# 第 15 天

## 資料處理技巧（2）

在[資料處理技巧](chapter14.md)我們已經對基礎的資料處理技巧駕輕就熟，在今天的章節我們想要討論一些更進階的資料處理技巧，包含如何使用 %>% 運算子、長寬表格的轉換、如何使用 dplyr 套件中的函數以及如何應用函數於資料框。

### tidyverse 套件

我們今天要使用幾個套件來輔助我們作進階的資料處理：

|  |  |  |
| --- | --- | --- |
| 套件 | 主要開發者 | 目的 |
| magrittr | Stefan Milton Bache | 能夠使用 %>% 運算子 |
| tidyr | Hadley Wickham | 能夠進行長寬表格的轉換 |
| dplyr | Hadley Wickham | 更有效率地作資料處理 |

我們當然可以分開安裝這些套件，但是我們另外介紹套件 tidyverse，它包含了上表我們要使用的這些套件，因此可以很方便地裝一個套件即可。我們可以透過命令列（Console）安裝：

> install.packages("tidyverse")

也可以透過 RStudio 介面安裝的步驟是在右下角的 **packages** 頁籤點選 **install**：

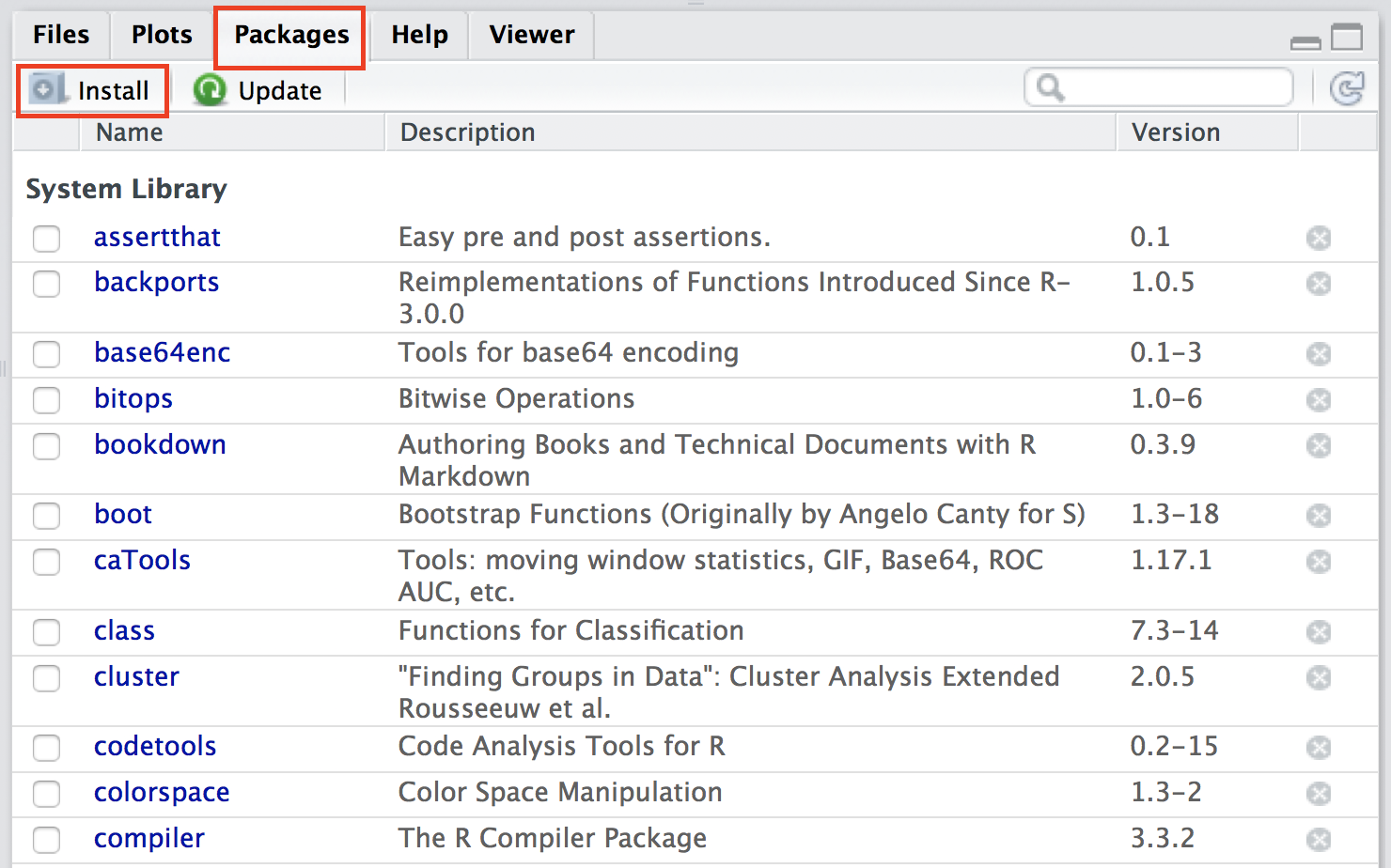


圖 15-1 點選 install

接著是輸入安裝套件的名稱：tidyverse。

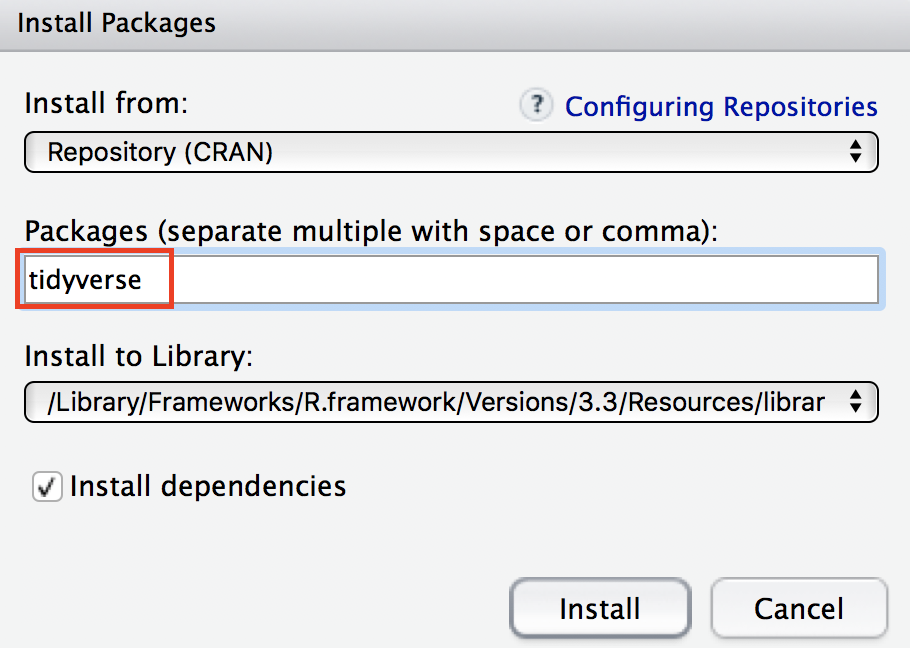


圖 15-2 輸入 tidyverse

### 載入 tidyverse 套件

接著我們可以透過命令列（Console）載入：

> library(tidyverse)

或者透過 RStudio 介面載入的方法是在右下角的 **packages** 頁籤下找到 tidyverse 然後將前面的核取方框打勾即可。

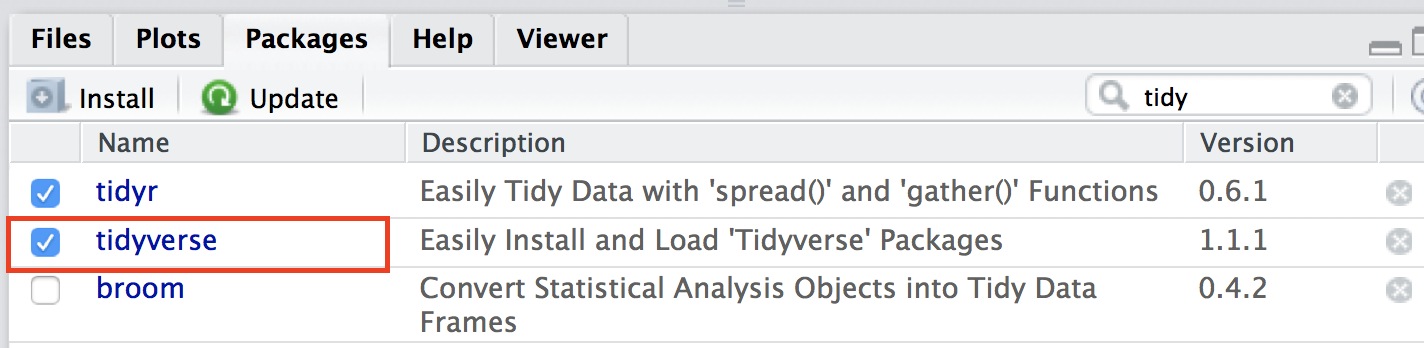


圖 15-3 載入 tidyverse

### %>% 運算子

#### 第一次使用 %>%

%>% 運算子稱作 **Pipeline**，它是進階的 R 語言使用者在需要呼叫多次函數時候會採用的運算子，它稍微修改了我們傳統呼叫函數的習慣，例如傳統我們想要將內建資料集 cars 作為 summary() 函數的輸入時：

> summary(cars) # 傳統呼叫函數  
 speed dist   
 Min. : 4.0 Min. : 2.00   
 1st Qu.:12.0 1st Qu.: 26.00   
 Median :15.0 Median : 36.00   
 Mean :15.4 Mean : 42.98   
 3rd Qu.:19.0 3rd Qu.: 56.00   
 Max. :25.0 Max. :120.00

我們是將輸入（inputs）放置在函數名稱的小括號中；但是在使用 %>% 運算子的時候，我們將輸入放置在 %>% 運算子的左邊，並且將函數放置在 %>% 運算子的右邊，看起來就像是把輸入丟入函數中的樣子：

> library(tidyverse)  
> cars %>% summary() # 使用 %>%  
 speed dist   
 Min. : 4.0 Min. : 2.00   
 1st Qu.:12.0 1st Qu.: 26.00   
 Median :15.0 Median : 36.00   
 Mean :15.4 Mean : 42.98   
 3rd Qu.:19.0 3rd Qu.: 56.00   
 Max. :25.0 Max. :120.00

#### %>% 的使用時機

那在什麼時候我們可以考慮不使用傳統呼叫函數的寫法改用 %>% 呢？舉例來說，R 語言的內建函數 Sys.Date() 會回傳一個系統日期：

Sys.Date()

假設我們現在有一個需求是把系統日期的年份擷取出來，並且轉換為數字，我們可能會這樣寫：

> sys\_date <- Sys.Date()  
> sys\_date\_yr <- format(sys\_date, format = "%Y")  
> sys\_date\_num <- as.numeric(sys\_date\_yr)  
> sys\_date\_num  
[1] 2017

這樣子寫的可讀性（readibility）很高，但是為了得到我們要的答案，過程中額外建立了 sys\_date 與 sys\_date\_yr 這兩個物件，覺得好像不是太有效率，那麼我們試著把上面這段程式改寫得精簡一點：

> sys\_date\_num <- as.numeric(format(Sys.Date(), format = "%Y"))  
> sys\_date\_num  
[1] 2017

這樣子寫雖然精簡，但是可讀性就變得比較低，尤其是小括號很多，我們都不喜歡去檢查哪個左括號應該對應哪個右括號。在這種呼叫多次函數，並且需要將前一次函數的輸出作為後一次函數的輸入時，我們就應該想到 %>%：

> library(tidyverse)  
>  
> sys\_date\_num <- Sys.Date() %>%  
+ format(format = "%Y") %>%  
+ as.numeric()  
> sys\_date\_num  
[1] 2017

這樣子的寫法就兼顧了可讀性高與精簡的兩個優點！

#### 加入運算符號

在使用 %>% 運算子將多個函數呼叫串連的流程中，也可以加入運算符號，舉例來說，如果我們想要計算香港搖滾樂隊 Beyond 成立幾週年紀念，可以用系統日期的年份減去成立年份：

> beyond\_start <- 1983  
> beyond\_yr <- Sys.Date() %>%  
+ format(format = "%Y") %>%  
+ as.numeric() %>%  
+ `-` (beyond\_start)  
> beyond\_yr  
[1] 34

我們將運算符號放入 `` 之中，這個符號叫做 tilt，您可以在鍵盤最左上角的按鍵找到它，它在 tab 的上方，數字 1 的左方，然後是將要運算的數字放入小括號中。

#### 調整輸入的位置

%>% 預設會將輸入放在函數輸入的第一個位置，在某些函數第一個位置不是輸入時，像是 lm() 函數的第一個位置必須註明方程式（formula），就可以透過 . 來指定輸入放置的地方：

> # 傳統呼叫 lm() 函數  
> cars\_lm <- lm(formula = dist ~ speed, data = cars)  
>   
> # 使用 . 指定 cars 放的位置  
> cars\_lm <- cars %>%  
+ lm(formula = dist ~ speed, data = .)

### 長寬表格的轉換

#### 寬轉長表格

透過 tidyr 套件中的 gather() 函數能夠將多個數值變數堆積在同一個數值變數（value）中，並且再使用一個類別變數（key）來記錄每一個數值變數的來源。舉例來說，我們原先用兩個變數 wins 與 losses 來記錄 1995-96 球季芝加哥公牛隊與 2015-16 球季金州勇士隊的勝場數與敗場數：

> team\_name <- c("Chicago Bulls", "Golden State Warriors")  
> wins <- c(72, 73)  
> losses <- c(10, 9)  
>   
> great\_nba\_teams <- data.frame(team\_name, wins, losses)  
> great\_nba\_teams  
 team\_name wins losses  
1 Chicago Bulls 72 10  
2 Golden State Warriors 73 9

這樣外觀的資料框我們稱之為寬表格，在寬表格的結構中，如果我們希望增加一個變數，例如像是勝率，會以**增加欄位數**的方式來更新：

> great\_nba\_teams <- data.frame(team\_name, wins, losses)  
> great\_nba\_teams$winning\_percentage <- great\_nba\_teams$wins / (great\_nba\_teams$wins + great\_nba\_teams$losses)  
> great\_nba\_teams  
 team\_name wins losses winning\_percentage  
1 Chicago Bulls 72 10 0.8780488  
2 Golden State Warriors 73 9 0.8902439

透過 tidyr 套件中的 gather() 函數可以將這樣外觀的寬表格轉換為長表格：

> library(tidyverse)  
>  
> team\_name <- c("Chicago Bulls", "Golden State Warriors")  
> wins <- c(72, 73)  
> losses <- c(10, 9)  
>   
> great\_nba\_teams <- data.frame(team\_name, wins, losses)  
> gather(great\_nba\_teams, key = variable\_names, value = values, wins, losses)  
 team\_name variable\_names values  
1 Chicago Bulls wins 72  
2 Golden State Warriors wins 73  
3 Chicago Bulls losses 10  
4 Golden State Warriors losses 9

其中參數 key 是指定記錄數值來源的類別變數名稱為何，我們命名作 variable\_names；參數 value 是指定堆積數值的變數名稱為何，指定好這兩個參數以後，則是再指定有哪些變數要堆積起來

#### 長轉寬表格

透過 tidyr 套件中的 spread() 函數能夠將前述的長表格再轉換為原本的寬表格：

> library(tidyverse)  
>  
> team\_name <- c("Chicago Bulls", "Golden State Warriors")  
> wins <- c(72, 73)  
> losses <- c(10, 9)  
>   
> great\_nba\_teams <- data.frame(team\_name, wins, losses)  
> long\_format <- gather(great\_nba\_teams, key = variable\_names, value = values, wins, losses)  
> spread(long\_format, key = variable\_names, value = values)  
 team\_name losses wins  
1 Chicago Bulls 10 72  
2 Golden State Warriors 9 73

### dplyr 套件

接著我們要介紹的是 dplyr() 套件，相較於原生的資料處理語法，dplyr() 套件中融入很多概念與**結構化查詢語言（Structured Query Language，SQL）**相仿的函數。搭配 %>% 運算子一起使用，能夠讓我們整理資料的能力獲得一個檔次的提升！我們將 dplyr() 套件提供的常用函數整理如下表：

|  |  |
| --- | --- |
| 函數 | 用途 |
| filter() | 篩選符合條件的觀測值 |
| select() | 選擇變數 |
| mutate() | 新增變數 |
| arrange() | 依照變數排序觀測值 |
| summarise() | 聚合變數 |
| group\_by() | 依照類別變數分組，搭配 |

#### 使用 filter() 函數

在 filter() 函數中我們輸入要篩選的資料框，以及依據什麼條件進行篩選，舉例來說我們可以將 straw\_hat\_df 中的女性篩選出來：

> library(tidyverse)  
>  
> name <- c("蒙其·D·魯夫", "羅羅亞·索隆", "娜美", "騙人布", "賓什莫克·香吉士", "多尼多尼·喬巴", "妮可·羅賓", "佛朗基", "布魯克")  
> gender <- c("男", "男", "女", "男", "男", "男", "女", "男", "男")  
> age <- c(19, 21, 20, 19, 21, 17, 30, 36, 90)  
> straw\_hat\_df <- data.frame(name, gender, age, stringsAsFactors = FALSE)  
> filter(straw\_hat\_df, gender == "女")  
 name gender age  
1 娜美 女 20  
2 妮可·羅賓 女 30

我們也藉此機會對照之前學習的 R 語言原生寫法：

> name <- c("蒙其·D·魯夫", "羅羅亞·索隆", "娜美", "騙人布", "賓什莫克·香吉士", "多尼多尼·喬巴", "妮可·羅賓", "佛朗基", "布魯克")  
> gender <- c("男", "男", "女", "男", "男", "男", "女", "男", "男")  
> age <- c(19, 21, 20, 19, 21, 17, 30, 36, 90)  
> straw\_hat\_df <- data.frame(name, gender, age, stringsAsFactors = FALSE)  
> straw\_hat\_df[straw\_hat\_df$gender == "女", ]  
 name gender age  
3 娜美 女 20  
7 妮可·羅賓 女 30

在 filter() 函數中我們可以利用 & 以及 | 連結多個篩選條件，舉例來說我們可以將 straw\_hat\_df 中超過 30 歲的男性篩選出來：

> library(tidyverse)  
>  
> name <- c("蒙其·D·魯夫", "羅羅亞·索隆", "娜美", "騙人布", "賓什莫克·香吉士", "多尼多尼·喬巴", "妮可·羅賓", "佛朗基", "布魯克")  
> gender <- c("男", "男", "女", "男", "男", "男", "女", "男", "男")  
> age <- c(19, 21, 20, 19, 21, 17, 30, 36, 90)  
> straw\_hat\_df <- data.frame(name, gender, age, stringsAsFactors = FALSE)  
> filter(straw\_hat\_df, gender == "男" & age > 30)  
 name gender age  
1 佛朗基 男 36  
2 布魯克 男 90

我們也藉此機會對照之前學習的 R 語言原生寫法：

> name <- c("蒙其·D·魯夫", "羅羅亞·索隆", "娜美", "騙人布", "賓什莫克·香吉士", "多尼多尼·喬巴", "妮可·羅賓", "佛朗基", "布魯克")  
> gender <- c("男", "男", "女", "男", "男", "男", "女", "男", "男")  
> age <- c(19, 21, 20, 19, 21, 17, 30, 36, 90)  
> straw\_hat\_df <- data.frame(name, gender, age, stringsAsFactors = FALSE)  
> straw\_hat\_df[(straw\_hat\_df$gender == "男") & (straw\_hat\_df$age > 30), ]  
 name gender age  
8 佛朗基 男 36  
9 布魯克 男 90

#### 使用 select() 函數

在 select() 函數中我們輸入資料框的名稱，以及想要選取的變數名稱，舉例來說我們可以將 straw\_hat\_df 中的 name 篩選出來：

> library(tidyverse)  
>  
> name <- c("蒙其·D·魯夫", "羅羅亞·索隆", "娜美", "騙人布", "賓什莫克·香吉士", "多尼多尼·喬巴", "妮可·羅賓", "佛朗基", "布魯克")  
> gender <- c("男", "男", "女", "男", "男", "男", "女", "男", "男")  
> age <- c(19, 21, 20, 19, 21, 17, 30, 36, 90)  
> straw\_hat\_df <- data.frame(name, gender, age, stringsAsFactors = FALSE)  
> select(straw\_hat\_df, name)  
 name  
1 蒙其·D·魯夫  
2 羅羅亞·索隆  
3 娜美  
4 騙人布  
5 賓什莫克·香吉士  
6 多尼多尼·喬巴  
7 妮可·羅賓  
8 佛朗基  
9 布魯克

select() 函數選擇單一變數時並不會像 R 語言原生語法預設轉換為向量資料結構，而是維持原本的資料框資料結構，在原生語法中我們可以在中括號中多指定 drop = FALSE 達到這個效果：

> name <- c("蒙其·D·魯夫", "羅羅亞·索隆", "娜美", "騙人布", "賓什莫克·香吉士", "多尼多尼·喬巴", "妮可·羅賓", "佛朗基", "布魯克")  
> gender <- c("男", "男", "女", "男", "男", "男", "女", "男", "男")  
> age <- c(19, 21, 20, 19, 21, 17, 30, 36, 90)  
> straw\_hat\_df <- data.frame(name, gender, age, stringsAsFactors = FALSE)  
> straw\_hat\_df[, "name", drop = FALSE]  
 name  
1 蒙其·D·魯夫  
2 羅羅亞·索隆  
3 娜美  
4 騙人布  
5 賓什莫克·香吉士  
6 多尼多尼·喬巴  
7 妮可·羅賓  
8 佛朗基  
9 布魯克

select() 函數也可以在選擇變數的同時對變數重新進行命名：

> library(tidyverse)  
>  
> name <- c("蒙其·D·魯夫", "羅羅亞·索隆", "娜美", "騙人布", "賓什莫克·香吉士", "多尼多尼·喬巴", "妮可·羅賓", "佛朗基", "布魯克")  
> gender <- c("男", "男", "女", "男", "男", "男", "女", "男", "男")  
> age <- c(19, 21, 20, 19, 21, 17, 30, 36, 90)  
> straw\_hat\_df <- data.frame(name, gender, age, stringsAsFactors = FALSE)  
> select(straw\_hat\_df, crew\_name = name, crew\_age = age)  
 crew\_name crew\_age  
1 蒙其·D·魯夫 19  
2 羅羅亞·索隆 21  
3 娜美 20  
4 騙人布 19  
5 賓什莫克·香吉士 21  
6 多尼多尼·喬巴 17  
7 妮可·羅賓 30  
8 佛朗基 36  
9 布魯克 90

#### 使用 mutate() 函數

利用 mutate() 函數新增衍生變數或者非衍生變數相當簡潔，舉例來說，我們要在 great\_nba\_teams 資料框中新增衍生變數 winning\_percentage 與非衍生變數 season：

> library(tidyverse)  
>  
> team\_name <- c("Chicago Bulls", "Golden State Warriors")  
> wins <- c(72, 73)  
> losses <- c(10, 9)  
> season <- c("1995-96", "2015-16")  
> great\_nba\_teams <- data.frame(team\_name, wins, losses, stringsAsFactors = FALSE)  
> mutate(great\_nba\_teams,  
+ winning\_percentage = wins / (wins + losses),  
+ season = season  
+ )  
 team\_name wins losses winning\_percentage season  
1 Chicago Bulls 72 10 0.8780488 1995-96  
2 Golden State Warriors 73 9 0.8902439 2015-16

#### 使用 arrange() 函數

利用指定的變數來排序觀測值，舉例來說以 age 變數來排序 straw\_hat\_df：

> library(tidyverse)  
>  
> name <- c("蒙其·D·魯夫", "羅羅亞·索隆", "娜美", "騙人布", "賓什莫克·香吉士", "多尼多尼·喬巴", "妮可·羅賓", "佛朗基", "布魯克")  
> gender <- c("男", "男", "女", "男", "男", "男", "女", "男", "男")  
> age <- c(19, 21, 20, 19, 21, 17, 30, 36, 90)  
> straw\_hat\_df <- data.frame(name, gender, age, stringsAsFactors = FALSE)  
> arrange(straw\_hat\_df, age)  
 name gender age  
1 多尼多尼·喬巴 男 17  
2 蒙其·D·魯夫 男 19  
3 騙人布 男 19  
4 娜美 女 20  
5 羅羅亞·索隆 男 21  
6 賓什莫克·香吉士 男 21  
7 妮可·羅賓 女 30  
8 佛朗基 男 36  
9 布魯克 男 90

如果想改為遞減排序，就在變數外面增加 desc() 函數：

> library(tidyverse)  
>  
> name <- c("蒙其·D·魯夫", "羅羅亞·索隆", "娜美", "騙人布", "賓什莫克·香吉士", "多尼多尼·喬巴", "妮可·羅賓", "佛朗基", "布魯克")  
> gender <- c("男", "男", "女", "男", "男", "男", "女", "男", "男")  
> age <- c(19, 21, 20, 19, 21, 17, 30, 36, 90)  
> straw\_hat\_df <- data.frame(name, gender, age, stringsAsFactors = FALSE)  
> arrange(straw\_hat\_df, desc(age))  
 name gender age  
1 布魯克 男 90  
2 佛朗基 男 36  
3 妮可·羅賓 女 30  
4 羅羅亞·索隆 男 21  
5 賓什莫克·香吉士 男 21  
6 娜美 女 20  
7 蒙其·D·魯夫 男 19  
8 騙人布 男 19  
9 多尼多尼·喬巴 男 17

#### 使用 summarise() 函數

在 summarise() 函數中我們輸入資料框的名稱，以及想要聚合的變數名稱，聚合運算的結果通常是一個數字，代表某個數列的運算結果，像是**總和**、**平均數**或**標準差**都是聚合運算的結果，舉例來說，我們可以運算 straw\_hat\_df 所有成員的平均年齡：

> library(tidyverse)  
>  
> name <- c("蒙其·D·魯夫", "羅羅亞·索隆", "娜美", "騙人布", "賓什莫克·香吉士", "多尼多尼·喬巴", "妮可·羅賓", "佛朗基", "布魯克")  
> gender <- c("男", "男", "女", "男", "男", "男", "女", "男", "男")  
> age <- c(19, 21, 20, 19, 21, 17, 30, 36, 90)  
> straw\_hat\_df <- data.frame(name, gender, age, stringsAsFactors = FALSE)  
> summarise(straw\_hat\_df, mean(age))  
 mean(age)  
1 30.33333

#### 使用 group\_by() 函數

聚合函數的運算常常會搭配 group\_by() 函數一起使用，這時我們就可以整合 %>% 運算子，舉例來說，我們可以計算 straw\_hat\_df 中男性與女性的平均年齡：

> library(tidyverse)  
>  
> name <- c("蒙其·D·魯夫", "羅羅亞·索隆", "娜美", "騙人布", "賓什莫克·香吉士", "多尼多尼·喬巴", "妮可·羅賓", "佛朗基", "布魯克")  
> gender <- c("男", "男", "女", "男", "男", "男", "女", "男", "男")  
> age <- c(19, 21, 20, 19, 21, 17, 30, 36, 90)  
> straw\_hat\_df <- data.frame(name, gender, age, stringsAsFactors = FALSE)  
> group\_by(straw\_hat\_df, gender) %>%  
+ summarise(mean(age)) %>%  
+ as.data.frame()  
 gender mean(age)  
1 女 25.00000  
2 男 31.85714

這裡我們必須記得先呼叫 group\_by() 函數然後再呼叫 summarise() 函數，這個順序與撰寫 **SQL** 語法相反，熟練 **SQL** 語法的使用者需要特別留意這個差異。在連結 group\_by() 與 summarise() 兩個函數之後的輸出是一種叫做 **tibble** 的改良式資料框，為了不要增添初學者的負擔，我們不說明它跟原生資料框的差異，**tibble** 可以利用 as.data.frame() 函數轉換為原生的資料框。

### 應用函數於資料框

還記得我們在[迴圈與流程控制](chapter9.md)探討了 for 與 while 迴圈嗎？資料量小的時候使用迴圈語法不太會感受到它們較低的運行效率，但是在資料量大的時候會感受到它較慢的運行效率，舉例來說我們先來隨機產生 50 萬人的身高體重：

> heights <- runif(500000) \* 50 + 140  
> weights <- runif(500000) \* 50 + 40  
> h\_w\_df <- data.frame(heights, weights)

假如我們現在利用迴圈計算這 50 萬人的 BMI：

> heights <- ceiling(runif(500000) \* 50) + 140  
> weights <- ceiling(runif(500000) \* 50) + 40  
> h\_w\_df <- data.frame(heights, weights)  
> bmi <- rep(NA, times = nrow(h\_w\_df))  
> for (i in 1:nrow(h\_w\_df)){  
+ bmi[i] <- h\_w\_df[i, "weights"] / (h\_w\_df[i, "heights"]/100)^2  
+ }

執行上面這段程式就明顯地發現了迴圈執行蠻久的一段時間，我們可以用 system.time() 這個函數得知執行時間：

> heights <- ceiling(runif(500000) \* 50) + 140  
> weights <- ceiling(runif(500000) \* 50) + 40  
> h\_w\_df <- data.frame(heights, weights)  
> bmi <- rep(NA, times = nrow(h\_w\_df))  
> system.time(  
+ for (i in 1:nrow(h\_w\_df)){  
+ bmi[i] <- h\_w\_df[i, "weights"] / (h\_w\_df[i, "heights"]/100)^2  
+ }  
+ )  
 user system elapsed   
 12.521 0.039 12.583

大概花了 13 秒左右的時間，假如直接用兩個向量來做運算：

> heights <- ceiling(runif(500000) \* 50) + 140  
> weights <- ceiling(runif(500000) \* 50) + 40  
> h\_w\_df <- data.frame(heights, weights)  
> bmi <- h\_w\_df$weights / (h\_w\_df$heights / 100)^2

哇！這個運算時間也太短了，看一下執行時間：

> heights <- ceiling(runif(500000) \* 50) + 140  
> weights <- ceiling(runif(500000) \* 50) + 40  
> h\_w\_df <- data.frame(heights, weights)  
> system.time(  
+ bmi <- h\_w\_df$weights / (h\_w\_df$heights / 100)^2  
+ )  
 user system elapsed   
 0.003 0.000 0.004

比迴圈快太多了吧！這就是為什麼很多資深的 R 語言使用者會說用**向量計算**優先於**迴圈**的原因，那假如這個需求不是向量計算能解決的呢？這時候資深 R 語言使用者會告訴我們用一系列的 apply() 函數來解決這個問題！舉例來說，假如我們想要知道內建資料 iris 中每個變數的相異值個數：

> distinct\_counts <- function(x){  
+ unique\_values <- unique(x)  
+ return(length(unique\_values))  
+ }  
> apply(iris, MARGIN = 2, distinct\_counts)  
Sepal.Length Sepal.Width Petal.Length Petal.Width Species   
 35 23 43 22 3

其中 MARGIN = 2 參數是指定我們自訂的 distinct\_counts() 函數是應用在變數（Column）的方向上，如果想要應用在觀測值（Row）的方向上，我們必須指定 MARGIN = 1。

除了最基本的 apply() 函數以外，這一系列函數還有很多種的形式，例如 lapply() 函數會將輸出儲存為清單這樣的資料結構：

> distinct\_counts <- function(x){  
+ unique\_values <- unique(x)  
+ return(length(unique\_values))  
+ }  
> lapply(iris, FUN = distinct\_counts)  
$Sepal.Length  
[1] 35  
  
$Sepal.Width  
[1] 23  
  
$Petal.Length  
[1] 43  
  
$Petal.Width  
[1] 22  
  
$Species  
[1] 3

值得注意的是，不同形式的 apply() 函數有著自己的屬性，像是 lapply() 函數只能針對變數（Column）方向應用函數，不像 apply() 函數可以指定 MARGIN 參數。

sapply() 函數屬於簡化的 lapply() 函數，它回傳的結果是向量而非清單，在我們範例中使用的效果就像 apply() 函數指定 MARGIN = 2 一樣：

> distinct\_counts <- function(x){  
+ unique\_values <- unique(x)  
+ return(length(unique\_values))  
+ }  
> sapply(iris, FUN = distinct\_counts)  
Sepal.Length Sepal.Width Petal.Length Petal.Width Species   
 35 23 43 22 3

tapply() 函數是融入 table() 函數功能的形式，舉例來說我們可以拆分不同品種的 Species 來計算 Petal.Length 變數的相異個值：

> distinct\_counts <- function(x){  
+ unique\_values <- unique(x)  
+ return(length(unique\_values))  
+ }  
> tapply(iris$Petal.Length, INDEX = iris$Species, FUN = distinct\_counts)  
 setosa versicolor virginica   
 9 19 20

這裡我們講的 apply()、lapply()、sapply() 與 tapply() 函數是比較常見的 apply() 函數成員，未來當您有更特殊的需求時，還會再碰到它們的親朋好友。

### 小結

好啦！第十五天的內容就是這麼多，我們今天介紹了三個很實用的套件，並且用 tidyverse 一次就全部載入，%>% 運算子可以在不降低可讀性的前提下簡化多層的函數呼叫；tidyr 套件中提供兩個函數讓我們可以彈性地轉換長寬表格；dplyr 套件讓我們處理資料的效率更高。最後我們簡單討論了關於程式執行的效率，當您之後編寫 R 語言的經驗愈來愈多，開始注重程式執行效率時，記得優先考慮向量運算，然後是 apply() 形式的各種函數，最後才是撰寫迴圈。

### 練習題

###### 延續今天所舉的 50 萬筆的身高體重資料，請分別用向量運算、mapply() 函數與迴圈來計算 BMI，並且都以 system.time() 函數觀察執行時間。

heights <- ceiling(runif(500000) \* 50) + 140  
weights <- ceiling(runif(500000) \* 50) + 40  
h\_w\_df <- data.frame(heights, weights)

### 延伸資訊

* [Package magrittr](https://cran.r-project.org/web/packages/magrittr/magrittr.pdf)
* [Package tidyr](https://cran.r-project.org/web/packages/tidyr/tidyr.pdf)
* [Package dplyr](https://cran.r-project.org/web/packages/dplyr/dplyr.pdf)
* [The magrittr package](https://cran.r-project.org/web/packages/magrittr/vignettes/magrittr.html)
* [Introducing tidyr](https://blog.rstudio.org/2014/07/22/introducing-tidyr/)
* [Introduction to dplyr](https://cran.r-project.org/web/packages/dplyr/vignettes/introduction.html)