個零長度字串 vector 的奇特方式；也就是說，它回傳一個空 vector，因為工作區中沒有定義任何東西。

如果您想要的不僅僅是列出所有變數名稱，請嘗試使用 `ls.str`；因為它還會告訴您關於每個變數的一些資訊：

```
x <- 10
y <- 50
z <- c("three", "blind", "mice")
f <- function(n, p) sqrt(p * (1 - p) / n)
ls.str()
#> f : function (n, p)
#> x :  num 10
#> y :  num 50
#> z :  chr [1:3] "three" "blind" "mice"
```

這個函式被稱為 `ls.str`，是因為它先列出了您的變數，並對這些變數套用 `str` 函式，於是就可以顯示出它們的結構（請參閱錦囊 12.13）。

通常，`ls` 不回傳任何以點（`.`）開頭的名稱，這些以點開頭的名稱被認為是使用者通常不感興趣的變數，所以是被隱藏的（這和 Unix 的慣例一樣，不列出名稱以點開頭的檔案）。透過設定 `all.names` 參數為 `TRUE`，可以強制 `ls` 列出所有內容：

```
ls()
#> [1] "f" "x" "y" "z"
ls(all.names = TRUE)
#> [1] ".Random.seed" "f"            "x"            "y"
#> [5] "z"
```

RStudio 中的 Environment 窗格也會隱藏名稱以點開頭的物件。

參見

刪除變數見錦囊 2.4，檢查變數見錦囊 12.13。

2.4 刪除變數

問題

您希望從工作區中刪除不需要的變數或函式，或者刪除工作區中全部內容。

解決方案

使用 `rm` 函式。

討論

在使用 R 的過程中，您的工作空間可能很快就會變得雜亂。`rm` 函式能從工作區永久刪除一個或多個物件：

```
x <- 2 * pi
x
#> [1] 6.28
rm(x)
x
#> Error in eval(expr, envir, enclos): object 'x' not found
```

請注意，這個指令執行後無法「還原」；也就是說，一旦變數被刪除後，它就沒有了。

若有需要，您可以同時刪除多個變數：

```
rm(x, y, z)
```

您甚至可以一次刪除整個工作區中的所有內容，`rm` 函式有一個 `list` 引數，用來設定要刪除的變數名稱組成的 vector。還記得前面介紹過，`ls` 函式回傳一個變數名稱 vector；因此，您可以結合 `rm` 和 `ls` 來刪除所有內容：

```
ls()
#> [1] "f" "x" "y" "z"
rm(list = ls())
ls()
#> character(0)
```

刪除變數另外一個方法是，按下 RStudio 中 Environment 窗格頂部的掃把圖示，如圖 2-1 所示。

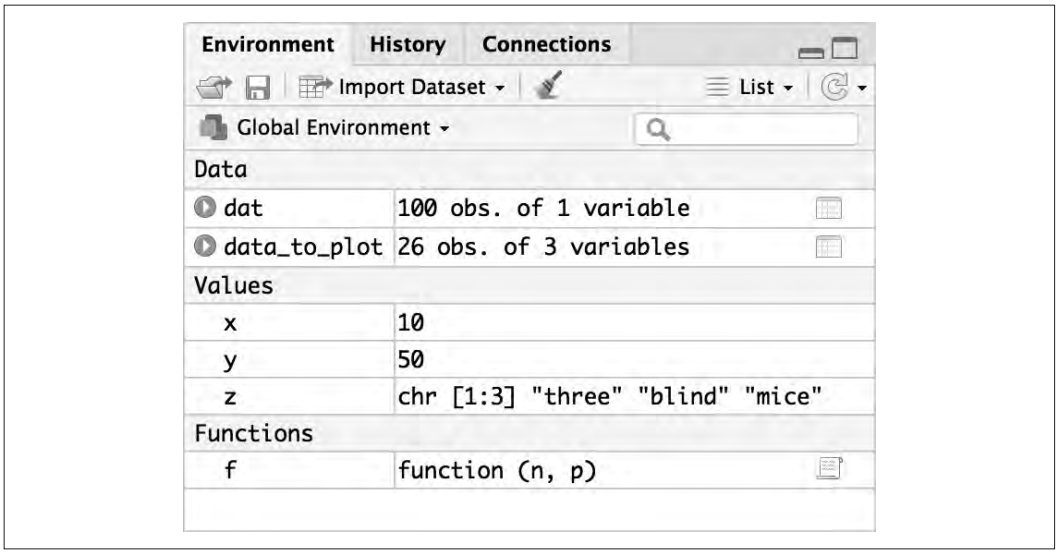

圖 2-1　RStudio 中 Environment 窗格

永遠不要將 `rm(list=ls())` 放入與他人共用的程式碼中，例如放入函式庫函式或發送到電子郵件串列或 Stack Overflow 的範例程式碼。刪除別人工作空間中的所有變數不僅不禮貌，而且會讓您非常不受歡迎。

參見

請參考錦囊 2.3。

2.5 建立 vector

問題

您想要建立一個 vector。

解決方案

使用 `c(...)` 運算子從給定值建構一個 vector。

討論

vector 是 R 的一個核心元件，而不僅僅是一個資料結構而已。vector 可以包含數字、字串或邏輯值，但不能混合使用。

`c(...)` 運算子是用來建立一個由簡單元素組成的 vector。

```
c(1, 1, 2, 3, 5, 8, 13, 21)
#> [1]  1  1  2  3  5  8 13 21
c(1 * pi, 2 * pi, 3 * pi, 4 * pi)
#> [1]  3.14  6.28  9.42 12.57
c("My", "twitter", "handle", "is", "@cmastication")
#> [1] "My"            "twitter"       "handle"        "is"
#> [5] "@cmastication"
c(TRUE, TRUE, FALSE, TRUE)
#> [1]  TRUE  TRUE FALSE  TRUE
```

如果指定給 `c(...)` 的引數是多個 vector，那麼 `c(...)` 運算子會將引數 vector 壓扁，並且合併成為一個 vector：

```
v1 <- c(1, 2, 3)
v2 <- c(4, 5, 6)
c(v1, v2)
#> [1] 1 2 3 4 5 6
```

vector 中不能混和不同資料型態，例如混合數值和字串等資料類型。如果您用混合元素去建立一個 vector，R 會為您去試著轉換資料型態：

```
v1 <- c(1, 2, 3)
v3 <- c("A", "B", "C")
c(v1, v3)
#> [1] "1" "2" "3" "A" "B" "C"
```

這一段程式碼中，我們嘗試用數值和字串建立一個 vector。在建立 vector 之前，R 將所有數值轉換為字串，從而使資料元素相容。注意，R 做這個動作時不會輸出警告或抱怨訊息。

從技術上講，只有當兩個資料元素具有相同的**模式**（*mode*）時，它們才能在一個 vector 中共存。`3.1415` 的模式為 `numeric`，`"foo"` 的模式為 `character`：

```
mode(3.1415)
#> [1] "numeric"
mode("foo")
#> [1] "character"
```

這些模式是不相容的，R 將 3.1415 轉換為 character 模式，使其與 "foo" 相容。

```
c(3.1415, "foo")
#> [1] "3.1415" "foo"
mode(c(3.1415, "foo"))
#> [1] "character"
```

c 是一個泛型運算子，這代表它可以處理許多種資料類型，而不僅僅是 vector。但是，它可能無法完全符合您的期望，所以請檢查它的行為後，才將其應用於其他資料類型和物件。

參見

有關 vector 和其他資料結構的更多資訊，請參見第 5 章的介紹。

2.6 計算基本統計量

問題

您希望計算基本統計量，如：平均數、中位數、標準差、變異數、相關係數或共變異數。

解決方案

使用下列函式進行計算，其中 x 和 y 是 vector：

- `mean(x)`
- `median(x)`
- `sd(x)`
- `var(x)`
- `cor(x, y)`
- `cov(x, y)`

討論

第一次使用 R 時，您很有可能一開始就打開文件並搜尋名為“Procedures for Calculating Standard Deviation”（計算標準差的流程）的資料，所以這樣一個重要的話題似乎需要用整整一章來說明才是。

但，其實沒那麼複雜。

標準差等基本統計量是用簡單的函式來計算的。通常，函式引數是一個數值 vector，函式回傳計算得到的統計量：

```
x <- c(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
mean(x)
#> [1] 8.8
median(x)
#> [1] 4
sd(x)
#> [1] 11
var(x)
#> [1] 122
```

`sd` 函式計算樣本標準差，`var` 計算樣本變異數。

`cor` 和 `cov` 函式可以分別計算兩個 vector 之間的相關係數和共變異數：

```
x <- c(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
y <- log(x + 1)
cor(x, y)
#> [1] 0.907
cov(x, y)
#> [1] 11.5
```

這些函式都對引數中不可用的值（`NA`）很挑剔，只要 vector 引數中有一個 `NA` 值，也會導致這些函式回傳 `NA`，甚至完全停止，並產生一個模糊的錯誤：

```
x <- c(0, 1, 1, 2, 3, NA)
mean(x)
#> [1] NA
sd(x)
#> [1] NA
```

R 這麼謹慎是令人討厭，但這是一個適當的態度。此時，您必須仔細考慮您的情況，在您的資料中的 `NA` 是否會使統計量無效？如果是，那麼 R 是對的。如果不是，可以透過設定 `na.rm=TRUE`，告訴 R 忽略 `NA` 值：

```
x <- c(0, 1, 1, 2, 3, NA)
sd(x, na.rm = TRUE)
#> [1] 1.14
```

在較老版本的 R 中，mean 和 sd 在處理資料幀上是很聰明的，它們可以理解資料幀的每一欄都是一個不同的變數，因此它們可分欄計算每一欄的統計值。但現在的版本已經不能了，因此，您可能在網路上或較老的書籍（如本書的第一版）中讀到舊版的行為。現在，若想將函式套用到資料幀的每一欄上，我們需要使用輔助函式。tidyverse 輔助函式家族中，要做這些事的輔助函式位於 purrr 套件中。與其他 tidyverse 套件一樣，當執行 library(tidyverse) 時，將載入 purrr 套件，在這個套件中我們要使用的函式是 map_dbl：

```
data(cars)

map_dbl(cars, mean)
#> speed  dist
#>  15.4  43.0
map_dbl(cars, sd)
#> speed  dist
#>  5.29 25.77
map_dbl(cars, median)
#> speed  dist
#>    15    36
```

注意，在本例中，mean 和 sd 各回傳兩個值，這些值都是由資料幀中對應欄產生（從技術上講，它們回傳的是由兩個元素組成的 vector，其 names 屬性即資料幀中的欄名）。

var 函式可以在映射函式或輔助函式的情況下，讀懂資料幀的結構。它會計算資料幀各欄之間的共變異數，回傳共變異數矩陣：

```
var(cars)
#>       speed dist
#> speed    28  110
#> dist    110  664
```

同樣，假設 x 代表資料幀或矩陣，則使用 cor(x) 會回傳相關係數矩陣，cov(x) 回傳共變異數矩陣：

```
cor(cars)
#>       speed  dist
#> speed 1.000 0.807
#> dist  0.807 1.000
cov(cars)
```

```
#>       speed dist
#> speed    28  110
#> dist    110  664
```

參見

參見錦囊 2.14、錦囊 5.27 和錦囊 9.17。

2.7 建立數列

問題

您想要建立一個數值數列。

解決方案

使用一個 *n:m* 運算式建立簡單數列 *n*、*n*+1、*n*+2、…、*m*：

```
1:5
#> [1] 1 2 3 4 5
```

使用 seq 函式，可產生數字間隔不為 1 的數列：

```
seq(from = 1, to = 5, by = 2)
#> [1] 1 3 5
```

使用 rep 函式，可以建立由重複數值構成的數列：

```
rep(1, times = 5)
#> [1] 1 1 1 1 1
```

討論

冒號運算子（*n:m*）建立包含 *n*、*n*+1、*n*+2、…、*m* 的 vector 序列：

```
0:9
#>  [1] 0 1 2 3 4 5 6 7 8 9
10:19
#>  [1] 10 11 12 13 14 15 16 17 18 19
9:0
#>  [1] 9 8 7 6 5 4 3 2 1 0
```

R 巧妙地處理最後一個運算式（9:0）。因為 9 比 0 大，所以它會以遞減的順序產生數列。您也可以直接將冒號運算式合併管道（pipe）使用，一起將資料傳遞給另一個函式：

```
10:20 %>% mean()
```

冒號運算子適用於產生增量為 1 的數列，但 seq 函式也構建序列，而且支援可選的第三個引數，這個引數用於指定增量值：

```
seq(from = 0, to = 20)
#>  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
seq(from = 0, to = 20, by = 2)
#>  [1]  0  2  4  6  8 10 12 14 16 18 20
seq(from = 0, to = 20, by = 5)
#> [1]  0  5 10 15 20
```

或者，您可以指定輸出數列的長度，然後 R 將自動計算增量：

```
seq(from = 0, to = 20, length.out = 5)
#> [1]  0  5 10 15 20
seq(from = 0, to = 100, length.out = 5)
#> [1]   0  25  50  75 100
```

增量不需要是整數；R 也可以建立以分數遞增的數列：

```
seq(from = 1.0, to = 2.0, length.out = 5)
#> [1] 1.00 1.25 1.50 1.75 2.00
```

特殊情況下，若需要建立內含重複值的“數列”，應該使用 rep 函式，這個函式會重複它的第一個參數：

```
rep(pi, times = 5)
#> [1] 3.14 3.14 3.14 3.14 3.14
```

參見

建立由 Date 物件組成的數列，請參閱錦囊 7.13。

2.8 比較 vector

問題

您想比較兩個 vector，或者您想比較整個 vector 和一個常量。

解決方案

比較運算子（==、!=、<、>、<=、>=）可以執行兩個 vector 中各元素的比較。它們還可以將 vector 的元素與常量進行比較，產出的結果是邏輯值構成的 vector，其中每個值都是一個元素比較的結果。

討論

R 有兩個邏輯值，TRUE 和 FALSE。在其他程式設計語言中，這些值通常稱為**布林**（*Boolean*）值。

比較運算子的功能是比較兩個值，根據比較結果回傳 TRUE 或 FALSE：

```
a <- 3
a == pi # 測試是否相等
#> [1] FALSE
a != pi # 測試是否不相等
#> [1] TRUE
a < pi
#> [1] TRUE
a > pi
#> [1] FALSE
a <= pi
#> [1] TRUE
a >= pi
#> [1] FALSE
```

藉由一次比較整個 vector，您可體驗 R 的厲害之處。R 將執行元素對元素的比較，並回傳一個邏輯值 vector，其中每個值都是一個元素比較的結果：

```
v <- c(3, pi, 4)
w <- c(pi, pi, pi)
v == w # 比較兩個 3 維 vector
#> [1] FALSE  TRUE FALSE
v != w
#> [1]  TRUE FALSE  TRUE
v < w
#> [1]  TRUE FALSE FALSE
v <= w
#> [1]  TRUE  TRUE FALSE
v > w
#> [1] FALSE FALSE  TRUE
v >= w
#> [1] FALSE  TRUE  TRUE
```

您還可以將 vector 與單個常量進行比較，在這種情況下，R 將複製該常量，使其擴展到 vector 的長度，然後再執行元素的比較。前面的例子可以這樣簡化：

```
v <- c(3, pi, 4)
v == pi # 比較一個擁有三個元素的 vector 與一個數值常量
#> [1] FALSE  TRUE FALSE
v != pi
#> [1]  TRUE FALSE  TRUE
```

這是錦囊 5.3 裡討論的循環規則中的一個應用。

在比較兩個 vector 之後，通常您會想知道比較結果中是否有**任何一個**為真，或者是否**所有**結果皆為真。`any` 和 `all` 函式此時派上用場，它們兩個都會去檢查由邏輯值組成的 vector，如果 vector 中的任何元素為 `TRUE`，則 `any` 函式回傳 `TRUE`。如果 vector 中的所有元素都為 `TRUE`，則 `all` 函式回傳 `TRUE`：

```
v <- c(3, pi, 4)
any(v == pi) # 如果 v 中任何元素與 pi 中的相同，則回傳 TRUE
#> [1] TRUE
all(v == 0) # 若 v 中所有元素為 0，則回傳 TRUE
#> [1] FALSE
```

參見

請參考錦囊 2.9。

2.9 選擇 vector 元素

問題

您想要從一個 vector 中取得一個或多個元素。

解決方案

請選擇適合的索引（indexing）技術：

- 使用中括號根據 vector 元素的位置選擇 vector 元素，例如 `v[3]` 代表 `v` vector 中的第三個元素。
- 使用負索引來排除不要的元素。

- 使用索引組成的 vector 來選擇多個元素值。
- 使用邏輯 vector，根據條件選擇元素。
- 使用名稱取得已命名元素。

討論

從 vector 中選擇元素是 R 的另一個強大特性，基本的元素選取和其他許多程式設計語言一樣處理——使用中括號和一個簡單的索引：

```
fib <- c(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
fib
#>  [1]  0  1  1  2  3  5  8 13 21 34
fib[1]
#> [1] 0
fib[2]
#> [1] 1
fib[3]
#> [1] 1
fib[4]
#> [1] 2
fib[5]
#> [1] 3
```

注意，第一個元素的索引為 1，而不是像其他一些程式設計語言索引從 0 開始。

vector 索引有一個很酷的功能，您可以一次選擇多個元素。索引本身可以是一個 vector，該索引 vector 的每個元素可從資料 vector 中取得一個元素：

```
fib[1:3] # 選取元素 1 到 3
#> [1] 0 1 1
fib[4:9] # 選取元素 4 到 9
#> [1]  2  3  5  8 13 21
```

`1:3` 的索引表示選擇元素 1、2 和 3，如上方程式執行結果所示。然而，索引 vector 不一定是一個簡單的序列。您可以選擇資料 vector 中的任何元素——就像在下方範例中，它選擇元素 1、2、4 和 8：

```
fib[c(1, 2, 4, 8)]
#> [1]  0  1  2 13
```

R 會將負索引理解為**排除**某些元素值。例如，`-1` 的索引表示排除第一個值並回傳所有其他值：

```
fib[-1] # 忽略第一個元素
#> [1]  1  1  2  3  5  8 13 21 34
```

您還可以進一步應用這個方法，搭配索引 vector 來排除整個段負索引元素：

```
fib[1:3] # 和之前一樣
#> [1] 0 1 1
fib[-(1:3)] # 將索引加上負號，改為排除元素
#> [1]  2  3  5  8 13 21 34
```

另外還有一種索引技術，是使用邏輯 vector 從資料 vector 中選取元素。當邏輯 vector 的元素為 TRUE 時，即選取相對位置的元素：

```
fib < 10 # 當 fib 小於 10 時，這個 vector 中的值為 True
#>  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
fib[fib < 10] # 使用 vector 選取小於 10 的元素
#> [1] 0 1 1 2 3 5 8
fib %% 2 == 0 # 當 fib 是偶數時，這個 vector 中的值為 True
#>  [1]  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE
fib[fib %% 2 == 0] # 使用 vector 選取偶數元素
#> [1]  0  2  8 34
```

一般來說，邏輯 vector 的長度應該與資料 vector 的長度相同，因此要取得或排除哪個元素就很清楚（如果長度不同，則需要用上錦囊 5.3 中討論的循環規則）。

vector 比較、邏輯運算子和 vector 索引可以組合使用，您可以用很少的 R 程式碼執行強大的選取工作。

例如，您可以選取所有大於中位數的元素：

```
v <- c(3, 6, 1, 9, 11, 16, 0, 3, 1, 45, 2, 8, 9, 6, -4)
v[ v > median(v)]
#> [1]  9 11 16 45  8  9
```

或選擇最高 5% 與最低 5% 的所有元素：

```
v[ (v < quantile(v, 0.05)) | (v > quantile(v, 0.95)) ]
#> [1] 45 -4
```

前面的範例中使用了 | 運算子，它在索引時表示“或者（or）”的意思。如果您需要“並且（and）”，可以使用 & 運算子。

您還可以選擇所有超過平均數 ±1 個標準差的元素：

```
v[ abs(v - mean(v)) > sd(v)]
#> [1] 45 -4
```

或選取所有不是 NA 也不是 NULL 的元素：

```
v <- c(1, 2, 3, NA, 5)
v[!is.na(v) & !is.null(v)]
#> [1] 1 2 3 5
```

最後一個索引功能是允許您按名稱選擇元素。它假設 vector 有一個 names 屬性，為每個元素定義一個名稱，您可準備一個字串 vector，為屬性定義名稱：

```
years <- c(1960, 1964, 1976, 1994)
names(years) <- c("Kennedy", "Johnson", "Carter", "Clinton")
years
#> Kennedy Johnson  Carter Clinton
#>    1960    1964    1976    1994
```

名稱定義好了以後，就可以透過名稱引用各個元素：

```
years["Carter"]
#> Carter
#>   1976
years["Clinton"]
#> Clinton
#>    1994
```

這種索引功能也可用名稱 vector 進行索引；R 回傳被指名的每個元素：

```
years[c("Carter", "Clinton")]
#>  Carter Clinton
#>    1976    1994
```

參見

有關循環規則的更多資訊，請參見錦囊 5.3。

2.10 執行 vector 數學運算

問題

您想一次對整個 vector 做數學運算。