第 2 章: 向量物件與基本操作

2: Vector and Basic Operator

R 是以物件導向爲主的程式語言, 在 R 中, 資料或運算指令以具有名稱的 物件 (object), 形式儲存, 資料物件可以是 向量 (vector), 矩陣 (matrix), 陣列 (array), 列表 (Lists), 或 資料框架 (data frames) 等. 在 R 中, 資料分析基本上是 產生資料物件, 對物件命名, 使用函式對物件運算操作. 透過指令, 很容易地對物件進行統計分析與統計繪圖.

2.1 物件命名

R 透過函式或指令, 很容易地對資料物件進行統計分析與統計繪圖. 須特別注意, 在 R 對物件或指令命名的英文大小寫是有差異, `s` 與 `S` 是不同的. 對物件命名時, 物件 名字 (name) 起始位置必須以 "文字" 或 "." (句點) 命名, 若物件 名字 以 "." 爲起始, 則 名字 的第二個位置需爲文字, 物件名字其餘位置, 以文字 (A-Z 或 a-z), 數字 (0-9), "/", ".", 或 "-", 等皆可. 中間不可有空格或 "_" (underscore).

R 也保留一些特定名字做爲特定的物件名字, 指令名字或常用函式名字使用, 例如, `c`, `s`, `C`, `T`, codeF 等, 這些名稱叫做 "保留 名字" (reserved names). 例如:

```
FALSE Inf NA NaN NULL TRUE
break else for function if in next repeat while
```

```
3 F T
4 c q s t C D I
5 diff mean pi range rank var
```

初學者對物件命名時, 應盡量避免定義一個物件名字, 與現有的物件同名, 所以命名時要避免重覆, 同時避免物件名字過短, 以免後來引起錯亂.

2.2 運算式語言 Expression Language

R 基本介面是一個互動式指令視窗, 指令可分成 2 種, 一爲 運算式 (expression), 例如, 1+2; 另一個爲 指派運算 或 賦值運算 (assignment). R 的 運算式 (expression) 如

```
1 > 4.3 + 2.7 + 8.6
2 > log(6.2)
3 > mean(3.5+7.6+9.8)
```

指派運算 或 賦值運算 例如,

```
1 > x <- 1+2
2 > x = 15-3
3 > y = log(x)
4 > z <- exp(y)
```

當一個 R 程式需要使用者輸入指令時, 它會顯示 指令提示符號 (prompt symbol), 指令提示符號通常是一個 “>” (大於符號). 當使用者輸入完整的 運算式, 則運算式指令輸入後的結果, R 會馬上顯示在指令下方. 當使用者輸入完整的 指派運算, R 同樣會做運算式, 並且把結果 (值) 傳給變數, 但結果不會自動顯示在 R 視窗螢幕上. 在 R 中可利用 `options(prompt = "R>")` 將指令提示符號 `>` 改成 `R>`.

指派運算符號 (assignment symbol) 通常是 “ `<-` ”, 一個小於符號和一個短線符號組成, 例如, `x <- 1+2`, 讀成 `x` “得到” $(1+2)$. 在 R 中可以如傳統的程式語言或統計軟體, 使用 “ `=` ” (等號) 爲 指派運算符號, 例如, `x = 1+2`, 但在 R 中, “ `=` ” (等號) 還有其他用途, 使用者可依個人習慣使用 “ `<-` ” 或 “ `=` ”, 但是多數 R 專業人士建議使用 “ `<-` ”.

要顯示賦值運算的結果, 可以輸入物件名, 使用函式 print, 或在賦值運算時, 前後加上小括號顯示賦值運算的結果.

```
> x <- 1
> x        # {\McQ\cH157}\z{\MaQ\cH112}\z{\MbQ\cH211}\z{\MaQ\cH75}\z{\MaQ\cH177}
[1] 1
> print(x) # {\MdQ\cH131}\z{\MbQ\cH31} print()
 [1] 1
> msg <- "hello"
msg     # {\McQ\cH157}\z{\MaQ\cH112}\z{\MbQ\cH211}\z{\MaQ\cH75}\z{\MaQ\cH177}
[1] "hello"
```

如果一條輸入的指令在第一行結束的時候, 在 R 語法上還不完整, 若用鍵盤上 <Enter> 按鍵時, 則 R 會給出另一個不同的提示符號, 通常是 “+” (加號), 且該提示符號 “+” 會出現在第二行, 和隨後的數行中, R 持續地等待使用者輸入指令. 當一指令在語法上是完整的時候, 使用鍵盤上 <Enter> 按鍵時, R 才會執行指令. 不同的完整指令要在同一行輸入時, 可用 ; (分號) 隔開, 或是另起一新輸入行輸入指令分別輸入不同的完整指令. 例如,

```
> ## {\MaQ\cH176}\z{\McQ\cH248}\z{\McQ\cH87}\z{\McQ\cH157}\z{\MaQ\cH112} {\MaQ\cH170}\z{\
    MbQ\cH224}\z{\MaQ\cH125}\z{\McQ\cH85} ;
> x <- 1 + 2; y <- 3 + 4
> ## {\MaQ\cH125}\z{\MaQ\cH129}\z{\McQ\cH157}\z{\MaQ\cH112}
> x <- 1 + 2
> y <- 3 + 4
```

數個指令也可以放入一組大括弧內, { }, 數個指令放在一起, 構成一個 複合運算式 (compound expression), 這部份在函式的章節會再進一步說明. 基本指派運算之指令符號見表 2.1.

在 R 中, 若要對任何指令, 物件, 程式語言加上 注釋 (commands), 則注釋從 # (井號) 開始, 到句子收尾之間的語句就是是注釋, 在 R 中, 注釋幾乎可以放在任何地方的任何一行之中. 習慣上, 整行的注釋使用雙井號作爲開始, ##, 運算式尾端注釋使用單井號開始 #.

```
> ## This is my R code
> log(pi)
> ## {\McQ\cH21}\z{\MaQ\cH192}\z{\McQ\cH108}\z{\McQ\cH18}
> 2 + 1 # {\McQ\cH108}\z{\McQ\cH18}\z{\MbQ\cH156}: two plus one
```

在 R 互動式窗 Console 中, 若要重複一個指令, 或是叫回之前輸入的指令, 可以用鍵盤上的 ↑ ("向上") 箭頭按鍵, 調出之前已經輸入的指令, 視窗中便可顯示之前的輸入指令, 再利用鍵盤上 <DEL> 按鍵更改成所要輸入的指令. 可以再次練習以下指令, 檢視 R 會傳結果.

```
> ## This is my R code
> x = 1 + 2  # one plus two
> x
> x + 4
> x - 1
```

R 是一種運算式語言 (expression language), 運算式由含有指令形式的物件 (object), 算數操作符號 (operators), 函式 (function) 組成. R 的最基本物件是向量, 可利用基本算數符號, 如 `+, -, *, /, %/%, %%, %*%, ^, = = , !, ! = , <, >, < = . > = , &, &&, |, ||, xor` 等, 或函式, 如 `sum(), mean(), sqrt(), log(), exp(), abs(), sin()` 等, 對向量做運算, 基本運算符號與函也可以對任何資料物件做運算.

```
> ## expression
> x.vec <- 3
> x.vec + 5
[1] 8
> y.vec <- 1:10
> x.vec + y.vec
[1]  4  5  6  7  8  9 10 11 12 13
> # {\MaQ\cH85}\z{\MbQ\cH224} \{ ... \} {\MaQ\cH248}\z{\MbQ\cH98}\z{\MaQ\cH95}\z{\MbQ\cH78
    }\z{\MaQ\cH73}\z{\MbQ\cH91}\z{\MaQ\cH202}\z{\McQ\cH248}\z{\McQ\cH150}\z{\MaQ\cH207}\z
    {\McQ\cH87}
> { x.vec <- 3
  x.vec + 5
  y.vec <- 1:10
  x.vec + y.vec
  }
[1]  4  5  6  7  8  9 10 11 12 13
```

表 2.1: R Console 指派運算之指令符號

符號	定義
?	尋求協助 (Help)
<-	左側指派 (Left assignment, binary)
->	右側指派 (Right assignment, binary)
$	顯示出列表的子集或子成分 (List subset, binary)
:	序列 (Sequence, binary)
:	模型方程式中變數之間的交互作用 (In model formulae: interaction)
	模型方程式中變數之間的配適 (Used for model formulae, either uniary or binary)

2.3 列印輸出函式: print() 與 cat()

函式 print() 可以用來列印 R 物件, 有時候, 可以用 cat() 列印輸出. 列印文字向量的時候, 採用雙引號 (有時根本不用引號). R 採用 C 語言 的轉義控制序列, 用 \ 表示轉義字元, 所以輸入 \\ 將會得到 \ 的輸出, 而想插入雙引號, ", 則要輸入 \". 其他有用的轉義字元, 例如: \n (換行), \t (跳位字元), 和 \b (倒退鍵) 等等.

```
> ## print() cat()
> x.vec <- c(1, 2)
> print(x.vec)
[1] 1 2
> cat("\\")
\
> cat("\"")
"
> cat(x.vec, "\\", x.vec, "\"", x.vec)
1 2 \ 1 2 " 1 2
> cat(x.vec, "\\", x.vec, "\"", "\n", x.vec)
1 2 \ 1 2 "
 1 2
> cat(x.vec, "\\", x.vec, "\"", "\t", x.vec)
1 2 \ 1 2 "      1 2
> cat(x.vec, "\\", x.vec, "\"", "\b", x.vec)
1 2 \ 1 2 " 1 2
```

2.4 物件結構 Data Structure

在 R 中產生資料和控制運算的實體稱爲 “物件”, 物件包含儲存資料的向量 (vector), 矩陣 (matrix), 陣列 (array), 列表 (Lists), 資料框架 (data frames) 或執行特定運算指令的函式 (function) 等.

R 物件的 資料結構 (data structure) 包含由資料的 維度 (dimensionality) 與組成元素的 模式 (mode) 決定. 維度包含 1-D, 2-D, or p-D 等, 而組成元素的 模式 爲所有元素都是同質性或異質性, 同質性是所有元素都是相同的基本模式, 參見表 2.2.

表 2.2: R 資料結構

維度 Dimensionality	同質性 Homogeneous	異質性 Heterogeneous
1-D	vector	list
2-D	matrix	data frame
p-D	array	

物件的 結構 (structure) 可簡單分成 原型 (atomic) 與 遞迴型 (recursive). 原型物件 (atomic object) 有向量 (vector), 矩陣 (matrix), 陣列 (array) 等, 組成元素爲同質性模式, 如 向量 (vector), 矩陣 (marix) 與 陣列 (array). R 的最基本物件是向量, 向量是由包含相同 模式 (mode) 的元素組成, 矩陣 與 陣列也由包含相同模式的元素組成, 同一向量, 矩陣 與 陣列 內的元素不可混合, 單一數值 (scalar), 可視爲僅具有單一元素的向量.

遞迴型物件 (recursive object) 有列表 (list), 資料框架 (data frame), 函式 (function) 等. 組成元素爲異質性模式, 遞迴型物件組成元素包含有複雜 (混合) 元素. 其他如, `"expression"`, `"name"`, `"symbol"` 等也是遞迴型物件. 使用函式指令 `mode()` 可以用來查看物件的模式.

2.5 向量

R 的最基本物件是向量, 向量 是指包含相同 模式 (mode) 的元素組成, 主要的 基本模式 (basic mode) 有

1. `"numeric"`, 數值型 (實數型), 含 `"single"` 單精準度型 與 `"double"` 倍精準度型.
2. `"integer"`, 整數向量 (有時需特別指定輸入成 1L, 2L, ...).
3. `"logical"`, 邏輯型 (true or false), 以 `TRUE` (`T`) 或 `FALSE` (`F`) 呈現, (也可以是 1 與 0 (分別代表 `T` 與 `F`).
4. `"complex"`, 複數型.
5. `"character"`, 文字型 或 字串型, 通常輸入時在文字兩側加上雙引號 (").

同一向量內的元素不可混合. 單一數值 (scalar), 可視爲僅具有單一元素的向量, 標準的向量是倍精準度 (double) 的數值向量 (numerical vector).

每一種科學領域, 每一種教科書, 每一本統計教科書, 每一種統計軟體等等, 都會對資料進行分類, 主要的依據是資料儲存與運算操作 R 對資料分成 `numeric` (數值), `integer` (整數), `logical` (邏輯), `complex` (複數), 與 `character`, (文字) 等, numeric 爲 實數 或 連續變數, character 約等同於爲 類別變數.

向量是具有相同基本類型的元素序列, 在 R 中, 純量或單一數值 (scalar) 也可看成是長度爲 1 的向量, 向量大體上相當於其他程式語言中的 1-維度數列, 但在 R 中向量並不具有 沒有維度 (no dimension), 例如, 在 R 中 `x.vec <- c(1, 2, 3)`, 可以視爲 1×3 的矩陣, 也可以視爲 3×1 的矩陣, 但是, 當 向量 `x.vec` 與 其它 向量/矩陣 進行運算時, 向量 `x.vec` 會受到與其進行運算的矩陣物件影響, 若任由 R 的內在設定, 則將會有意想不到的運算結果, 因此在進行線性代數相關的計算, 若要避免混淆與錯誤, 初學者可將數學的 1-維度 k-個元素的向量, 重新定義成 R 的 $1 \times k$ 的矩陣 或是 $k \times 1$ 的矩陣, 然後再進行線性代數相關的計算.

2.5.1 向量產生函式 算數操作 Arithmetic Operator c()

輸入簡單的向量資料, 可以用函式 `c()` 指令或函式 `assign()` 指令輸入資料. 函式 `c()` 爲 concatenate (連接), 將數值或文字連接成向量. 當 R 顯示向量物件, 若向量長度超過 R Console 螢幕設定出現的字元數時 `[number]` 出現在某特定元素左方時, 表示某特定元素在該向量物件的位置 (position), 例如, `[k]` 表示在該向量的第 k 個位置之元素, 例如以下指令中的 `[7] 12.0000000`, 表示在該向量的第 7 個位置之元素爲 12.

```
> ## c()
> ## numerical
> x.vec <- c(1/1, 1/2, 1/3, 1/4, 1/5)
> x.vec
[1] 1.0000000 0.5000000 0.3333333 0.2500000 0.2000000
> #
> ## integer
> x.vec <- c(1L, 2L, 3L)
> x.vec
[1] 1 2 3
> ## character
> flavors.vec <- c("chocolate", "vanilla", "strawberry") # character
> flavors.vec
[1] "chocolate"  "vanilla"    "strawberry"
> y.vec <- c("Hello", "What's your name?", "Your email?")
> y.vec
[1] "Hello"             "What's your name?" "Your email?"

> ## logical
> z.vec <- c(F, T, T, F, F)
> z.vec
[1] FALSE  TRUE  TRUE FALSE FALSE

> ## complex
> x.complex.vec <- c(8+3i, 9+0i, 2+4i)
> x.complex.vec
[1] 8+3i 9+0i 2+4i

> ## numerical
> x.vec <- c(1/1, 1/2, 1/3, 1/4, 1/5)
> y.vec <- c(1, 2, 3, 4, 5)
> z.vec <- c(x.vec, 11, 12, y.vec)
> z.vec
 [1]  1.0000000  0.5000000  0.3333333  0.2500000  0.2000000 11.0000000
 [7] 12.0000000  1.0000000  2.0000000  3.0000000  4.0000000  5.0000000
```

2.5.2 向量產生函式: vector()

向量的產生也可使用函式 `vector()` 產生特定模式的向量.

```
> vector(mode = "logical", length = 0)
```

但是使用函式 `vector()` 須特別設定向量的 長度 (length) 與 模式 (mode). 使用函式 `numeric()`, `integer()`, `complex()`, `logical()`, `character()` 等, 則可以直接產生所要設定的模式之向量.

```
> ## vector()
> vector("numeric", 5)
[1] 0 0 0 0 0
> vector("integer", 5)
[1] 0 0 0 0 0
> vector("logical", 5)
[1] FALSE FALSE FALSE FALSE FALSE
> vector("character", 5)
[1] "" "" "" "" ""
> vector("complex", 5)
[1] 0+0i 0+0i 0+0i 0+0i 0+0i
```

2.6 基本運算操作符號 Basic Operator

R 是以 向量 (vector), 矩陣 (matrix), 陣列 (array), 列表 (Lists), 或 資料框架 (data frames) 等物件儲存資料. R 的最基本物件是向量, 向量 是指包含相同 模式 (mode) 的元素組成, 主要的 基本模式 (basic mode) 有量化數值型 (實數型), 整數向量 (有時需特別指定) 邏輯型 (TRUE or FALSE), 複數型, 文字型 (或字串). 資料分析基本上是 產生資料物件, 對物件命名, 使用函式對物件運算操作. 透過指令, 很容易地對物件進行統計分析與統計繪圖.

R 對物件運算操作有其 基本操作符號 (basic operators), 如同 C 語言, 可以分成算數操作 (arithmetic operator), 相關比較操作 (relation/comparison operator), 邏輯操作 (logical operator). R 也是一種高階程式語言 (programming language), 因此提供了其它程序語言共有的條件 (if-else), 轉換 (switch), 迴圈 (loop) 與函式

(function) 等, 程序控制結構語法, 進階資料分析使用高階程式語言與函式寫作進行. 本章先討論基本運算操作.

2.6.1 算數操作 Arithmetic Operator

向量物件的 算數操作 (arithmetic operator) 符號包含 +, -, *, /, ^, %%, %*%, %/%, %x% 等, 這些符號的使用, 如同一般數字的算數操作, R 延伸運用在向量物件上, 通常其含意是對向量的每一個元素進行運算的 "單元運算子" (unary) 或 "二元運算子" (binary), 向量物件主要的算數操作符號, 參見表 2.3.

表 2.3: 算數操作 (Arithmetic Operator)

符號	定義
-	減法運算 (Substraction, can be unary or binary)
+	加法運算 (Addition, can be unary or binary)
!	否定運算 (Unary not)
*	乘法運算 (Multiplication, binary)
/	除法運算 (Division, binary)
^	指數乘冪運算 (Exponentiation, binary)
%%	整數除法的餘數 (Modulus, binary)
%/%	整數除法的商數 (Integer divide, binary)
%*%	矩陣內積乘法 (Matrix product, binary)
%o%	矩陣外積乘法 (Outer product, binary)
%x%	矩陣 Kronecker 乘法 (Kronecker product, binary)
%in%	配對運算 (Matching operator, binary, in model formulae: nesting)

```
> ## Arithmetic Operator
> 1+2
[1] 3
> 2*3+4
[1] 10
> 2*(3+4)
[1] 14
> (3+11*2)/4
```

```
[1] 6.25
> x.complex <- (8+3i)+(1+2i)
> x.complex
[1] 9+5i
> #
> x.vec <- 1:5
> y.vec <- c(-1, -2, 0, 2, 4)
> z.vec <- c(2, 2, 3, 3, 4)
> x.vec+y.vec
[1] 0 0 3 6 9
> x.vec-y.vec
[1] 2 4 3 2 1
> #
> x.vec*2
[1]  2  4  6  8 10
> x.vec*y.vec
[1] -1 -4  0  8 20
> x.vec/2
[1] 0.5 1.0 1.5 2.0 2.5
> x.vec/y.vec
[1] -1.00 -1.00   Inf  2.00  1.25
> #
> x.vec^2
[1]  1  4  9 16 25
> x.vec^z.vec
[1]   1   4  27  64 625
> y.vec/2
[1] -0.5 -1.0  0.0  1.0  2.0
> y.vec/x.vec
[1] -1.0 -1.0  0.0  0.5  0.8
> #
> x.vec%*%y.vec # matrix inner product
     [,1]
[1,]   23
> y.vec%*%x.vec
     [,1]
[1,]   23
> x.vec%x%y.vec # matrix Kronecker product
 [1]  -1  -2   0   2   4  -2  -4   0   4   8  -3  -6
[13]   0   6  12  -4  -8   0   8  16  -5 -10   0  10
[25]  20
> y.vec%x%x.vec
 [1]  -1  -2  -3  -4  -5  -2  -4  -6  -8 -10   0   0
[13]   0   0   0   2   4   6   8  10   4   8  12  16
[25]  20
> #
> y.vec%%3  # modular arithmetic remainder
[1] 2 1 0 2 1
> y.vec%/%3 # integer division
[1] -1 -1  0  0  1
```

```
> y.vec%/%x.vec
[1] -1 -1  0  0  0
```

2.6.2 關係比較操作 Relation/Comparison Operator

邏輯向量 (logic vector) 的元素值有 TRUE, FALSE. 可以分別簡寫爲 T 和 F. 注意 T 和 F 不是系統真正的 保留 名字 (reserved word), 可以被覆寫, 使用邏輯數值應該儘量使用 TRUE 和 FALSE. 邏輯向量通常可以由 條件控制算式 (conditional expression) 產生.

在 R 之內, 向量之間的 關係比較操作 (relation/comparison operator) 可以產生邏輯向量. 關係比較操作符號包含常見的 <, < =, >, > = 以及判斷相等的 = = 和判斷不等的 ! = 等. 關係比較操作符號參見 表 2.4. 例如:

```
> x.vec <- 1:5
> y.vec <- (x.vec > 2)
> y.vec
[1] FALSE FALSE  TRUE  TRUE  TRUE
```

上述的程式顯示 y.vec 是一個長度和 x.vec 一樣的向量, 它的元素值爲 TRUE 表示 x.vec 的元素符合控制條件 x.vec > 2, 它的元素值爲 TRUE 則相反.

```
> x.vec <- 1:5
> y.vec <- c(0, 2, 4, 6, 8)
> #
> x.vec < 2
[1]  TRUE FALSE FALSE FALSE FALSE
> x.vec< =  2
[1]  TRUE  TRUE FALSE FALSE FALSE
> x.vec = =  2
[1] FALSE  TRUE FALSE FALSE FALSE
> x.vec ! =  2
[1]  TRUE FALSE  TRUE  TRUE  TRUE
> #
> x.vec < y.vec
[1] FALSE FALSE  TRUE  TRUE  TRUE
> x.vec < (y.vec-2)
[1] FALSE FALSE FALSE FALSE  TRUE
> x.vec < =  y.vec
[1] FALSE  TRUE  TRUE  TRUE  TRUE
> x.vec < =  (y.vec-2)
```

```
[1] FALSE FALSE FALSE  TRUE  TRUE
> #
> x.vec = =  y.vec
[1] FALSE  TRUE FALSE FALSE FALSE
> x.vec = =  (y.vec-2)
[1] FALSE FALSE FALSE  TRUE FALSE
> x.vec ! =  y.vec
[1]  TRUE FALSE  TRUE  TRUE  TRUE
> x.vec ! =  (y.vec-2)
[1]  TRUE  TRUE  TRUE FALSE  TRUE
```

表 2.4: 關係比較操作符號 Relation/Comparison Operator

符號	定義
<	小於 Less than, binary
>	大於 Greater than, binary
= =	相等 Equal to, binary
! =	不相等 Not equal to
> =	大於或等於 Greater than or equal to, binary
< =	小於或等於 Less than or equal to, binary
!	邏輯 “否定” (Logical NOT)

2.6.3 邏輯操作 Logical Operator

在 R 之內, 向量之間的關係比較操作可以產生邏輯向量. 在 R 中可對邏輯向量進行運算操作, 邏輯操作 (logical operator) 符號包含常見的 `!`, `&`, `&&`, `|`, `||`, 如果 `X` 和 `Y` 表示邏輯向量, 那麼 `X & Y` 是它們的交集運算 (AND), `X | Y` 是聯集運算 (`OR`); `!X` 是 `X` 的否定運算 (`X` 與 非 `X`). 在算術運算中採用邏輯變數, 會將邏輯變數強制轉換成數值變數, `FALSE` 變成 0, `TRUE` 變成 1. `xor(X, Y)` 的運算是互斥聯集運算, X 或 Y 僅能有一個爲 `TRUE`, 才會回傳 `TRUE`, 若 X 或 Y 都爲 `FALSE`, 會回傳 `FALSE`. 邏輯運算符號參見表 2.5 與 圖 2.1.

表 2.5: 邏輯運算操作符號 Logic Operator

符號	定義
`!`	邏輯 “否定”, 向量的個別元素使用 (Logical NOT)
`&`	邏輯 “和”, 向量的個別元素使用 (Logical AND, binary, vectorized)
`&&`	邏輯 “和”, 二元操作 (Logical AND, binary, not vectorized)
`|`	邏輯 “或”, 向量的個別元素使用 (Logical OR, binary, vectorized)
`||`	邏輯 “或”, 二元操作 (Logical OR, binary, not vectorized)
`xor()`	邏輯 “或”, 向量個別元素互斥聯集運算, 僅有 1 爲 TRUE

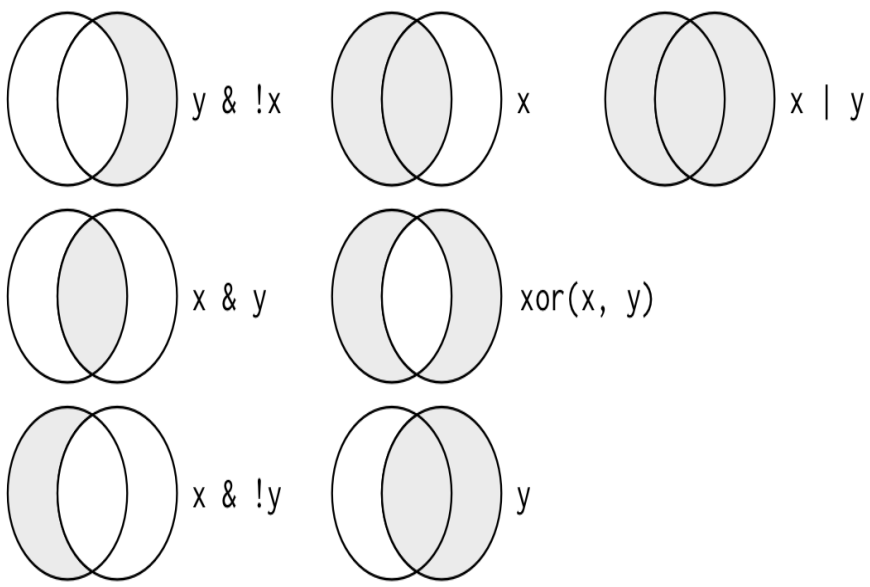

圖 2.1: R 邏輯操作

```
> ## Logical Operator: AND OR XOR
> x.vec <- 1:5
> y.vec <- c(0, 2, 4, 6, 8)
> (x.vec > 0) &  (y.vec > 0)  # return vector AND
[1] FALSE  TRUE  TRUE  TRUE  TRUE
> (x.vec > 0) && (y.vec > 0)  # return scalar AND
[1] FALSE
> #
> (x.vec > 0) &  ((y.vec-3) > 0)     # return vector AND
[1] FALSE FALSE  TRUE  TRUE  TRUE
> ((x.vec-2) > 0) && ((y.vec-3) > 0) # return scalar AND
[1] FALSE
```

```
> #
> (x.vec > 0) &  ((y.vec+3) > 0)     # return vector AND
[1] TRUE TRUE TRUE TRUE TRUE
> ((x.vec-2) > 0) && ((y.vec+3) > 0) # return scalar AND
[1] FALSE
> #
> (x.vec > 0)  (y.vec > 0) # return vector OR
[1] TRUE TRUE TRUE TRUE TRUE
> ((x.vec- 2) > 0)  ((y.vec - 3) > 0)
[1] FALSE FALSE  TRUE  TRUE  TRUE
> #
> (x.vec > 0)  (y.vec > 0) # return scalar OR
[1] TRUE
> ((x.vec-2) > 0)  ((y.vec-3) > 0)
[1] FALSE
> #
> (x.vec > 0)  ((y.vec+3) > 0) # return scalar OR
[1] TRUE
> ((x.vec-2) > 0)  ((y.vec+3) > 0)
[1] TRUE
> #
> xor((x.vec > 0), (y.vec > 0)) # return vector exclusive OR
[1]  TRUE FALSE FALSE FALSE FALSE
> xor(((x.vec-2) > 0), ((y.vec-3) > 0))
[1] FALSE FALSE FALSE FALSE FALSE
> xor(((x.vec-2) > 0), ((y.vec+3) > 0))
[1]  TRUE  TRUE FALSE FALSE FALSE
> #
> xx.vec <- (x.vec < =  3)
> yy.vec <- (y.vec > =  4)
> xx.vec
[1]  TRUE  TRUE  TRUE FALSE FALSE
> yy.vec
[1] FALSE FALSE  TRUE  TRUE  TRUE
> #
> xx.vec && yy.vec
[1] FALSE
> xx.vec & yy.vec
[1] FALSE FALSE  TRUE FALSE FALSE
> xx.vec  yy.vec
[1] TRUE
> xx.vec  yy.vec
[1] TRUE TRUE TRUE TRUE TRUE
> xor(xx.vec, yy.vec)
[1]  TRUE  TRUE FALSE  TRUE  TRUE
> #
> x.vec <- c(NA, FALSE, TRUE)
> names(x.vec) <- as.character(x.vec)
> y.vec <- x.vec
> outer(x.vec, y.vec, "&") ## AND table
```

```
       <NA> FALSE  TRUE
<NA>     NA FALSE    NA
FALSE FALSE FALSE FALSE
TRUE     NA FALSE  TRUE
> outer(x.vec, y.vec, "") ## OR table
      <NA> FALSE TRUE
<NA>    NA    NA TRUE
FALSE   NA FALSE TRUE
TRUE  TRUE  TRUE TRUE
> outer(x.vec, y.vec, "xor") ## XOR table
      <NA> FALSE  TRUE
<NA>    NA    NA    NA
FALSE   NA FALSE  TRUE
TRUE    NA  TRUE FALSE
```

2.6.4 向量元素命名 Names of Vectors

向量的每個元素或部分元素都可以命名, 使用者可以在輸入元素時直接給予命名, 或另外使用函式 `names()` 給予命名. 若要移除命名, 使用函式 `unname()` 移除命名, 或使用 `names(x.vec) <- NULL` 移除命名.

```
> ## vector names
> x.vec <- c(age = 50, chol = 220, dbp = 84, sbp = 132) # directly
> x.vec
 age chol  dbp  sbp
  50  220   84  132
> names(x.vec)
[1] "age"  "chol" "dbp"  "sbp"
> #
> x.vec <- c(55, 236, 80, 140)
> names(x.vec) <- c("age", "chol", "dbp", "sbp")
> #
> y.vec.name <- names(x.vec)
> y.vec <- c(60, 214, 90, 144)
> names(y.vec) <- y.vec.name
> y.vec
 age chol  dbp  sbp
  60  214   90  144
```

2.6.5 向量下標與索引 Vector Indexing

一個向量的長度 (length) 是向量元素的數目, 一個向量的個別元素或部分元素

可以向量的下標 (index) 取得, 向量的 下標 或 索引 (index) 是在向量名稱後面加 “中括號 []”, 並放入下標數目 (或向量) 得到. 原始向量的下標可以採用下列四種方式的任何一種, 正整數, 負整數, 文字或字串 與 邏輯向量.

1. 正整數 (positive integrs) 下標向量. 這種情況下, 下標向量必須是正整數 `1, 2, ..., length(x)`, 的子向量. 這種下標向量可以是任意長度, 取得原始向量中的下標向量所相對應的元素.

```
> #### Vector Indexing
> ## positive integer
> x.vec <- 1:50
> x.vec[7]
[1] 7
> x.vec[11:15]
[1] 11 12 13 14 15
> y.vec <- x.vec[11:15]
> y.vec
[1] 11 12 13 14 15
```

2. 負整數 (negative integrs) 下標向量. 下標向量在負整數 `-1, -2, ..., -length(x)`, 中取下標值, 表示刪除原始向量中所相對應位置的元素.

```
> ## negative integer
> z.vec <- 6:10
> z.vec[-c(2,4)]
[1]  6  8 10
```

3. 文字或字串 (character strings) 下標向量. 在定義原始向量時, 可以對個別的元素加上名字, 先以函式 `names()` 加上屬性, 這種情況下 文字 或 字串 下標向量可以如上整數下標向量使用. 文字 或 字串 下標向量的好處就是容易了解與記憶使用 (但不能打錯字).

```
> ## character string
> fruit.vec <- c(5, 10, 1, 20)
> fruit.vec
[1]  5 10  1 20
> names(fruit.vec) <- c("orange", "banana", "apple", "peach")
> fruit.vec
orange banana  apple  peach
     5     10      1     20
> lunch.vec <- fruit.vec[c("apple", "orange")]
```

```
> lunch.vec
 apple orange
     1      5
```

4. 邏輯向量 (logical vector) 下標向量. 邏輯下標向量必須和原始向量長度一致, 才能挑選元素, 若原始向量中所對應的下標向量爲 TRUE, 則將會得到原始向量中所對應的下標向量的元素; 而原始向量中那些所對應的下標向量爲 FALSE 的元素則將會被刪除. 如果原始向量的下標都是 FALSE, 則回傳結果是一個 0 長度的向量, 顯示爲 numeric(0).

```
> ## logical index
> x.vec <- c(NA, -2, -1, NA, 1, 2, NA)
> x.vec
[1] NA -2 -1 NA  1  2 NA
> y.vec <- x.vec[!is.na(x.vec)]
> y.vec
[1] -2 -1  1  2
> z.vec <- x.vec[x.vec > 0 & !is.na(x.vec)]
> z.vec
[1] 1 2
> x.vec[x.vec < 0] # Note: NA
[1] NA -2 -1 NA NA
> y.vec[y.vec < 0]
[1] -2 -1
> z.vec[z.vec < 0]
numeric(0)
```

2.6.6 複數向量

R 具備複數運算, 複數向量 (complex vector) 只要使用類似 2.3+4.6.1i 這樣的格式即可. 使用函式 complex 產生複數向量

```
> complex(length.out = 0, real = numeric(), imaginary = numeric(),
          modulus = 1, argument = 0)
```

複數向量的每一個元素都是複數, 可以使用函式 Re() 計算複數的實部, 使用函式 Im() 計算複數的虛部, 使用函式 Mod() 計算複數極座標的模數, 使用函式 Arg() 計

算複數極座標的角度. 例如, 令

$$
\begin{aligned}
z &= x + y\mathrm{i} \\
r &= \sqrt{x^2 + y^2} \\
x &= r \star \cos(\phi) \\
y &= r \star \sin(\phi)
\end{aligned}
$$

則複數的實部爲 `x = Re(z)`, 複數的虛部爲 `y = Im(z)`, 極座標的模數爲 `r = Mod(z)`, 極座標的角度爲 `phi = Arg(z)`, 最後, `z.conj = Arg(z)` 得到共軛複數.

```
> ## complex vector
> z.vec <- c(3+5i, 5+1i, 6+7i)
> Re(z.vec)  # real component
[1] 3 5 6
> Im(z.vec)  # imaginary component
[1] 5 1 7
> Mod(z.vec) # modulus argument
[1] 5.830952 5.099020 9.219544
> Arg(z.vec) # complex argument
[1] 1.0303768 0.1973956 0.8621701
> #
> z <- complex(real = 0, imaginary = 1)
> Mod(z)
[1] 1
> Arg(z)
[1] 1.570796
> pi / 2
[1] 1.570796
> Conj(z) # conjugate
[1] 0-1i
> #
> sqrt(-1)
[1] NaN
Warning message:
In sqrt(-1) : NaNs produced
> sqrt(as.complex(-1))
[1] 0+1i
```

2.6.7 向量長度改變與元素重覆使用原則 Recycling Rule

R 對不同長度的向量做運算操作, 其結果向量的長度會與最長向量的長度相同. 長度較短的向量, 其元素會 循環使用 (recycle), 直到長度與較長的向量長度相同. 相同長度的向量做運算, R 會運算並得到結果, 不同長度的向量做運算, R 也會運算並得到結果, 但 R 會提出警告, 使用者須小心. 對於不同基本模式的向量元素進行融合, 若不合階層結構, 則 R 回傳缺失值.

```
> ## Recycling and length change
> x.vec <- 1:3
> y.vec <- 5:10
> 2 + x.vec # 2 recycles 3 times, no warning
[1] 3 4 5
> z.vec <- x.vec + y.vec  # x.vec recycles + Warning
> z.vec
[1]  6  8 10  9 11 13
```

2.6.8 物件融合規則 Coercion Rule

向量, 矩陣, 陣列 等原型物件 (atomic mode) 的元素都必須由相同的基本模式組成, 若要由不同基本模式的原型物件組合成新的原型物件, R 會將原型物件資料中不同的基本模式的元素 融合 (coercion) 成相同的基本模式之原型物件, 使得新的原型物件內的元素都具有相同的基本模式. 其融合的依據是階層結構: `character > complex > numeric > logical`. 例如, 若向量中包含有任一文字元素, 則新的向量內所有的元素都將轉換融合成文字基本模式.

```
> ## coercion rule
> x.vec <- c("hellp", 3+6i, 5.42, FALSE) # coerce to character
> x.vec
[1] "hellp" "3+6i"  "5.42"  "FALSE"
> mode(x.vec)
[1] "character"
> typeof(x.vec)
[1] "character"
```

```
> y.vec <- c(3+6i, 5.42, FALSE) # coerce to complex
> y.vec
[1] 3.00+6i 5.42+0i 0.00+0i
> mode(y.vec)
[1] "complex"
> typeof(y.vec)
[1] "complex"
> z.vec <- c(5.42, FALSE) # coerce to numerical
> z.vec
[1] 5.42 0.00
> mode(z.vec)
[1] "numeric"
> typeof(z.vec)
[1] "double"
```

2.7 遺失值 (缺失值) Missing Values

研究資料, 通常會有 遺失值 或 缺失值 (missing value, incomplete data), 在 R 中, 輸入或輸出遺失值, 通常以 NA 表示, (NA = "Not Available"), R 還有另外有 NaN = "Not a Number" 是指物件運算後產生非數值結果, 以及 NULL 是指物件的長度是 0. 任何對遺失值 (NA) 的算數操作, 會得到遺失值 (NA) 結果. 使用函式 is.na(), is.nan() 可以查看向量內那些元素是遺失值. 回傳一個邏輯向量. 對遺失值作比較大小運算須非常小心. 要移除遺失值, 可以使用函式 na.omit(), na.fail(), na.exclude(), na.action() 等指令. 且函式 complete.cases() 可以同時移出多個缺失值. 對於不同基本模式的向量元素進行融合, 若不合階層結構, 則 R 回傳缺失值.

```
> ## missing value
> z.vec <- c(1:2, NA)
> z.vec
[1]  1  2 NA
> is.na(z.vec)
[1] FALSE FALSE  TRUE
> log(z.vec)
[1] 0.0000000 0.6931472        NA
> z.vec/0
[1] Inf Inf  NA
> 0/0
```

```
[1] NaN
> Inf-Inf
[1] NaN
> #
> is.na(z.vec)
[1] FALSE FALSE  TRUE
> is.nan(z.vec)
[1] FALSE FALSE FALSE
> is.nan(0/0)
[1] TRUE
> #
> x.vec <- c(1, 2, NA, 4, NA, 5, 6)
> bad <- is.na(x.vec)
> x.vec[!bad]
[1] 1 2 4 5 6
> #
> x.vec <- c(1, 2, NA, 4, NA, 5, 6)
> y.vec <- c("a", "b", NA, "d", NA, "f", "g")
> good <- complete.cases(x.vec, y.vec)
> good
[1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE
> x.vec[good]
[1] 1 2 4 5 6
> y.vec[good]
[1] "a" "b" "d" "f" "g"
> #
> data(airquality)
> airquality[1:6, ]
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
> good <- complete.cases(airquality)
> airquality[good, ][1:6, ]
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
7    23     299  8.6   65     5   7
8    19      99 13.8   59     5   8
> # Nonsensical coercion results in NA
> x.vec <- c("a", "b", "c")
> as.numeric(x.vec)
[1] NA NA NA
Warning message:
NAs introduced by coercion
```

```
> as.logical(x.vec)
[1] NA NA NA
> as.complex(x.vec)
[1] NA NA NA
Warning message:
NAs introduced by coercion
```