

# R 學習筆記

Ting-Chih Hung  
Email: [b08302310@ntu.edu.tw](mailto:b08302310@ntu.edu.tw)

Last updated: 2021-08-14



# Contents



# 序言

本筆記主要內容為？與？。



## Part I

# 基本語法





# Chapter 1

## R 資料結構

### 1.1 變數與基本的運算

---

以下分別是「加」、「減」、「乘」、「除」、「餘數除法」、與「整數除法」：

```
3 + 4
3 - 4
3 * 4
3 / 4
15 %% 2 # get "1"
15 %/% 2 # get "7"
```

我們可以 `<-` 來指派值給變數：

```
x <- 3 * 4
y <- 4 * 20
z <- x + y
z # get "92"
```

最基本的資料型態有三種：numeric、character 與 logical。numeric 即數字，character 即字串而 logical 即布林值 TRUE 或 FALSE。此外，使用 `as.integer()` 可以把 numeric 轉成型態 integer，<sup>1</sup>也另有複數為型態 complex。而我們可以使用 `class()` 來查看變數的資料型態，如：

```
GPA <- 4.3
class(GPA) # get "'numeric'"

word <- "GPA"
class(word) # get "'character'"
```

---

<sup>1</sup>傳 character 至 `as.integer` 的話將會產生 NA，傳 TRUE 或 FALSE 的話會變成整數 1 或 0，而傳 complex 的話則虛數的部份將會被自動捨棄。

```
yes.no <- TRUE
class(yes.no) # get "'logical'"

GTA <- as.integer(5)
class(GTA) # get "'integer'"

complex.number <- 2 + 5i
class(complex.number) # get "'complex'"

as.integer(TRUE)
```

我們也可以使用科學記號代表很大的數，如：

```
2.35e7 # get "23500000"
```

Logical value 也可以進行運算，其中 TRUE 代表 1，而 FALSE 代表 0，如：

```
TRUE + TRUE # get "2"
TRUE + FALSE # get "1"
TRUE * FALSE # get "0"
FALSE * FALSE # get "0"
```

此外，還有兩種特殊的資料型態：NULL 與 NA。當我們指派一個 NULL 給一個變數時，會讓其變成一個空的物件。

```
null.object <- NULL
null.object # get "NULL"
class(null.object) # get "'NULL'"
```

而 NA 的意思是 non-available，常常表示 missing data。但特別的是用 class() 檢視為 NA 的變數時會發現其為一 logical value。

```
missing <- NA
missing # get "NA"
class(missing) # get "'NA'"
```

NA 運算後也會產生 NA：

```
1 + NA # get "NA"
TRUE - NA # get "NA"
```

## 1.2 向量

向量 (vector) 是一列元素。我們以 c() 可以創造向量，裡頭的元素可以是 numeric、character 或 logical 其中一種，或者混合，而之間須以逗號分隔；我們可以用 length() 來得知向量的長度；如果裡頭的元素都是 numeric、character 或 logical 其中一種，則該向量的型態即其元素的型態，如：

```
vector.numbers <- c(1, 2, 3, 4)
vector.numbers # get "1 2 3 4"
length(vector.numbers) # "4"
class(vector.numbers) # get "'numeric'"

vector.characters <- c("R", "is", "cool")
vector.characters # get "'R' 'is' 'cool'"
class(vector.characters) # get "'character'"

vector.logicals <- c(TRUE, FALSE)
vector.logicals # get "TRUE FALSE"
class(vector.logicals) # get "'logical'"
```

但如果是向量內有不同種類的元素，那麼 R 會去讓向量裡的元素的型態變得更「一般」，即 character 優先於 numeric 又優先於 logical，就連 NA 都能被變成其他型態，如：

```
c("I", "am", 1) # get "'I' 'am' '1'"
class(c("I", "am", 1)) # get "'character'"

c("He", "is", "the", TRUE) # get "'He' 'is' 'the' 'TRUE'"
class(c("He", "is", "the", TRUE)) # get "'character'"

c(1, FALSE) # get "1 0"
class(c(1, FALSE)) # get "'numeric'"

c(NA, "1") # get "NA '1'"
class(c(NA, "1")) # get "'character'"

c(NA, TRUE) # get "NA TRUE"
class(c(NA, TRUE)) # get "'logical'"
```

### 1.2.1 創造向量的其他方法

如果我們要創造一個連續的向量，可以使用 a:b，如：

```
3:12 # get "3 4 5 6 7 8 9 10 11 12"
class(3:12) # get "'numeric'"
```

如果我們想要重複 (repeat) 某個數字或某群數字好幾遍，可以使用 rep()，例如：

```
rep(c(3, -1, 0.5), 3) # get "3.0 -1.0 0.5 3.0 -1.0 0.5 3.0 -1.0 0.5"
rep(c(3, -1, 0.5), times=3) # get "3.0 -1.0 0.5 3.0 -1.0 0.5 3.0 -1.0 0.5"
# 所以第二個 argument 預設是整串重複幾次的意思
# 但也能改成個別重複幾次，如下：
rep(c(3, -1, 0.5), each=3) # get "3.0 3.0 3.0 -1.0 -1.0 -1.0 0.5 0.5 0.5"
```

或者，我們可以在第二個 argument 丟入一個向量，指定第一個 argument 的個別的向量分別要重複幾次，但這時候會長得像在第二個 argument 丟入 `each=` 的樣子，如：

```
rep(c(3, -1, 0.5), times=c(2, 1, 3)) # get "3.0 3.0 -1.0 0.5 0.5 0.5"
```

我們也可以在第二個 argument 丟入一個 `length.out=`，這會使第一個 argument 的向量重複，而直到長度與 `length.out=` 的值相當，如：

```
rep(c(3, -1, 0.5), length.out=8) # get "3.0 -1.0 0.5 3.0 -1.0 0.5 3.0 -1.0"
```

除了手動輸入、使用 `a:b` 或者以 `rep()` 的方式產生向量，我們還可以使用 `seq()`，這可以創造出一個數列。`seq()` 有三個引數，第一個引數為起始值，第二個引數為結束值，第三個引數則為公差，如：

```
seq(0, 10, 2.5) # get "0.0 2.5 5.0 7.5 10.0"
seq(0, 10, 2.3) # get "0.0 2.3 4.6 6.9 9.2"
```

### 1.2.2 標籤

只要創建了向量，裡頭的元素就能以 `name()` 命名。在我們要指出向量中的特定元素時還蠻有用的。此功能也有點像 Python 裡頭的字典，如：

```
temperatures <- c(28, 29, 27, 27, 30)
names(temperatures) <- c("Monday", "Tuesday", "Wednesday",
"Thursday", "Friday")
temperatures
```

```
##      Monday    Tuesday Wednesday  Thursday    Friday
##          28          29          27          27          30
```

這時候，output 就不會有 `[1]` 了，而是變成結構性的帶有標籤的向量。此外，如果要新增一個有同樣多座標且標籤順序一致的向量，有兩種方法。第一種方法是直接把 `names`(標籤向量) 指派給 `names`(欲標籤的向量)，這樣會把前者的標籤貼到後者身上；或者，我們還是可以仿效原先的做法，再新創一個同等長度的字串向量，然後輸入 `names`(欲標籤的向量)，如下：

```
# Method 1
rains <- c(0, 5, 6, 0, 2)
names(rains) <- names(temperatures)
rains
```

```
##      Monday    Tuesday Wednesday  Thursday    Friday
##          0          5          6          0          2
```

```
# Method 2
rains <- rep(NULL, 5) # 先清空剛剛指派的 named num [1:5]
rains <- c(0, 5, 6, 0, 2)
days <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
names(rains) <- days
```

```
rains
```

```
##      Monday    Tuesday Wednesday  Thursday    Friday
##           0          5          6          0          2
```

### 1.2.3 向量的運算

而如果我們想要把向量中的每個數字都加上另一個數字，只要直接把兩個物件相加就行了（雖然這違反了數學邏輯的公理）；如果要把向量中的每個數字都乘上某個數字也是同理，如攝氏溫標轉換成凱式溫標與華氏溫標：

```
# Kelvin degrees
Ktemp <- temperatures + 273.15
Ktemp
```

```
##      Monday    Tuesday Wednesday  Thursday    Friday
##      301.15    302.15    300.15    300.15    303.15
```

```
# Fahrenheit degrees
Ftemp <- temperatures * 1.8 + 32
Ftemp
```

```
##      Monday    Tuesday Wednesday  Thursday    Friday
##       82.4      84.2      80.6      80.6      86.0
```

減、除、次方也是類似的道理。

此外，`sum()` 可以將所有向量的座標加總，如：

```
total.rains <- sum(rains)
total.rains # get "13"
```

### 1.2.4 存取向量中的元素

在 R 語言中，第一個位置是 1（而不像有些語言是從 0 開始），並依序下去。如果我們要存取一個向量中的元素，可以直接指定向量中的位置；若要存取多個元素，就要使用 變數 `[c(想指定的位置)]`；存取連續的位置則依然可以使用 `a:b`；當然也可以丟等差數列 `seq()` 進去，如：

```
# 存取一個元素
rains[2]
```

```
## Tuesday
##       5
```

```
# 存取多個元素
rains[c(2, 5)]
```

```
## Tuesday  Friday
##       5       2
```

```
# 存取連續的元素
```

```
rains[2: 5]
```

```
##   Tuesday Wednesday Thursday   Friday
##         5           6           0           2
```

```
# 等差地存取元素
```

```
rains[seq(1, 5, 2)]
```

```
##   Monday Wednesday   Friday
##         0           6           2
```

除了以位置來存取，當然也可以標籤來存取向量中的元素，如：

```
rains[c("Monday", "Tuesday")]
```

```
##   Monday Tuesday
##         0         5
```

### 1.2.5 關係

- <：小於。
- >：大於。
- <=：小於或等於。
- >=：大於或等於。
- ==：相等。
- !=：不相等。

在兩個值之間加入這些關係符號，會輸出一個布林值，描述兩者的關係是否符合輸入的關係，如：

```
4 > 3 # get "TRUE"
4 != 4 # get "FALSE"
```

向量也能與一個數字比較，此時會依據向量裡頭的位置順序，連帶著標籤（如果有的話），輸出各個座標的布林值，如：

```
rains > 0
```

```
##   Monday Tuesday Wednesday Thursday   Friday
##   FALSE     TRUE      TRUE      FALSE     TRUE
```

也能考慮多重的關係。& 代表的是「且」(and)，而 | 代表的是「或」(or)，而函數 xor() 在其中一者為真一者為假時會回傳真，在兩者皆真或皆假的情況會回傳假，如：

```
3 == 4 & 3 == 3 # get "FALSE"
3 == 4 | 3 == 3 # get "TRUE"
3 < 5 & 4 > 2 # get "TRUE"
TRUE & TRUE & FALSE # get "FALSE"
```

```
xor(3 > 2, FALSE) # get "TRUE"
xor(3 > 2, TRUE)  # get "FALSE"
xor(3 < 2, FALSE) # get "FALSE"
```

我們可以運用這些比較關係來確認向量裡頭的元素是否符合某個性質。例如是否下雨、是否溫度高於某個給定的溫度，即：

```
# 沒下雨的日子
not.rainy.days <- rains == 0
not.rainy.days
```

```
##      Monday   Tuesday Wednesday Thursday   Friday
##      TRUE      FALSE      FALSE      TRUE      FALSE
```

```
# 炎熱的日子
hot.days <- temperatures >= 29
hot.days
```

```
##      Monday   Tuesday Wednesday Thursday   Friday
##      FALSE      TRUE      FALSE      FALSE      TRUE
```

還可以用剛剛新建的向量，來存取原先向量中的位置，如：

```
# 在 hot.days 中，Tues. 與 Fri. 是真
# 所以 rains 中 Tues. 與 Fri. 標籤的位置的標籤與值會被叫出來
rains[hot.days]
```

```
## Tuesday   Friday
##         5         2
```

此外，我們可以透過兩個函數來一次檢查向量中的元素與某個性質之間的關係，即 `all()` 與 `any()`。前者在都有滿足的情況下就會回傳 `TRUE`，反之則回傳 `FALSE`；後者在其中一個座標有滿足的情況下就會回傳 `TRUE`，反之則回傳 `FALSE`。例如以下分別為「如果所有日子氣溫都大於等於 28 度則回傳 `TRUE`」與「如果其中一天氣溫恰等於 30 度則回傳 `TRUE`」，即：

```
all(temperatures >= 28) # get "FALSE"
any(temperatures >= 30) # get "TRUE"
```

另外，`which()` 則可以找到滿足條件的位置，即回傳為 `TRUE` 的位置的標籤與索引值，如找出溫度恰等於 27 度的日子：

```
which(temperatures == 27)
```

```
## Wednesday Thursday
##          3         4
```

### 1.2.6 排序

我們可以使用 `order()`，其引數為一個向量，而此函數會回傳此向量中的元素的索引值為一個向量，由原向量的元素由小到大排列。如下，`order(some.vector)` 回傳的向量的第一個元素為 5，這意味著原先的向量中最小的元素 2 其索引值為 5，而次小的元素 3 其索引值為 1，依此類推，最大的元素 9 其索引值為 3：

```
some.vector <- c(3, 7, 9, 6, 2, 8)
order(some.vector)
```

```
## [1] 5 1 4 2 6 3
```

所以，將 `order(some.vector)` 作為 `some.vector[]` 的引數（回憶節 ??），我們可以由小到大重新排列向量，即回傳一個向量，其第一個元素為原向量索引值 5 的元素，其值為 2，依此類推，如：

```
some.vector[order(some.vector)]
```

```
## [1] 2 3 6 7 8 9
```

不過要把向量由小排到大也可以使用 `sort()` 就是了。此操作不會改變原本的向量的排序，但會回傳排序後的向量。不過與上面的方法不同的是，此方法無法知道這些值原先在什麼位置：

```
some.vector <- c(3, 7, 9, 6, 2, 8)
sort(some.vector)
```

```
## [1] 2 3 6 7 8 9
```

### 1.2.7 其他操作

我們可以使用 `sum()` 將向量的所有元素的值加總；`max()` 與 `min()` 則分別會回傳其中最大與最小的值；`range()` 則將回傳向量中的元素的範圍（即同時回傳 `min()` 與 `max()`）；`mean()` 可以計算向量的算術平均，如：

```
sum(some.vector) # get "35"
max(some.vector) # get "9"
min(some.vector) # get "2"
range(some.vector) # get "2 9"
mean(some.vector) # get "5.833333"
```

結合 `which()`（回傳為 TRUE 的位置的標籤與索引值）與 `max()`（找到氣溫最高的值），就可以找到氣溫最高的日子的標籤與索引值，如：

```
temperatures
```

```
##      Monday    Tuesday Wednesday  Thursday    Friday
##         28         29         27         27         30
```



```
which(temperatures == min(temperatures))
```

```
## Wednesday Thursday
##          3         4
```

```
max(temperatures)
```

```
## [1] 30
```

### 1.3 矩陣

---

#### 1.3.1 創建矩陣

我們可以用 `matrix()` 來創建矩陣，其語法為：

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
```

其中，`data` 為我們要放進矩陣的向量資料；`nrow` 為 row 的數量；`ncol` 為 column 的數量；`byrow` 為以 column 或 row 的方式填滿矩陣（預設為 `FALSE`，即以 column 的方式填滿矩陣）；`dimnames` 則可以添加一個 list 為 row 與 column 的名字到矩陣（見節 ??）。

對於 `byrow` 的用法如下兩個例子：

```
matrix(1:6, nrow = 2, byrow = TRUE)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```
matrix(1:6, nrow = 2, byrow = FALSE) # default argument
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

而如果我們以向量的資料填滿矩陣，只要丟入 `nrow` 或 `ncol` 其中一者的引數即可，R 會根據資料的長度，算出 column 或 row 該要是多少。但如果向量資料的長度無法非 `nrow` 或 `ncol`，就會出現 `Warning`。但 R 還是會像資料長度為 `nrow` 或 `ncol` 的因數的情況一樣，從頭開始以原本的向量把矩陣給填滿，如：

```
matrix(1:11, nrow = 3)
```

```
## Warning in matrix(1:11, nrow = 3): data length [11] is not a sub-multiple or
## multiple of the number of rows [3]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9    1
```

矩陣也可以放以 character 或 logical 組成的向量資料，如：

```
matrix(c("January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"), nrow = 4)
```

```
##      [,1]      [,2]      [,3]
## [1,] "January" "May"      "September"
## [2,] "February" "June"    "October"
## [3,] "March"   "July"    "November"
## [4,] "April"   "August"  "December"
```

```
matrix(c(TRUE, FALSE, TRUE, TRUE, FALSE, FALSE), nrow = 2)
```

```
##      [,1] [,2] [,3]
## [1,] TRUE TRUE FALSE
## [2,] FALSE TRUE FALSE
```

我們可以 `rownames()` 與 `colnames` 為 row 與 column 加上名字，使用方法如：

```
climate <- matrix(c(temperatures, rains), byrow = TRUE, nrow = 2)
rownames(climate) <- c("Temperatures", "Rains")
colnames(climate) <- days
climate
```

```
##      Monday Tuesday Wednesday Thursday Friday
## Temperatures    28     29         27         27     30
## Rains           0      5          6          0      2
```

我們可以透過 `dim()` (想得知的矩陣) 得知 row 與 column 的數量，如：

```
dim(climate) # get "2 5"
```

### 1.3.2 合併矩陣與矩陣運算

我們可以透過 `rbinds()` 與 `cbinds()` 在矩陣新增 row 或 column，如：

```
Winds <- c(30, 25, 22, 24, 18)
total.climate <- rbind(climate, Winds)
total.climate
```

```
##      Monday Tuesday Wednesday Thursday Friday
## Temperatures    28     29         27         27     30
## Rains           0      5          6          0      2
## Winds           30     25         22         24     18
```

我們可以使用 `rowSums()` 或 `colSums()` 把各個 row 或 column 的值加起來。把各個 row 的值相加以後指派給變數 `totals`，然後與原先的 `climate` 合併，就如：

```
totals <- rowSums(total.climate)
cbind(total.climate, totals)
```

```
##      Monday Tuesday Wednesday Thursday Friday totals
```

```
## Temperatures      28      29      27      27      30      141
## Rains              0       5       6       0       2       13
## Winds             30      25      22      24      18      119
```

### 1.3.3 存取矩陣中的元素

存取矩陣的方式與向量類似，即 矩陣 [row, col]，如：

```
total.climate[2, 3] # get "6"
```

如果不輸入 row 的引數，則會列出該 column 所有的資料，如下列出了週三的氣溫、雨量及風速：

```
total.climate[, 3]
```

```
## Temperatures      Rains      Winds
##              27         6         22
```

如果不輸入 column 的引數，則會列出該 row 所有的資料，如下列出了每日的雨量：

```
total.climate[2, ]
```

```
##    Monday  Tuesday Wednesday Thursday   Friday
##         0         5         6         0         2
```

另外，也可以直接指定某個 row 或 column 的名字來存取資料，如：

```
total.climate[, "Wednesday"]
```

```
## Temperatures      Rains      Winds
##              27         6         22
```

```
total.climate["Rains",]
```

```
##    Monday  Tuesday Wednesday Thursday   Friday
##         0         5         6         0         2
```

```
total.climate["Rains", "Wednesday"]
```

```
## [1] 6
```

### 1.3.4 矩陣的運算

我們可以用 mean() 計算向量的算術平均，我們也能以此計算矩陣的 column 或 row 的算術平均。例如想要計算每天的平均氣溫，可以如下：

```
mean(total.climate["Temperatures",])
```

```
## [1] 28.2
```

矩陣也能像向量一般運算（回憶節 ??），如：

```
climate * 2
```

```
##           Monday Tuesday Wednesday Thursday Friday
## Temperatures    56     58         54         54     60
## Rains           0      10         12          0      4
```

```
climate ^ 2
```

```
##           Monday Tuesday Wednesday Thursday Friday
## Temperatures   784     841         729         729    900
## Rains          0      25         36          0      4
```

```
# 計算雨量與氣溫的比例
```

```
climate[2, ] / climate[1, ]
```

```
##      Monday   Tuesday Wednesday   Thursday     Friday
## 0.00000000 0.17241379 0.22222222 0.00000000 0.06666667
```

#### 1.4 FACTORS

質性或類別變數 (qualitative or categorical variable) 的值即類別，而非數值。把向量丟到 `factor()` 中可以把向量轉換成 `factor`。

創建一個向量 `sizes`，如：

```
sizes <- c("Small", "Big", "Big", "Medium", "Medium", "Small", "Medium", "Small", "Small")
sizes
```

```
## [1] "Small" "Big"   "Big"   "Medium" "Medium" "Small" "Medium" "Small"
## [9] "Small"
```

但這時候如果用 `summary()`，我們只看得到資料長度和他們的類別：

```
summary(sizes)
```

```
##      Length      Class      Mode
##           9 character character
```

此時，把向量丟到 `fator()` 中將會回傳不太一樣的結果，並且 `summary()` 將會回傳各個值出現的次數；`levels()` 則會回傳此 `factor` vector 中有哪些元素（重複的不計），如：

```
factor.sizes <- factor(sizes)
factor.sizes
```

```
## [1] Small Big   Big   Medium Medium Small Medium Small Small
## Levels: Big Medium Small
```

```
summary(factor.sizes)
```

```
##      Big Medium Small
```

```
##      2      3      4
levels(factor.sizes)

## [1] "Big"      "Medium" "Small"
```

類別變數可以分為無序 (nominal) 與有序 (ordinal) 兩種。前者如顏色，後者如尺寸。如果要讓變數是有序的，我們在 `factor()` 中須將引數 `ordered` 設置為 `TRUE`，並且以引數 `levels` = 向量由小到大地來描述順序關係，如：

```
sizes2 <- c("Small", "Big", "Big", "Medium", "Medium", "Small", "Medium", "Small", "Small")
factor.sizes2 <- factor(sizes, ordered=TRUE, levels=c("Small", "Medium", "Big"))
factor.sizes2
```

```
## [1] Small Big Big Medium Medium Small Medium Small Small
## Levels: Small < Medium < Big
```

我們也可以把向量指派給 `level`(目標 factor)，來覆寫 factor 中的值，如：

```
survey.vector <- c("M", "F", "F", "M", "M", "F", "M", "M")
factor.survey.vector <- factor(survey.vector)
# 此時儲存的順序是 "F M"，因此可以 c("Female", "Male") 覆寫之
levels(factor.survey.vector) <- c("Female", "Male")
factor.survey.vector
```

```
## [1] Male Female Female Male Male Female Male Male
## Levels: Female Male
```

無序的 factor 的元素無法比較，但有序的可以，如：

```
factor.sizes2[1] < factor.sizes2[2] # get "TRUE"
factor.sizes2[2] == factor.sizes2[2] # get "TRUE"
factor.sizes2[2] != factor.sizes2[3] # get "FALSE"
```

## 1.5 DATA FRAMES

在 vectors 與 matrices 中，資料都會儲存成一樣的型態。但進行資料分析時，我們會同時處理不同型態的資料。此時我們需要 data frames 來儲存資料表。與矩陣不同的是，data frames 可以儲存不同型態的資料。R 有內置的 datasets，可以使用以下指令查看其資訊：

```
?datasets
library(help="datasets")
```

我們也可以使用 `data()` 查看電腦內現有的 datasets。

我們可以使用其中一個現成的 dataset `OrchardSprays`。輸入 `OrchardSprays` 可以查看整個表。我們可以發現有四個變數，分別是 `decrease`、`rowpos`、`colpos` 與 `treatment`。`decrease` 為對處置的反應，以數值記載；`rowpos` 與 `colpos` 分別代表其 design 的 row 與 column；`treatment` 則為 A 至 H 的類別變數。以

`head(OrchardSprays)` 可以查看表格的前六個 rows。以 `str()` 可以獲知表格的資訊，如：

```
str(OrchardSprays)
```

```
## 'data.frame':    64 obs. of  4 variables:
## $ decrease : num  57 95 8 69 92 90 15 2 84 6 ...
## $ rowpos   : num  1 2 3 4 5 6 7 8 1 2 ...
## $ colpos   : num  1 1 1 1 1 1 1 1 2 2 ...
## $ treatment: Factor w/ 8 levels "A","B","C","D",...: 4 5 2 8 7 6 3 1 3 2 ...
```

我們可以用 `dataframe()` 來創建 data frames。其中的引數放入向量，而依序構成該 data frames 的各個 columns，如：

```
name <- c("Alfonso", "Carlos", "Lluis", "Diego")
last.name <- c("Zamora", "Quesada", "Hurtado", "Mondejar")
second.last.name <- c("Saiz", "Gonzalez", "Gil", "Ruiz")
age <- c(33, 32, 30, 37)
phd <- c("math", "math", "physics", "math")
office <- c(4, 14, 6, 8)
from.madrid <- c(FALSE, TRUE, FALSE, TRUE)
professors <- data.frame(name, last.name, second.last.name, age, phd, office, from.madrid)
str(professors)
```

```
## 'data.frame':    4 obs. of  7 variables:
## $ name      : chr  "Alfonso" "Carlos" "Lluis" "Diego"
## $ last.name : chr  "Zamora" "Quesada" "Hurtado" "Mondejar"
## $ second.last.name: chr  "Saiz" "Gonzalez" "Gil" "Ruiz"
## $ age       : num  33 32 30 37
## $ phd       : chr  "math" "math" "physics" "math"
## $ office    : num  4 14 6 8
## $ from.madrid : logi  FALSE TRUE FALSE TRUE
```

### 1.5.1 存取 data frames 中的元素與 subset

想要選取 data frames 中的元素，就像選取矩陣中的元素一樣，即 `dataframes[row, col]`，如：

```
professors[2, 3]
professors[1, ] # 顯示第一個 row
professors[, 2] # 顯示第二個 column
professors[1 : 2, ] # 選擇頭兩個 rows
```

要注意的是，data frames 的 column 的名字是來自 column vector 的名字，所以我們也可以用 `$` 來存取特定的 column vector，如：

```
professors$phd
```

```
## [1] "math"      "math"      "physics"   "math"
```

```
professors$phd[3]
```

```
## [1] "physics"
```

我們還可以用 logical 的方式來叫出 data frames 的 subset，例如要叫出 from.madrid 為 TRUE 的兩個 rows，即：

```
madrileans <- professors$from.madrid
professors[madrileans, ]
```

```
##      name last.name second.last.name age  phd office from.madrid
## 2 Carlos   Quesada      Gonzalez  32 math    14        TRUE
## 4 Diego  Mondejar              Ruiz  37 math     8        TRUE
```

以指令 subset(data\_frame, subset=logical\_condition) 也能做到類似的事，如：

```
subset(professors, subset=age > 31)
```

```
##      name last.name second.last.name age  phd office from.madrid
## 1 Alfonso   Zamora      Saiz  33 math     4        FALSE
## 2 Carlos   Quesada      Gonzalez  32 math    14        TRUE
## 4 Diego  Mondejar              Ruiz  37 math     8        TRUE
```

```
subset(professors, subset=phd == "math")
```

```
##      name last.name second.last.name age  phd office from.madrid
## 1 Alfonso   Zamora      Saiz  33 math     4        FALSE
## 2 Carlos   Quesada      Gonzalez  32 math    14        TRUE
## 4 Diego  Mondejar              Ruiz  37 math     8        TRUE
```

### 1.5.2 排序

想要以某個特定的變數的順序整理 data frames，可以使用 order()，如：

```
positions <- order(professors$age)
professors[positions, ]
```

```
##      name last.name second.last.name age  phd office from.madrid
## 3 Lluís   Hurtado      Gil  30 physics    6        FALSE
## 2 Carlos   Quesada      Gonzalez  32 math    14        TRUE
## 1 Alfonso   Zamora      Saiz  33 math     4        FALSE
## 4 Diego  Mondejar              Ruiz  37 math     8        TRUE
```

### 1.5.3 其他操作

我們可以 as.data.frame() 把矩陣轉為 data frames。

## 1.6 LISTS

Lists 就像向量，但其元素可以有不同的長度、大小、型態。我們可以用 `list()` 創建 lists，如：

```
new.list <- list(days, factor.sizes, climate)
new.list
```

```
## [[1]]
## [1] "Monday"    "Tuesday"    "Wednesday"  "Thursday"   "Friday"
##
## [[2]]
## [1] Small  Big    Big    Medium Medium Small  Medium Small  Small
## Levels: Big Medium Small
##
## [[3]]
##           Monday Tuesday Wednesday Thursday Friday
## Temperatures  28      29      27      27      30
## Rains         0       5       6       0       2
```

我們也可以選取 lists 中的元素，如：

```
new.list[1]    # 第一個元素
new.list[[1]][3] # 第一個元素中的第三個元素
new.list[[3]][1, 2:5] # 第三個元素中的第一個 row 的第二至五個的元素
```

我們也可以為 list 中的元素命名，如將第一個元素命名為 `the.days`、第二個元素命名為 `the.factors`、將第三個元素命名為 `the.data`：

```
new.list <- list(the.days=days, the.factors=factor.sizes, the.data=climate)
new.list
```

```
## $the.days
## [1] "Monday"    "Tuesday"    "Wednesday"  "Thursday"   "Friday"
##
## $the.factors
## [1] Small  Big    Big    Medium Medium Small  Medium Small  Small
## Levels: Big Medium Small
##
## $the.data
##           Monday Tuesday Wednesday Thursday Friday
## Temperatures  28      29      27      27      30
## Rains         0       5       6       0       2
```

這樣就可以用 `$` 名字來存取 list 中的元素，如：

```
new.list$the.factors
new.list$the.data[2, 5]
new.list["the.data"]
```



如果要新增元素到 list，可以 `list_name[[" 名字"]] <-` 要輸入的物件指令，其中名字當然是選擇性的，也能空下來；此外，如果要刪除 list 中的成份，只要指派 NULL 給它就行，如：

```
new.list[["professors"]] <- professors # 新增第四個元素，名為 "professors" 的 data frames
new.list[[""]] <- positions # 新增第五個元素，無名的向量
new.list[[5]] <- NULL # 刪除第五個元素
new.list[["professors"]] <- NULL # 刪除名為 "professors" 的元素
```

我們也可以用 `str()` 來查看 lists 的資訊，如：

```
str(new.list)
```



## Chapter 2

# 控制結構與函數

### 2.1 CONDITIONALS

---

條件句的語法如下：

```
if ("condition is satisfied") {  
  "do something"  
}
```

例如：

```
x <- 3  
if (x > 0) {  
  print("Positive")  
}
```

```
## [1] "Positive"
```

如果我們希望在條件沒有滿足時執行其他動作，就得使用 `else`，其語法如：

```
if ("condition is satisfied"){  
  "do something"  
} else {  
  "otherwise do something else"  
}
```

例如：

```
x <- 0  
if (x > 0) {  
  print("Positive")  
} else {
```

```
    print("Negative")
  }
```

上述的 codes 其實在邏輯上是正確的，但顯然與我們希望電腦能做的事有落差 ( $x = 0$  非正也非負才對)。所以，如果有好幾個條件要確認，也能多用幾個 `else blocks`，如：

```
x <- 0
if (x > 0) {
  print("Positive")
} else if (x < 0) {
  print("Negative")
} else {
  print("Zero")
}
```

不過，要注意的是上述的結構在 R 的運行速度較慢，不如以 `ifelse()` 函數，把上述的結構寫成一行，其語法即：

```
ifelse("condition", "task if TRUE", "task if FALSE")
```

前開判別變數為正或負的條件結構也能寫成：

```
x <- 9
ifelse(x > 0, "Positive", "Negative")
```

```
## [1] "Positive"
```

只要在向量使用 `binary operator`，同樣的條件就也可以被確認多次。例如如果要確認向量中的元素是否小於 5，我們可以寫成：

```
ifelse((1 : 10) < 5, "Fail", "Pass")
```

```
## [1] "Fail" "Fail" "Fail" "Fail" "Pass" "Pass" "Pass" "Pass" "Pass" "Pass"
```

## 2.2 LOOPS

`for` 用於重複次數預先指定的時候，`while` 則反之，將持續運行直到條件無法滿足。

### 2.2.1 for 迴圈

`for` 迴圈的語法為：

```
for ("counter" in "vector of indices") {
  "do something"
}
```

例如印出 1-10 可以如此：

```
for (i in 1 : 10) {  
  print(i)  
}
```

```
## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5  
## [1] 6  
## [1] 7  
## [1] 8  
## [1] 9  
## [1] 10
```

因為 [1] 出現了十次，我們可知這段程式碼有十次輸出，每次輸出包含一個 row，顯然這是 `print(i)` 運行十次的結果。在此迴圈中，每次執行都會使 `i` 的值增加，所以如果我們此時在 console 輸入 `i`，將會回傳 10，顯示 `i` 作為一個變數，其值已經變成 10 了。

那如果我們要把連續的整數和存成一個向量該怎麼做呢？

```
v <- c()  
  
s <- 0  
for (i in 1 : 10) {  
  s <- s + i  
  v[i] <- s  
}  
v
```

```
## [1] 1 3 6 10 15 21 28 36 45 55
```

其中，`s` 的作用就是在迴圈結束後儲存剛剛的結果，以便與下一個數相加。如此，可以依序將  $1$ 、 $1+2$ 、 $1+2+3$ 、 $\dots$ 、 $1+\dots+10$  存到向量 `v` 並印出。

以下還有幾個 `for` 迴圈運作的簡單範例。例如印出 1-19 的奇數，可以：

```
odd <- 2 * (1 : 10) - 1  
for (i in odd) {  
  print(i)  
}
```

```
## [1] 1  
## [1] 3  
## [1] 5  
## [1] 7  
## [1] 9
```

```
## [1] 11
## [1] 13
## [1] 15
## [1] 17
## [1] 19
```

我們也可以把 for 迴圈與 if 條件句結合，要把某個向量中低於某個數字的元素印出 FAIL，如：

```
for (i in 1 : 10) {
  if (i < 5) {
    print("Fail")
  } else {
    print("Pass")
  }
}
```

```
## [1] "Fail"
## [1] "Fail"
## [1] "Fail"
## [1] "Fail"
## [1] "Pass"
## [1] "Pass"
## [1] "Pass"
## [1] "Pass"
## [1] "Pass"
## [1] "Pass"
```

### 2.2.2 while 迴圈

除了 for 迴圈以外，還有 while 迴圈，其語法為：

```
while ("condition holds") {
  "do something"
}
```

以下為一個 while 迴圈的簡單範例：

```
i <- 0
while (i < 10) {
  print(c(i,"is less than 10"))
  i <- i + 1
}
```

```
## [1] "0"           "is less than 10"
## [1] "1"           "is less than 10"
## [1] "2"           "is less than 10"
## [1] "3"           "is less than 10"
## [1] "4"           "is less than 10"
```

```
## [1] "5"           "is less than 10"
## [1] "6"           "is less than 10"
## [1] "7"           "is less than 10"
## [1] "8"           "is less than 10"
## [1] "9"           "is less than 10"
```

第一圈  $i$  為 0，運行到 `print()` 時將會印出包含 “0” 與 “is less than 10” 元素的向量，而 `i <- i + 1` 相當於計數器，第一個迴圈結束後，變數  $i$  就會變成 1，以此類推。在第十個迴圈結束後，變數  $i$  將會變成 10，而就不符合 `while` 迴圈繼續執行的條件了，迴圈至此終止。

## 2.3 FUNCTIONS AND OPERATORS

---

### 2.3.1 創建新函數

R 本身就有很多預設的函數，如 `mean()` 可以取平均，`sum()` 可以加總，`sqrt()` 可以計算數字的平方根，`log()` 與 `exp()` 可以計算數字的對數值與指數值，`sin()`、`cos()`、`tan()` 可以計算三角函數值，`is.logical()` 可以確認型態是否是 `logical`。

如果要創建我們自己的函數，其語法為：

```
function.name <- function(argument1, argument2,...) {
  "body function"
}
```

函數的名字隨意，只要沒有與既存的函數同名或以某些不合法的方式命名即可；引述的數量可多可少；`body function` 則裝載執行程序，裡頭用的值由引數而來，最後一行命令則會回傳成輸出。

例如若想新建一個函數來協助計算  $f(x, y) = x^2 - \frac{y}{5}$  的話，可以

```
f <- function(x, y) {
  x ^ 2 - y / 5 # the output is the evaluation of last line
}
```

或者我們也可以寫成：

```
f <- function(x, y) x ^ 2 - y / 5
```

執行的最後一行預設就是輸出了，我們不需要寫出 `return`，但也能把它寫出來，如：

```
f <- function(x, y) {
  return (x ^ 2 - y / 5)
}
```

定義函數以後我們就可以使用此函數進行運算。

### 2.3.2 引數的預設值

函數不一定要輸入引數。如果引數有預設值，而未輸入引數的話，即隨預設值。創建一個有預設的引數的函數的語法，如：

```
function.name <- function(name.argument1=default.value1,
                           name.argument2=default.value2,...) {
  "body function"
}
```

以下是一個能輸出連續的數字為向量或矩陣的簡單範例：

```
mat.vec <- function(a, b=2, flag=FALSE){
  if (flag) {
    matrix(1 : a, nrow=b)
  } else {
    1:a
  }
}
```

此函數有三個變數：a 指矩陣或向量的元素個數；b 為如果我們想輸出的是矩陣的話其 rows 的數量；flag 決定是輸出矩陣還是向量，若 flag=TRUE，將會輸出包含 1-至 a，而有 b 個 rows 的矩陣（預設為 b=2），若 flag=FALSE，將會輸出包含 1 至 a 的向量（預設為 flag=FALSE）。有預設的引數還有一個好處，即就算我們漏輸入了、忘記了有哪些引數，程式碼還是能正常運行，而不會報錯。

函數能輸出的就只有單一個物件。所以如果想要輸出多個元素，可以使用 lists。例如我們可以創建一個函數，其回傳三項資訊：資料長度、總和與平均值，即：

```
items <- function(x) list(len=length(x),total=sum(x), mean=mean(x))
```

我們把向量 1:10 丟進此函數，可得：

```
data <- 1 : 10
result <- items(data)
result
```

```
## $len
## [1] 10
##
## $total
## [1] 55
##
## $mean
## [1] 5.5
```

如果我們想查看函數是如何構成的，可以直接輸入其名，如：



```
log
```

```
## function (x, base = exp(1)) .Primitive("log")
```

如果想要套用函數在多個元素時，可以使用 `lapply(list, function)`，其會把函數套用到 `vector` 或 `list` 的所有元素。`sapply()` 則類似於 `lapply()`，不同的是會把輸出簡化成向量或矩陣，而不是元素。其使用範例如：

```
salutation <- function(x) print("Hello")
# Note that this output does not depend on the value of x
output <- sapply(1 : 5, salutation)
```

```
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
```

或者也可以簡化成：

```
output <- sapply(1 : 5, function(x) print("Hello"))
```

```
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
```

以內建的 `cars` datasets 為例，其有兩個變數：`speed` 與 `dist`。當我們想要計算這兩者的算術平均，可以：

```
lapply(cars, mean)
```

```
## $speed
## [1] 15.4
##
## $dist
## [1] 42.98
```

或者：

```
sapply(cars, mean)
```

```
## speed dist
## 15.40 42.98
```



## **Part II**

# **資料庫與基本的資料前處理**



## Chapter 3

# R 的資料庫

### 3.1 資料來源、匯入與匯出

---

#### 3.1.1 Tabulated File Types

最簡單的表格檔就是每個 column 的資料由空格分開，而每個 row 的資料由換行分開。這種分隔符號（separator）方便卻有一個大問題，即兩個字的資訊無法儲存，例如 orange shirt 就會變成兩塊。

所以要解決這個問題，就是每個 column 的資料改由逗號分開。不過想當然耳，如果資料內有逗號，那儲存上也會出現問題。如果想解決這個問題，也能用分號分開每個 column。雖然使用到分號的機率低上許多，但還是沒辦法免於這種風險。所以說，分隔好的選擇其實沒有什麼放諸四海皆準的通則，只能多加注意。

表格檔可以儲存種許多種檔案格式，最常見的是 *comma separated values format*，即 .csv 檔。雖然它是如此稱呼，但我們在匯入與匯出資料時也都要敘明 separator。

#### 3.1.2 匯入與匯出

##### .csv 檔

要匯入 .csv 檔可以使用 `read.csv()`。其有多個引數，最重要的是以下的四個。

```
our.table <- read.csv("myfilewithdata.csv", header=TRUE, sep=";", dec=".")
```

第一個引數指定要從哪裡讀取資料；第二個引數決定表格有無 header；第三個引數指定 separator 而第四個引數指定 decimal separator。

而以 `write.csv()` 可以匯出 data frames 成 .csv 檔。第一個引數指定要輸出的 data frames；第二個引數指定輸出的名稱；第三個與第四個引數分別指定有無包含 column 與 row 的名字；第五個與第六個引數分別指定 separator 與 decimal separator。

```
write.csv(our.table, "filetosave.csv", col.names=FALSE, row.names=TRUE, sep=";", dec="
```

除了 .csv 檔也還有其他資料類型，如 .txt 與 .dat 檔，分別可以 `read.table()` 與 `read.delim()` 讀取。不過，我們要匯出資料時最好還是匯出為 .csv 檔，因為它最通用。

但要注意的是，前開的 `read.csv()` 與 `write.csv()` 處理起大數據速度很慢。我們可以使用一個 R package `data.table` 來解決這個問題，只要在 console 輸入：`library(data.table)`，<sup>1</sup>並分別以 `fread()` 與 `fwrite()` 來讀取與輸出檔案，其中 `f` 為“fast”之義。除了處理速度大增，使用 `fread()` 與 `fwrite()` 還可以讀取 .csv 檔以外的檔案，也可以自動識別 separator，而資料將會開啟成 `data.table` 的型態（`data.frame` 的升級版），而非 `data.frame`。

#### .json 檔

除了表格檔，我們還會需要使用階層檔（**hierarchical files**）。標準的階層檔檔案格式為 *JavaScript Object Notation*，即 .json 檔，就像書目一樣儲存資訊。但我們要讀取這類的檔案無法直接為之，必須借助套件 `jsonlite`。想要匯入一個 .json 檔到 R 裡頭並轉成 `data.frame` 格式只要輸入：

```
library(jsonlite)
jsonastable <- as.data.frame(fromJSON("ourjsonfile.json"))
```

#### .xls 與 .xlsx 檔

除外，還有一種檔案格式為試算表（**spreadsheets**）。MS Excel 就是最受歡迎的試算表程式。試算表雖然在某些領域頗有用，但也有計算速度與檔案大小的限制：Excel 檔最多只能有 1,048,576 個 rows 與 16,384 個 columns。想要在 R 載入 Excel 檔（.xls 或 .xlsx 檔），可以使用套件 `readxl`。輸入：

```
library(readxl)
data <- read_excel("file.xlsx", col_names=TRUE, col_types=c("numeric", "numeric"), sheet="
```

引數 `col_types` 指定每個 column 裡的資料型態，而引數 `sheet` 則選擇要匯入檔案中的哪張試算表。

## 3.2 資料收集

資料依據其結構化的程度分成：

- **Fully Structured:** 變數已經明確定義而為表格的形式的 datasets。不費吹灰之力就能轉換成 `data frames`。使用 APIs 或一些公開的資料來源可以取得此類型的資料。
- **Semi Structured:** 變數也已經明確定義，但還沒有變成表格的形式，例如網頁。還需要經前處理。
- **Unstructured:** 變數尚未明確定義，例如圖片、音樂或未經處理的文字。需要先經過其他前處理，才能轉換成可供分析的資料。

<sup>1</sup>Mac OS Big Sur 與搭載 ARM 64 處理器的蘋果電腦用戶可參考此文，以安裝 OpenMP，發揮 `data.table` 的完整效能。

處理非結構化的資料需要足夠的數學基礎，此處將會專於前兩者。

### 3.2.1 Data Repositories

存取資料最簡單的方式是使用免費公開的資料庫，如 The World Data Bank、Eurostat、U.S. Government's Open Data 等，或者其他商用的資料庫。此外，也有套件 `mlbench`，包含了 UCI Machine Learning Repository 的 datasets。如果要載入 Glass dataset，只要輸入：

```
library(mlbench)
data(Glass)
```

### 3.2.2 APIs

**Application Programming Interfaces (APIs)** 使我們可以直接存取資訊，而不需要透過手動下載 datasets 的方式。APIs 是一種通訊協定 (communication protocols)，有清楚的結構，可以有效率的傳遞資訊。此外，還可以如果有需要限制下載量，或者確認憑證 (credentials)，或者分割過大的檔案。

#### REST APIs

APIs 都會有文件說明如何使用，有些要以特定的程式語言才能存取，例如 Python 或 PHP。最常見的 APIs 為 **Representational State Transfer APIs (REST APIs)**，其透過 URLs 要求資料，而提供的資料都會是標準的結構化格式 (standard structured formats)。

因為每個 REST API 組織用以存取資料的 URLs 的方式都不同，所以使用 REST APIs 前得要先閱讀文件，了解如何存取想要的文件，然後再透過 R 存取之，若有需要並將其轉換成 data frames。

#### OpenSky Network Example

OpenSky Network 為一非營利組織，其透過 REST API 分享資料，旨在增進航空交通的安全與效率。我們要獲取當下的航班的狀態。第一步是閱讀其文件。因為其提供的文檔是 .json 檔，所以我們要載入 `jsonlite`：

```
library(jsonlite)
```

然後透過所提供的 URL，請求檔案，並轉成 data frame：

```
url <- "https://opensky-network.org/api/states/all"
flights <- as.data.table(fromJSON(URLencode(url)))
```

第一行把 URL 指派給字串 `url`；而第二行中，`fromJSON()` 可以讀取 .json 檔，`URLencode()` 則是要告訴 R，將要放進一個有 .json 檔的 URL，其說明可在 Console 輸入 `?URLencode` 了解，然後以 `as.data.frame()` 將資料轉成 data frame，並存進變數 `flights` 中（也可以 `as.data.table()` 把資料轉成 data table 的型態）

另一方面，如果 APIs 提供的檔案是 .csv 檔，那我們可以直接使用 `fread(URLencode(指定的 url))`，而毋需使用 `jsonlite`。

### 3.2.3 Web Scraping

從網頁抓取資料時，一般的流程為：

1. 清楚定義要下載的資訊為何。
2. 了解網頁的編碼方式。
3. 以 R 選擇並下載之。
4. 將其轉換為變數並組合成 data frame。

以下以兩個例子說明網頁抓取的流程。

#### TheSportsDB

雖說這個網頁有 API，但此僅作為範例。

每個網頁都有一個 robots.txt，在瀏覽器的網址列輸入 目標網頁/robots.txt 可以查看。以此處的 TheSportsDB 為例，即輸入 <https://www.thesportsdb.com/robots.txt> 就能查看之。其記載著：

```
User-agent: *  
Disallow:  
User-agent: AhrefsBot  
Disallow: /
```

我們先從其第三、四行開始閱讀，其意指機器人 AhrefsBot 被禁止從網頁抓取任何東西，其中 / 為「任何東西」之義，Disallow: / 即禁止抓取任何東西。第一、二行則表示所有既存的機器人（表示為 \*）沒有禁止，即存取是合法的。

我們想抓取的網頁是 <https://www.thesportsdb.com/season.php?l=4387&s=2020-2021>，即 NBA 2020–2021 年的球賽。多數的網頁是由 **HyperText Markup Language (HTML)** 編碼的。當我們連到上述的網頁，按下鍵盤 F12，並選取「元素」(Elements) 的頁面標籤，可以開啟如圖 ?? 的畫面。

在 .html 檔中，所有元素的首尾都有一個 tag，即以 <tag> 為開頭（可能會包含屬性，如 <tag attribute=value>），而以 </tag> 結尾。

在該網頁中，如圖 ??，當我們把游標放到第一場球賽的結果時，右欄 <td style="text-align:center; width:10%">112 - 116</td> 也會被強調。在這個 tag 中，描述了文字的對齊方式與要呈現哪些文字在螢幕上。

了解我們所要的資訊所對應到的程式碼是哪部分以後，下一個步驟是用 R 打開網頁，並告訴它要存取哪部分的資料。接下來的流程是：

1. 呼叫網頁，並將其存在變數中。
2. 指定要抓取的部分。
3. 轉換資訊成文字。

首先使用套件 rvest，用以擷取網頁資料。在 Console 輸入：



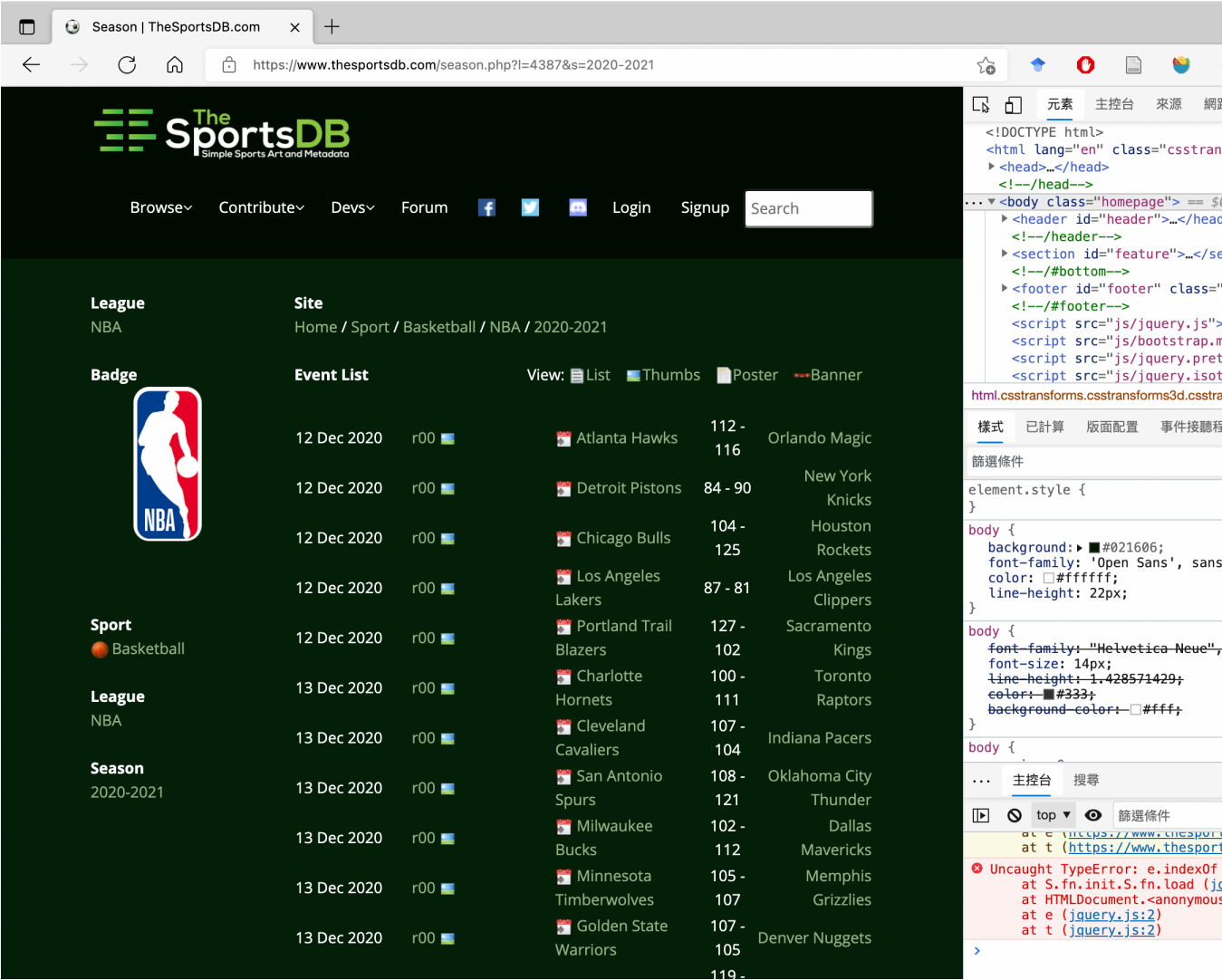


Figure 3.1: Inspection menu.

Site  
Home / Sport / Basketball / NBA / 2020-2021

Event List View: List Thumbs Poster Banner

12 Dec 2020	r00	Atlanta Hawks	112 - 116	Orlando Magic
12 Dec 2020	r00	Detroit Pistons	84 - 90	New York Knicks
12 Dec 2020	r00	Chicago Bulls	104 - 125	Houston Rockets

Figure 3.2: 第一場比賽的比分所對應到的程式碼。

Figure 3.2: 第一場比賽的比分所對應到的程式碼。

```
library(rvest)
```

並以指令 `read_html()` 呼叫網頁，將其儲存在變數中，如：

```
nbagames <- read_html("https://www.thesportsdb.com/season.php?l=4387&s=2020-2021")
```

輸入 `nbagames` 我們可以看到：

```
{html_document}
<html>
[1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">\n<scrip
ja ...
[2] <body class="homepage">\n\n<header id="header"><nav class="navbar navbar-inverse" 1
```

而我們可以利用 `html_nodes()` 指定要抓取的部分。此函數需要兩個引數，其一為網頁（剛剛把網頁存到的變數就要放在這裡），其二為我們所需要的 tag 的字串，因為我們需要 `<td>` 這個 tag，所以在這裡即輸入 `"td"`，如：

```
games <- html_nodes(nbagames, "td")
```

此時，變數 `games` 儲存了：

```
{xml_nodeset (7327)}
[1] <td><br></td>\n
[2] <td style="text-align:left; width:20%">12 Dec 2020</td>\n
[3] <td></td>\n
...
```

由此可見，匯入了所有 `td` tags，而儲存於 `xml_nodeset`，一種專為與 `.html` 互動而設計的資料型態。因為要轉換成 `character` 才能處理，所以我們使用 `html_text()`，如：

```
games <- html_text(html_nodes(nbagames, "td"))
```

此時，games 內儲存的就會是：

```
[1] "" "12 Dec 2020" ""
[4] "\n\t\t\t\n\t\t\t\tr00 " "Atlanta Hawks" "112 - 116"
[7] " Orlando Magic" "12 Dec 2020" ""
[10] "\n\t\t\t\t\n\t\t\t\t\tr00 " "Detroit Pistons" "84 - 90"
[13] " New York Knicks" "12 Dec 2020" ""
...

```

可以發現，此時還是有一些空字串與 `\n`（換新行）、`\t`（表格）與 `\tr00` 之類的字元。我們接下來的目標是把資訊拆開，存入不同的變數中。待到節 ?? 將會處理這些問題。

## Goodreads

第二個任務是 Goodreads 網頁所列的 21 世紀最佳書籍，可以參見：[https://www.goodreads.com/list/show/7.Best\\_Books\\_of\\_the\\_21st\\_Century](https://www.goodreads.com/list/show/7.Best_Books_of_the_21st_Century)。首先當然是查看 robots.txt。因此我們要連到 <https://www.goodreads.com/robots.txt>，可以發現有一大堆的 Disallow：

```
# See http://www.robotstxt.org/robotstxt.html for documentation on how to use the robots.txt file
User-agent: *
Disallow: /about/team_member/
Disallow: /admin
Disallow: /api
Disallow: /blog/list_rss
Disallow: /book/reviews/
Disallow: /book_link/follow/
...
```

但並非什麼存取都是不被允許的（因為 / 沒有被 Disallow），且 /list/ 也沒有被 Disallow，所以還是可以抓取此網頁。而我們可以發現，第一本書的標題所對應的 .html 碼為 `<span itemprop="name" role="heading" aria-level="4">Harry Potter and the Deathly Hallows (Harry Potter, #7)</span>`，而 span 這個 tag 就是對應到書名。因此我們似乎可以仿效之前的做法：

```
goodreadsurl <- "https://www.goodreads.com/list/show/7.Best_Books_of_the_21st_Century"
goodreads <- read_html(goodreadsurl)
books <- html_text(html_nodes(goodreads, "span"))
```

但我們將會發現，存進 books 的卻是：

```
[1] "Browse  "
[2] "Community  "
[3] ""
```

```
[4] ""
[5] ""
[6] ""
[7] "Profile"
[8] "Groups"
[9] "Groups"
[10] "Friends' recommendations"
[11] "Friends' recommendations"
[12] "Browse  "
[13] "Community  "
[14] "Harry Potter and the Deathly Hallows (Harry Potter, #7)"
...
```

直到第 14 個元素，才抓到第一本書的書名。現在的問題就在，tag span 雖然有包含書名，但還有包含許多其他的資訊。要解決這個問題，我們可以發現 tag span 是在 tag a 的下層，而 tag a 有一屬性為 class="bookTitle"。Class 在 HTML 中的用處就是用來分類 tags 的方式，所以我們可以有單一個 tag，但有不同的種類的資訊。因此，我們可以把剛剛的程式碼改成：

```
goodreadsurl <- "https://www.goodreads.com/list/show/7.Best_Books_of_the_21st_Century"
goodreads <- read_html(goodreadsurl)
books <- html_text(html_nodes(goodreads, ".bookTitle span"))
```

指涉 class 的時候，前面要加上 .，而我們可以在該引數中同時放入 classes 和 tags。上述的程式碼即選取所有有 class 為 bookTitle 的 nodes，而選取裡頭的 tag span。結果，books 會是：

```
[1] "Harry Potter and the Deathly Hallows (Harry Potter, #7)"
[2] "The Hunger Games (The Hunger Games, #1)"
[3] "The Kite Runner"
[4] "The Book Thief"
[5] "Harry Potter and the Half-Blood Prince (Harry Potter, #6)"
[6] "Harry Potter and the Order of the Phoenix (Harry Potter, #5)"
[7] "The Help"
...
```

顯然，結果就正常許多。而我們也需要提取作者與評分的資訊，因此程式碼為：

```
goodreads <- read_html(goodreadsurl)
book <- html_text(html_nodes(goodreads, ".bookTitle span"))
author <- html_text(html_nodes(goodreads, ".authorName span"))
rating <- html_text(html_nodes(goodreads, ".minirating"))
topbooks <- data.frame(book, author, rating)
```

這時候算是大致完成了，但評分的部分的格式還是不太行，有些多餘的 hyphen 或字詞，在節 ??，將會處理這個問題。

### 臺大活動報名系統與 CSS 選擇器

想要找出網頁的 CSS 選擇器路徑，除了上述手動查看的方法以外，如果使用 Google chrome 系列的瀏覽器，更方便的做法是使用如 SelectorGadget 之類的外掛。安裝完成並開啟以後直接用滑鼠點選想要抓取的元素即可，如圖 ??。而如果包含到不想要選取的元素，就點選使其變成紅色，未包含到想要選取的元素則亦再行點選之即可。之後，SelectorGadget 將會產生一組 CSS 選擇器。如我們要抓取活動的代碼，其 CSS 選擇器即 `.actID`。

因此，我們可以如此抓取臺大活動系統之過期活動中 2021 年的活動清單，並製作成一張 data table：

```
events.url <- "https://my.ntu.edu.tw/actregister/expiredActionList.aspx"
events <- read_html(events.url)
events.number <- html_text(html_nodes(events, ".actID"))
events.name <- html_text(html_nodes(events, ".multiline"))
events.time <- html_text(html_nodes(events, ".actTime"))
events.state <- html_text(html_nodes(events, "#ulbox .floatRight"))
events.table <- data.table(events.number, events.name, events.time, events.state)
```

結果如圖 ??。

### 臺大政治系專任教師資料簡表

```
library(data.table)
polisci.prof.url <- "http://politics.ntu.edu.tw/?cat=8"
polisci.prof <- read_html(polisci.prof.url)
prof.name <- html_text(html_nodes(polisci.prof, ".name a"))
prof.title <- html_text(html_nodes(polisci.prof, ".title"))
prof.phone <- html_text(html_nodes(polisci.prof, ".tel"))
prof.email <- html_text(html_nodes(polisci.prof, ".mail a"))
prof.table <- data.table(prof.name, prof.title, prof.email)
prof.phone <- append(prof.phone, NA, 13) # 蕭高彥沒有放電話
prof.table <- cbind(prof.table, as.data.table(prof.phone))

knitr::kable(prof.table, booktabs = TRUE, caption = '臺大政治系專任教師資料簡表。')
```

## 3.3 小結

此外，還有兩個抓取網頁上的困難，此處沒有提及：

1. 有些網頁在點擊之後會產生改變，但其網址並不會改變。此外，有些資訊需要登入才能存取。而有些網頁在 html 之上又使用 javascript。上述的問題都無法透過 rvest 來解決。雖然有其他套件，如 rselenium 可以解決這些問題，但並不簡單。
2. 此外，有些平臺不想讓資訊流出給外人使用，會去偵測並阻斷自動抓取，這時候就很難從中提取有用的資訊。



活動報名系統 - 過期活動總表

https://my.ntu.edu.tw/actregister/expiredActionList.aspx

國立臺灣大學 National Taiwan University

活動報名系統

首頁 活動場次總表 過期活動總表 停權資訊與常見問題

:::首頁 > 過期活動總表


▼ 2021年 - 活動清單

<p><b>2021H100_05</b> 講座</p> <p>報名狀態：未報名</p> <p>臺大教職員工全人關懷服務方案-線上課 about you ^^ 線上健康運動小時光</p> <p>活動期間：2021/07/08 ~ 2021/07/08</p> <p>場次數：1 場</p>	<p><b>2021H100_05</b> 課程</p> <p>臺大教職員工全人關懷服務方案-線上課 about you ^^ 線上健康運動小時光</p> <p>活動期間：2021/07/06 ~ 2021/07/08</p> <p>場次數：2 場</p>
<p><b>2021I200_02</b> 其他活動</p> <p>報名狀態：未報名</p> <p>臺大雙語標竿學校推動計畫說明會</p> <p>活動期間：2021/06/28 ~ 2021/07/01</p> <p>場次數：3 場</p>	<p><b>2021I200_03</b> 其他活動</p> <p>報名狀態：未報名</p> <p>20210612龍洞灣峽國際沙龍季知性一日</p> <p>活動期間：2021/07/06 ~ 2021/07/08</p> <p>場次數：1 場</p>
<p><b>2021I325_07</b> 講座</p> <p>報名狀態：未報名</p> <p>講座取消--「認識肺結核」講座</p> <p>活動期間：2021/06/11 ~ 2021/06/11</p> <p>場次數：1 場</p>	<p><b>2021I325_05</b> 講座</p> <p>報名狀態：未報名</p> <p>講座取消--愛滋與反歧視宣導講座</p> <p>活動期間：2021/06/04 ~ 2021/06/04</p> <p>場次數：1 場</p>
<p><b>2021I325_06</b> 講座</p> <p>報名狀態：未報名</p> <p>講座取消--牙齒保健講座</p> <p>活動期間：2021/06/03 ~ 2021/06/03</p> <p>場次數：1 場</p>	<p><b>2021C200_01</b> 其他活動</p> <p>報名狀態：未報名</p> <p>《瑩回臺大首部曲一踩泥巴·造生態》</p> <p>活動期間：2021/05/29 ~ 2021/05/29</p> <p>場次數：1 場</p>
<p><b>2021I325_01</b> 講座</p> <p>報名狀態：未報名</p> <p>【音樂學研究討論】講座：生命的音韻（講者-張意將-第65代張天師）（取消）</p> <p>活動期間：2021/05/28 ~ 2021/05/28</p> <p>場次數：1 場</p>	<p><b>2021I325_01</b> 講座</p> <p>報名狀態：未報名</p> <p>【音樂學研究討論】講座：生命的音韻（講者-張意將-第65代張天師）（取消）</p> <p>活動期間：2021/05/28 ~ 2021/05/28</p> <p>場次數：1 場</p>
<p><b>2021I325_01</b> 講座</p> <p>報名狀態：未報名</p> <p>【音樂學研究討論】講座：生命的音韻（講者-張意將-第65代張天師）（取消）</p> <p>活動期間：2021/05/28 ~ 2021/05/28</p> <p>場次數：1 場</p>	<p><b>2021I325_01</b> 講座</p> <p>報名狀態：未報名</p> <p>【音樂學研究討論】講座：生命的音韻（講者-張意將-第65代張天師）（取消）</p> <p>活動期間：2021/05/28 ~ 2021/05/28</p> <p>場次數：1 場</p>

.actID

Figure 3.3: 使用 SelectorGadget。





events.table x

Filter

	events.number	events.name	events.time
1	20212400_03	2021職衛講座(二)：肌肉骨骼傷害的辨識與預防	2021/07/08 ~ 2021/07/08
2	2021H100_05	臺大教職員工全人關懷服務方案-線上課程NTU cares about you	2021/07/06 ~ 2021/07/06
3	20211200_02	臺大雙語標竿學校推動計畫說明會	2021/06/28 ~ 2021/06/28
4	2021I200_03	20210612龍洞灣峽國際沙雕季知性一日遊	2021/06/12 ~ 2021/06/12
5	20211325_07	講座取消--「認識肺結核」講座	2021/06/11 ~ 2021/06/11
6	20211325_05	講座取消--愛滋與反歧視宣導講座	2021/06/04 ~ 2021/06/04
7	20211325_06	講座取消--牙齒保健講座	2021/06/03 ~ 2021/06/03
8	2021C200_01	《螢回臺大首部曲—踩泥巴·造生態》	2021/05/29 ~ 2021/05/29
9	20215133_01	【音樂學研究討論】講座：生命的音籟（講者-張意將-第6屆...	2021/05/28 ~ 2021/05/28
10	20211911_07	活動取消_【演講】成就未來的你：屬於未來世代人才精準...	2021/05/27 ~ 2021/05/27
11	20212204_03	109學年度第2學期 全校系所單位網路管理人員會議(限單位...	2021/05/27 ~ 2021/05/27
12	20211923_08	(因應疫情，自2021年5月12日起講習取消，造成不便之處...	2021/05/27 ~ 2021/05/27
13	20211325_03	暫停辦理「遠離菸害、健康無礙」講座	2021/05/26 ~ 2021/05/26
14	20212400_02	2021環境教育講座（一）在氣候變遷下漁撈對漁業資源的...	2021/05/26 ~ 2021/05/26
15	2021A900_03	[給教職員工的藝文通識課]場次三、YK奇幻古樂雙人樂團	2021/05/26 ~ 2021/05/26
16	20215100_02	(5/14更新，取消活動) No. 241 文學院午間音樂會：「...	2021/05/21 ~ 2021/05/21
17	20211406_02	公開取得電子報價單暨電子投開標班(實機操作)	2021/05/20 ~ 2021/05/20
18	2021C100_01	百瑞精鼎(Parexel) 2021職涯發展徵才說明會	2021/05/20 ~ 2021/05/20

Figure 3.4: 抓取臺大活動系統中 2021 年的過期活動的資訊。

Table 3.1: 臺大政治系專任教師資料簡表。

prof.name	prof.title	prof.email	prof.phone
張登及	系主任	tchang@ntu.edu.tw	3366-8374
陳思賢	教授	genesis@ntu.edu.tw	3366-8320
黃錦堂	教授	hwngntn@ntu.edu.tw	3366-8313
石之瑜	教授	cyshih@ntu.edu.tw	3366-8316
蘇彩足	教授	tsaitu@ntu.edu.tw	3366-8418
朱雲漢	教授	yunhan@gate.sinica.edu.tw	3366-8397
吳玉山	教授	yushanwu@gate.sinica.edu.tw	3366-8381
王業立	教授	ylwang2008@ntu.edu.tw	3366-8413
蘇宏達	教授	hdsu@ntu.edu.tw	3366-8357
林俊宏	教授	demian@ntu.edu.tw	3366-8419
陳淳文	教授	chwenwen@ntu.edu.tw	3366-8362
徐斯勤	教授	schsu01@ntu.edu.tw	3366-8406
張佑宗	教授	yutzung@ntu.edu.tw	3366-8399
左正東	教授	ctso@ntu.edu.tw	NA
蕭高彥	教授	carl@gate.sinica.edu.tw	3366-8355
黃長玲	教授	changling@ntu.edu.tw	3366-8347
黃旻華	教授	mhhuang5103@ntu.edu.tw	3366-8396
陶儀芬	副教授	yftao@ntu.edu.tw	3366-8420
陳世民	副教授	shihmin@ntu.edu.tw	3366-8350
林子倫	副教授	tllin@ntu.edu.tw	3366-8405
王宏文	副教授	hongwung@ntu.edu.tw	3366-8403
李鳳玉	副教授	fylee323@ntu.edu.tw	3366-8321
郭乃菱	副教授	nailing@ntu.edu.tw	3366-8323
唐欣偉	副教授	hsinweitang@ntu.edu.tw	3366-8417
蔡季廷	副教授	chiting@ntu.edu.tw	3366-8305
黃心怡	副教授	hsinihuang@ntu.edu.tw	3366-8366
劉康慧	副教授	helenliu4@ntu.edu.tw	3366-8382
童涵浦	副教授	hanstung@ntu.edu.tw	3366-8402
洪美仁	副教授	meijhung@ntu.edu.tw	3366-8304
郭銘峰	副教授	mingfeng.kuo@gmail.com	3366-8385
黃凱苹	助理教授	kaipinghuang@ntu.edu.tw	3366-8387
廖小娟	助理教授	mandyliao@ntu.edu.tw	3366-8325
安井伸介	助理教授	anjing@ntu.edu.tw	3366-8336
郭銘傑	助理教授	jasonkuo@g.ntu.edu.tw	3366-8333
吳舜文	助理教授	shunwu@ntu.edu.tw	3366-8309
張貴閔	助理教授	changkueimin@ntu.edu.tw	3366-8312
蘇翊豪	助理教授	yihao@ntu.edu.tw	3366-8331



## Chapter 4

# 資料前處理

在資料分析前，我們須先進行資料前處理（data preprocessing）。準備資料有以下幾個步驟（雖然並非都是永遠需要）：

1. **Cleaning:** 改變 dataset 的變數的格式。例如清除節 ?? 中的 rating 變數中無用的字元。
2. **Integration:** 從不同處來的資訊，在清潔以後，要整合成一張 data frame。
3. **Transformation:** 創造一些需要的變數，重構 dataset 成為更便於分析的格式。
4. **Reduction:** 如果 dataset 很龐大，而要從事的分析只是一小部分的資料，就要刪除一些變數，以釋出記憶體。

此章將專於使用 base 與 data.table；當然，也可以使用 tidyr、dplyr、stringr 或 stringi；<sup>1</sup>本書第 ??、??、?? 章將會分別介紹 dplyr、tidyr 與 stringr。

### 4.1 DATA TABLES

Data Tables 即 data frames 的改良版，效能更高、功能更多、速度更好。並且，一個 data.table 物件也同時是一個 data.frame 物件，所以前者也可以使用後者的語法。要將向量或矩陣轉換為 data table 可以使用 as.data.table()，即：

```
our.matrix.DT <- as.data.table(our.matrix)
```

而既存的 data frame 或 list 可以使用 setDT() 將其轉換成 data table（雖然也可以使用 as.data.table()，但前者使用較少記憶體，速度更快）。不過，data table 的 row 沒有名字，所以如果要把有名字的 data frame 轉換成 data table，要使用 keep.rownames=T 引數，新增一個名為 rn 的 column，例如：

---

<sup>1</sup>Fast data lookups in R: dplyr vs data.table. <https://www.r-bloggers.com/2017/03/fast-data-lookups-in-r-dplyr-vs-data-table/>

```
library(data.table)
example <- data.frame(info1 = c(1, 2), info2 = c("a", "b"))
row.names(example) <- c("line1", "line2")
setDT(example, keep.rownames=T)
```

```
class(example)
```

```
## [1] "data.table" "data.frame"
```

而 data.table 物件有如 example.data[i, j, by] 的格式，此三個引數分別代表 row、column 與「分類依據」。

#### 4.1.1 排序 Ordering

要重新排序 data table 的 row 除了可以用節 ?? 的 data frame 的方法以外，還有更簡單、快速的語法。以內置的套件 datasets 中的 swiss 為例。為了避免與原本的 dataset 混淆，我們可以創建一個複本。然後將其轉換成 data table，並以城鎮名稱字母序排列（現在城鎮名為新創的 rn column）：

```
DT.swiss <- copy(swiss)
setDT(DT.swiss, keep.rownames=T)
DT.swiss[order(rn)]
```

我們也可以依據多個變數來排列。order() 中的第一個引數會優先排列，如果值相等，再依據第二個引數排列，依此類推。而 - 為降冪排列之義。

```
DT.swiss[order(Education, -Agriculture)]
```

#### 4.1.2 子集 Subsetting

如節 ?? 一樣存取 data frame 中的元素，我們依樣畫葫蘆來存取 data table 內的元素。例如我們如果想要存取前三個 rows 與前三個 columns 的元素，可以：

```
DT.swiss[1 : 3, 1 : 3]
```

也可以使用名字來選取，如：

```
DT.swiss[1 : 3, "rn"]
# 顯示第一至三個 rows 與名為 rn 的 column
```

或者使用 ! 來不選取某個名字的 column，如：

```
DT.swiss[1 : 3, !"Agriculture"]
# 顯示第一至三個 rows，不顯示名為 Agriculture 的 column
```

如果使用 - 在數字前，則可以選取那個 row 或 column，如：

```
DT.swiss[-(2 : 47),] # 只會選取第一個 row
DT.swiss[, -1] # 不選取第一個 column
```

或者以 c() 包裹的任何組合：

```
DT.swiss[1 : 3, c("rn", "Education", "Catholic")]
# 顯示第一至三個 rows，與名為 rn、Education、Catholic 的 columns
```

我們也可以使用 `our.data[condition on certain variables]` 來選取滿足條件的 rows（類似節 ?? 提及的 `subset()`，但以 data table 速度更快）。如要找到 Education 恰等於 9 的城鎮，可以：

```
DT.swiss[Education == 9]
```

其結果為：

	rn	Fertility	Agriculture	Examination	Education	Catholic	Infant.Mortality
1:	Delemont	83.1	45.1	6	9	84.84	22.2
2:	Lavaux	65.1	73.0	19	9	2.84	20.0
3:	St Maurice	65.0	75.9	9	9	99.06	17.8

或者要找到 Education 小於等於 2 的城鎮，可以：

```
DT.swiss[Education <= 2]
```

其結果為：

	rn	Fertility	Agriculture	Examination	Education	Catholic	Infant.Mortality
1:	Echallens	68.3	72.6	18	2	24.20	21.2
2:	Oron	72.5	71.2	12	1	2.40	21.0
3:	Conthey	75.5	85.9	3	2	99.71	15.1
4:	Herens	77.3	89.7	5	2	100.00	18.3

以邏輯條件選取 data table 中的元素時，也可以超過一個條件，如：

```
mean.values <- sapply(DT.swiss[, -1], mean)
DT.swiss[Agriculture > mean.values[2] & Education > mean.values[4]]
```

回憶節 ?? 所提及的，`sapply(list, function)` 可以套用函數在多個元素上，並輸出為向量或矩陣。`sapply(DT.swiss[, -1], mean)` 之義為套用 `mean()` 這個函數在 DT.swiss 中除了第一個 column 以外的其他 columns，即算出其平均值，並輸出成有標籤的向量。而我們將輸出的資料存入 `mean.values` 變數中。如此，我們就能找出同時滿足「Agriculture 與 Education 皆大於平均值」的 rows，即：

	rn	Fertility	Agriculture	Examination	Education	Catholic	Infant.Mortality
1:	Aigle	64.1	62.0	21	12	8.52	16.5
2:	Avenches	68.9	60.7	19	12	4.43	22.7
3:	Nyone	56.6	50.9	22	12	15.14	16.7
4:	Sion	79.3	63.1	13	13	96.83	18.1

另一個 data table 的優勢是第二個引數 `j` 其實可以傳入非索引值的物件，例如我們想知道 Catholic > 50 的城鎮其 Education 的平均可以透過：

```
DT.swiss[Catholic > 50, mean(Education)]
```

```
## [1] 9.111111
```

也可以知道 Catholic > 50 的城鎮其 Education 的平均是否大於整體的 Education 平均：

```
DT.swiss[Catholic > 50, mean(Education) > mean.values[4]]
```

```
## Education
##      FALSE
```

或者 Education 小於 10 的城鎮究竟有多少：

```
DT.swiss[Education < 10, length(Education)]
```

```
## [1] 28
```

### 4.1.3 加總 Aggregation

前面曾經提及，`example.data[i, j, by]` 中的引數 `by` 為分組依據。其使用要配合引數 `j`。例如我們想得知每一個 Education 的值所對應到的 Fertility 的平均值，並且依據 Education 降冪排列，可以：

```
DT.swiss[order(-Education), mean(Fertility), by=Education]
```

```
      Education      V1
1:          53 35.00000
2:          32 64.40000
3:          29 43.75000
4:          28 55.70000
5:          20 54.30000
...
```

當使用 `by` 來分類時，在引數 `j` 就無法使用 `length()` 來計算出現次數了。這時候可以改用 `.N` 來計算各組的數字，如：

```
DT.swiss[order(-Education), .N, by=Education]
```

```
      Education N
1:          53 1
2:          32 1
3:          29 2
4:          28 1
5:          20 1
...
```

也可以與 `maen()` 結合起來，以下列出各 Education 程度的數量，並分別算出各 Education 程度的 Fertility 與 Catholic 的平均，而以 Education 降冪排列：

```
DT.swiss[order(-Education), .(N, mean(Fertility), mean(Catholic)), by=Education]
```

```

      Education N      V2      V3
1:          53 1 35.00000 42.34000
2:          32 1 64.40000 16.92000
3:          29 2 43.75000 54.38000
4:          28 1 55.70000 12.11000
5:          20 1 54.30000  2.15000
...

```

除此之外，by 也可以丟入邏輯式，如：

```
DT.swiss[, .N, .(Education < 15, Fertility > 60)]
```

```

      Education Fertility N
1:         TRUE         TRUE 37
2:        FALSE         TRUE  2
3:        FALSE        FALSE  6
4:         TRUE        FALSE  2

```

我們可以得知，有 37 個城鎮的 Education 小於 15，且 Fertility 大於 60；2 個城鎮的 Education 不小於 15，且 Fertility 大於 60；有 6 個城鎮的 Education 不小於 15，且 Fertility 不大於 60；2 個城鎮的 Education 小於 15，且 Fertility 不大於 60。

#### 4.1.4 Keying

Keys 是另一個選取子集 (subsetting) 更快的方法。只要在 data table 的某個變數中設定了 key，表格在物理上就會重新排列記憶體與儲存的 rows 所分派的順序。設定 key 的指令為 `setkey(data.table, key)`，如：

```
setkey(DT.swiss, Education)
```

之後，表格就會依據 Education 重新排列：

```

      rn Fertility Agriculture Examination Education Catholic Infant.Mortality
1:      1      72.5       71.2          12         1       2.40           21.0
2:      2      68.3       72.6          18         2      24.20           21.2
3:      3      75.5       85.9           3         2     99.71           15.1
4:      4      77.3       89.7           5         2    100.00           18.3
5:      5      65.0       55.1          14         3       4.52           22.4
6:      6      72.0       63.5           6         3       2.56           18.0
7:      7      79.4       64.9           7         3     98.22           20.2
...

```

想要 subsetting keyed variable，可以使用 `.()`，如下列出 Education 等於 3 的 rows：

```
DT.swiss[.(3)]
```

```
##           rn Fertility Agriculture Examination Education Catholic
## 1:      Moudon      65.0       55.1         14         3      4.52
## 2: Paysd'enhaut    72.0       63.5          6         3      2.56
## 3:      Monthey    79.4       64.9          7         3     98.22
## 4:       Sierre   92.2       84.6          3         3     99.46
##   Infant.Mortality
## 1:             22.4
## 2:             18.0
## 3:             20.2
## 4:             16.3
```

也可以輸入向量，如下列出 Education 等於 3 或 5 的 rows：

```
DT.swiss[.(c(3, 5))]
```

```
##           rn Fertility Agriculture Examination Education Catholic
## 1:      Moudon      65.0       55.1         14         3      4.52
## 2: Paysd'enhaut    72.0       63.5          6         3      2.56
## 3:      Monthey    79.4       64.9          7         3     98.22
## 4:       Sierre   92.2       84.6          3         3     99.46
## 5: Franches-Mnt   92.5       39.7          5         5     93.40
## 6:      Cossonay   61.7       69.3         22         5      2.82
##   Infant.Mortality
## 1:             22.4
## 2:             18.0
## 3:             20.2
## 4:             16.3
## 5:             20.2
## 6:             18.7
```

也可以與引數 `j` 和 `by` 搭配使用，如：

```
DT.swiss[.(1 : 2), !c("Agriculture", "Infant.Mortality")]
```

```
##           rn Fertility Examination Education Catholic
## 1:      Oron      72.5         12         1      2.40
## 2: Echallens    68.3         18         2     24.20
## 3:  Conthey    75.5          3         2     99.71
## 4:   Herens    77.3          5         2    100.00
```

與：

```
DT.swiss[.(3 : 6), mean(Fertility), by=Education]
```

```
##   Education    V1
## 1:         3 77.150
```

```
## 2:      NA      NA
## 3:      5 77.100
## 4:      6 71.075
```

#### 4.1.5 編輯表格 Updating by Reference

那如何編輯表格呢？使用 `:=`，之前放 column 的名字，而之後放要指派的值。如果要新增兩個 column，可以：

```
DT.swiss[, c("new.col.1", "new.col.2"):=list(1 : 47, 51 : 97)]
```

其結果為：

	rn	Fertility	Agriculture	Examination	Education	Catholic	Infant.Mortality	new.col.1
1:	Oron	72.5	71.2	12	1	2.40	21.0	1
2:	Echallens	68.3	72.6	18	2	24.20	21.2	2
3:	Conthey	75.5	85.9	3	2	99.71	15.1	3
4:	Herens	77.3	89.7	5	2	100.00	18.3	4
5:	Moudon	65.0	55.1	14	3	4.52	22.4	5
...								
46:	Neuchatel	64.4	17.6	35	32	16.92	23.0	46
47:	V. De Geneve	35.0	1.2	37	53	42.34	18.0	47

修改其值也如同剛才的做法：

```
DT.swiss[, c("new.col.1", "new.col.2"):=list(101 : 147, 151 : 197)]
```

結果為：

	rn	Fertility	Agriculture	Examination	Education	Catholic	Infant.Mortality	new.col.1
1:	Oron	72.5	71.2	12	1	2.40	21.0	101
2:	Echallens	68.3	72.6	18	2	24.20	21.2	102
3:	Conthey	75.5	85.9	3	2	99.71	15.1	103
4:	Herens	77.3	89.7	5	2	100.00	18.3	104
5:	Moudon	65.0	55.1	14	3	4.52	22.4	105
...								
46:	Neuchatel	64.4	17.6	35	32	16.92	23.0	146
47:	V. De Geneve	35.0	1.2	37	53	42.34	18.0	147

如果要刪除既存的 column，則可以：

```
DT.swiss[, c("new.col.1", "new.col.2"):=list(NULL, NULL)]
```

## 4.2 MERGING

如果兩個 data table 有一樣的變數，那要合併可以使用 `rbind()` (row binding)，如：

```
dataset.1 <- data.table(city=c("Large", "Medium"), population=c(1000000, 250000), km2=c(20, 7))
dataset.2 <- data.table(city=c("Small"), population=c(50000), km2=c(1))
```

```
dataset.final <- rbind(dataset.1, dataset.2)
dataset.final
```

```
##      city population km2
## 1: Large    1000000  20
## 2: Medium    250000   7
## 3: Small     50000   1
```

如果有相同多的觀察值，則使用 `cbind()` (column binding)，例如：

```
dataset.1 <- data.table(city=c("Large", "Medium", "Small"), population=c(1000000, 250000, 50000), km2=c(20, 7, 1))
dataset.2 <- data.table(km2=c(20, 7, 1))
dataset.final <- cbind(dataset.1, dataset.2)
dataset.final
```

```
##      city population km2
## 1: Large    1000000  20
## 2: Medium    250000   7
## 3: Small     50000   1
```

如果兩個表格共享某些 rows 與 columns，則可以使用 `merge()`，如：

```
dataset.1 <- data.table(city=c("city.1", "city.2", "city.3", "city.4", "city.5", "city.6"),
                        population=c(10000, 20000, 100000, 5000, 30000, 65000),
                        km2=c(1, 0.5, 0.9, 2, 1.2, 3))
dataset.2 <- data.table(city=c("city.1", "city.2", "city.3", "city.7"),
                        airport=c(FALSE, FALSE, TRUE, TRUE))
dataset.1
```

```
##      city population km2
## 1: city.1     10000 1.0
## 2: city.2     20000 0.5
## 3: city.3    100000 0.9
## 4: city.4      5000 2.0
## 5: city.5     30000 1.2
## 6: city.6     65000 3.0
```

```
dataset.2
```

```
##      city airport
## 1: city.1  FALSE
## 2: city.2  FALSE
## 3: city.3   TRUE
## 4: city.7   TRUE
```

*# inner join: 包含所有 columns，但有缺漏值的 rows 會被跳過*  
`merge(dataset.1, dataset.2)`



```
##      city population km2 airport
## 1: city.1      1e+04 1.0  FALSE
## 2: city.2      2e+04 0.5  FALSE
## 3: city.3      1e+05 0.9   TRUE

# full join: 包含所有 columns 與 rows
merge(dataset.1, dataset.2, all = TRUE)
```

```
##      city population km2 airport
## 1: city.1      10000 1.0  FALSE
## 2: city.2      20000 0.5  FALSE
## 3: city.3     100000 0.9   TRUE
## 4: city.4       5000 2.0    NA
## 5: city.5      30000 1.2    NA
## 6: city.6      65000 3.0    NA
## 7: city.7        NA  NA    TRUE

# left join: 包含所有 columns, 但跳過第一個表格沒有的 rows
merge(dataset.1, dataset.2, all.x = TRUE)
```

```
##      city population km2 airport
## 1: city.1      10000 1.0  FALSE
## 2: city.2      20000 0.5  FALSE
## 3: city.3     100000 0.9   TRUE
## 4: city.4       5000 2.0    NA
## 5: city.5      30000 1.2    NA
## 6: city.6      65000 3.0    NA

# right join: 包含所有 columns, 但跳過第二個表格沒有的 rows
merge(dataset.1, dataset.2, all.y = TRUE)
```

```
##      city population km2 airport
## 1: city.1      1e+04 1.0  FALSE
## 2: city.2      2e+04 0.5  FALSE
## 3: city.3      1e+05 0.9   TRUE
## 4: city.7        NA  NA    TRUE
```

## 4.3 實例

除了前章所介紹的指令，本節還會介紹用以：

- 資料清理的 `na.omit()`、`duplicated()`；
- 文字轉換的 `substr()`、`tstrplit()`、`grepl()`。

`na.omit()` 可以剔除表格中有 NA 的觀察值 (rows)；`duplicated()` 則可以幫助我們刪除重複的觀察值。例如我們新建一個 data table：

```
dupli.table <- data.table(ID=c("E123456789", "N123456789", "N213456897", "N123456789"),
```

使用 `na.omit()` 即會刪除有 NA 的觀察值：

```
dupli.table <- na.omit(dupli.table)
dupli.table
```

```
##           ID sex major
## 1: E123456789  M  Econ
## 2: N213456897  F  Math
## 3: N123456789  M   OR
## 4: A123456789  M   EE
## 5: N213456897  F   CS
## 6: E213456897  F   Eng
```

使用 `duplicated()` 可以協助我們找到特定的 columns 重複的觀察值，例如：

```
duplicated.rows1 <- duplicated(dupli.table[, c(1, 3)]) # 設定第一個與第三個 cols
# 得到 "FALSE FALSE FALSE FALSE FALSE FALSE"
# 表示完全沒有在第一個與第三個 cols 都重複的觀察值
```

```
duplicated.rows2 <- duplicated(dupli.table[, c(1, 2)]) # 設定第一個與第二個 cols
# 得到 "FALSE FALSE FALSE FALSE TRUE FALSE"
# 表示第五個觀察值重複了
```

```
duplicated.rows3 <- duplicated(dupli.table[, 1]) # 設定第一個 cols
# 等價於 duplicated.rows3 <- duplicated(dupli.table$ID)
# 得到 "FALSE FALSE FALSE FALSE TRUE FALSE"
# 表示第五個觀察值重複了
```

那麼怎樣刪除重複的觀察值呢？我們可以：

```
dupli.table <- dupli.table[!duplicated.rows2]
dupli.table
```

```
##           ID sex major
## 1: E123456789  M  Econ
## 2: N213456897  F  Math
## 3: N123456789  M   OR
## 4: A123456789  M   EE
## 5: E213456897  F   Eng
```

### 4.3.1 TheSportsDB NBA Dataset Preprocessing

上一章使用：

```
library(rvest)
nbagames <- read_html("https://www.thesportsdb.com/season.php?l=4387&s=2020-2021")
```

```
games <- html_text(html_nodes(nbagames, "td"))
```

得到的 games 儲存了許多無用的字元：

[illegible]

首先我們要先把空的項目刪除：

```
games <- games[!games==""]
```

接著，我們想把包含 \n 的項目刪除。但是，他們都長得不太一樣，於是我們要以 `grepl(text-to-find, where-to-find-it)` 來刪除：

```
games <- games[!grepl("\\n",games)]
```

現在這樣就正常很多了。而現在儲存的方式是向量，我們可以將其轉成有 4 個 column 的矩陣，然後將其轉換成 data table，並冠上 column names：

```
games.2021 <- as.data.table(matrix(games, ncol=4, byrow=T))
colnames(games.2021) <- c("Date", "TeamA", "Result", "TeamB")
```

不過，比分還是無法計算，因為現在是兩個數字。`data.table` 中的函數 `tstrsplit(variable, separator, keep)` 可以用以切開資訊，如：

```
games.2021[, tstrsplit(Result, "-")]
# 相當於 games.2021[, tstrsplit(Result, " ", keep = c(1, 3))]
```

	V1	V2
1:	112	116
2:	84	90
3:	104	125
4:	87	81
5:	127	102
---		
1217:	118	108
1218:	120	100
1219:	109	103
1220:	119	123
1221:	105	98

因此，我們如果要新增兩個 columns，分別是 A 隊與 B 隊的分數，可以：

```
games.2021[, c("PointsA", "PointsB"):=tstrsplit(Result, "-")]
games.2021$Result <- NULL
```

得：

	Date	TeamA	TeamB	PointsA	PointsB
1:	12 Dec 2020	Atlanta Hawks	Orlando Magic	112	116
2:	12 Dec 2020	Detroit Pistons	New York Knicks	84	90
3:	12 Dec 2020	Chicago Bulls	Houston Rockets	104	125
4:	12 Dec 2020	Los Angeles Lakers	Los Angeles Clippers	87	81
5:	12 Dec 2020	Portland Trail Blazers	Sacramento Kings	127	102
---					
1217:	09 Jul 2021	Phoenix Suns	Milwaukee Bucks	118	108
1218:	12 Jul 2021	Milwaukee Bucks	Phoenix Suns	120	100
1219:	15 Jul 2021	Milwaukee Bucks	Phoenix Suns	109	103
1220:	18 Jul 2021	Phoenix Suns	Milwaukee Bucks	119	123
1221:	21 Jul 2021	Milwaukee Bucks	Phoenix Suns	105	98

## (PART) 以 tidyverse 進行資料 探索

