

# R03：R 語言大資料資料分析實作

## 六、探索式資料分析

### (一)、什麼是探索式資料分析

探索式資料分析 (Exploratory Data Analysis) 的主要精神是運用視覺化、基本的統計等工具，反覆的探索資料特性，獲取資料所包含的資訊、結構和特點，因為在進行複雜或嚴謹的分析之前，必須要對資料有更多認識，才能訂定對的資料分析方向。

探索式資料分析包括分析各變數間的關聯性，看是否有預料之外的有趣發現，或是觀察資料內容是否符合預期。若否，檢查資料是否有誤，最後檢查資料是否符合分析前的假設。由上述可知，探索式資料分析通常不需要嚴謹的假設和細節呈現，主要功能還是「觀察」資料的特性。在資料量大/雜的時候，探索式資料分析就非常重要，因為透過探索式資料分析，分析人員可以在複雜的統計計算與耗時的模型建立前，就先發現可能的錯誤，更重要的是，可以透過探索性分析來調整分析的方向，減少因分析方向錯誤所造成的時間浪費。

探索式資料分析分為：

- 圖形化 (Graphical) 或量化 (Quantitative)
- 單變量 (Univariate) 或雙變量 (Bivariate) 或多變量 (Multivariate)

圖形化的分析方式包括做圖與清單，量化的分析方式則是資料初步統計，本單元著重於量化的分析方式。以單變量分析來說，量化的分析方式可包含：

- 計算集中趨勢：
  - 平均值 (Mean)：mean()
  - 中位數 (Median)：median()
  - 眾數 (Mode)，R 無內建函數，可直接用 table() 找出現次數最多的資料
- 計算資料分散程度：
  - 最小值 Min)：min()
  - 最大值 Max)：max()
  - 範圍 Range)：range()
  - 四分位差 (Quartiles)：quantile()
  - 變異數 (Variance)：var()
  - 標準差 (Standard deviation)：sd()
- 以雙變量分析來說，分析方式可包括：
  - 列聯表 (Crosstabs)：table()、ftable()、prop.table()
  - 共變數 (Covariance)：cov()
  - 相關性 (Correlation)：cor()

量化分析方式的測量值大多可用 R 的內建函數完成計算，但是在探索式分析時，常常需要遇到資料分組的分析情形（如觀察男性和女性的 BMI 差異、A 隊與 B 隊的三分球命中率差異、中鋒和後衛的助攻次數…等），若只用基本的內建函數計算，需要先完成資料分組或子集後，再作進一步的運算，相當耗時，為了使這類資料分組與分析的工作更容易被完成，本單元在介紹探索式資料分析時會搭配介紹 **dplyr** 套件（Package）。

## （二）、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 `%>%` 運算元一起使用，能夠提升處理資料的能力。常用的 dplyr 套件函數整理如下表：

函數	用 途
<code>filter()</code>	選要分析的觀察值，觀察值子集 (Row)
<code>select()</code>	選要分析的欄位，欄位子集(Column)
<code>mutate()</code>	增加新欄位
<code>arrange()</code>	觀察值排序
<code>summarise()</code>	計算統計值
<code>group_by()</code>	依照類別變數分組、搭配
<code>rename()</code>	欄位重新命名
<code>%&gt;%</code>	the "pipe" operator 連結上述函式，將所有函式計算串在一起執行

dplyr 的基本功能是 6 個能與 SQL 查詢語法相互呼應的函數：

- **filter()** 函數：SQL 查詢中的 where 描述。
- **select()** 函數：SQL 查詢中的 select 描述。
- **mutate()** 函數：SQL 查詢中的衍生欄位描述。
- **arrange()** 函數：SQL 查詢中的 order by 描述。
- **summarise()** 函數：SQL 查詢中的聚合函數描述。
- **group\_by()** 函數：SQL 查詢中的 group by 描述。

**【注意事項】**請先刪除「Ex08-樞紐分析表(2)-銷售分析.xlsx」的「金額」欄位！

```
> setwd('C:/R-Exercise')
> library('xlsx')
Loading required package: rJava
```

Loading required package: xlsxjars

```
> data <- read.xlsx("Ex08-樞紐分析表(2)-銷售分析.xlsx", sheetIndex=1, encoding="UTF-8")
```

```
> library(readxl)
```

```
> data <- read_excel("Ex08-樞紐分析表(2)-銷售分析.xlsx", sheet=1)
```

```
> fare <- as.Date(data$運費) - as.Date('1900-01-01') + 1
```

```
> data$運費 <- fare
```

```
> price <- as.Date(data$單價) - as.Date('1900-01-01') + 1
```

```
> data$單價 <- price
```

```
> View(data)
```

```
> write.table(data, file="SalesData.csv", sep="," , row.names=F, na="NA")
```

```
> dir()
```

```
> sales.data <- read.table("SalesData.csv", sep="," , header=TRUE)
```

```
> sales.data
```

- 使用 **filter()** 函數

在 filter() 函數中輸入要篩選的資料來源，以及依據什麼條件進行篩選，例如可以將 sales.data 中的「送貨方式」為「親送」的購買紀錄篩選出來：

```
> install.packages("tidyverse")
```

```
> library(tidyverse)
```

```
> filter(sales.data, 送貨方式=="親送")
```

在 select() 函式中可直接做變數計算後再篩選，例如可以將 sales.data 中的「單價\*數量 > 2500 元」的購買紀錄篩選出來：

```
> library(tidyverse)
```

```
> filter(sales.data, 單價*數量 > 2500)
```

在 filter() 函數中可以利用 **&** 以及 **|** 連結多個篩選條件，例如可以將 sales.data 中的「送貨城市」為「高雄市」而且「送貨方式」為「親送」的購買紀錄篩選出來：

```
> library(tidyverse)
```

```
> filter(sales.data, 送貨城市=="高雄市" & 送貨方式=="親送")
```

```
> library(tidyverse)
```

```
> filter(sales.data, 送貨城市=="高雄市" & 送貨方式=="親送" & 單價*數量*(1-折扣)>500)
```

在 filter() 函數中可以利用 **%in%** 設定多個篩選條件，例如可以將 sales.data 中的「顧客」名稱為「永大大企業」或「高上補習班」的購買紀錄篩選出來：

```
> library(tidyverse)
> filter(sales.data, 顧客 %in% c("永大大企業","高上補習班"))
```

- 使用 **select()** 函數

在 select() 函數中輸入資料來源的名稱，以及想要選取的變數名稱，例如可以將 sales.data 中的「顧客」、「產品」、「數量」篩選出來：

```
> library(tidyverse)
> select(sales.data, 顧客, 產品, 數量)
```

- 使用 **mutate()** 函數

使用 mutate() 增加新欄位，如需新增新欄位「金額」，欄位值為「單價\*數量\*(1-折扣)」，指令如下：

```
> newsales.data <- mutate(sales.data, 金額=單價*數量*(1-折扣))
> newsales.data$金額[1:10]
[1] 98.00 174.00 167.40 1696.00 77.00 1261.40 214.20 95.76 222.30 336.00
> newsales.data
```

- 使用 **summarise()** 函數

summarise() 函式用來計算統計值，像是紀錄筆數、不重複的訂單數、不重複的顧客數以及不重複的產品數等：

```
> library(tidyverse)
> X <- summarise(newsales.data, 紀錄筆數=n(), 訂單數=n_distinct(訂單號碼), 顧客數=n_distinct(顧客), 產品數=n_distinct(產品))
> X
  紀錄筆數  訂單數  顧客數  產品數
1     2153     830     89     76
```

- 使用 **group\_by()** 函數

group\_by() 函數的功能為設定分組依據，通常會與 summarise() 函式合併使用，例如計算每位「顧客」（以「顧客」做為分組依據）的訂單總數、購買總金額等：

```
> library(tidyverse)
> X <- group_by(newsales.data, 顧客) %>% summarise(訂單總數=n_distinct(訂單號碼), 購買總金額=sum(金額))
```

```
> head(X)
> arrange1[1:20, 1:3]
> View(X)
```

- 使用 **arrange()** 函數

arrange() 函數的功能為排序，預設為遞增排序：

```
> arrange1<-arrange(X, desc(購買總金額))
> arrange1[1:20, 1:3]
> View(arrange1)
```

## 七、可視化探索-各類圖形應用(繪圖功能及基本統計)

### (一)、常用的 R 繪圖程式

R 內建許多圖形工具函數，這些圖形工具可以顯示各種統計繪圖並且自建一些全新的圖。以下指令，可以先看看 R 的圖形示範。

```
> demo(graphics)
> demo(image)
```

R 圖形工具函數之特色是用同一個繪圖函數，對不同的類型物件，可以作出不同的圖形。圖形工具函數既可互動式使用，也可以批次處理使用。在許多情況下，互動式使用是最有效的。圖形工具函數所產生之圖形結果，無法指定成物件，必須送到圖形裝置 (Graphic device)，圖形裝置可以是一個視窗或某一特定格式之圖形檔案。R 有一系列圖形引數 (Graphical parameters)，這些圖形引數可以修改或制定所需的圖形環境。

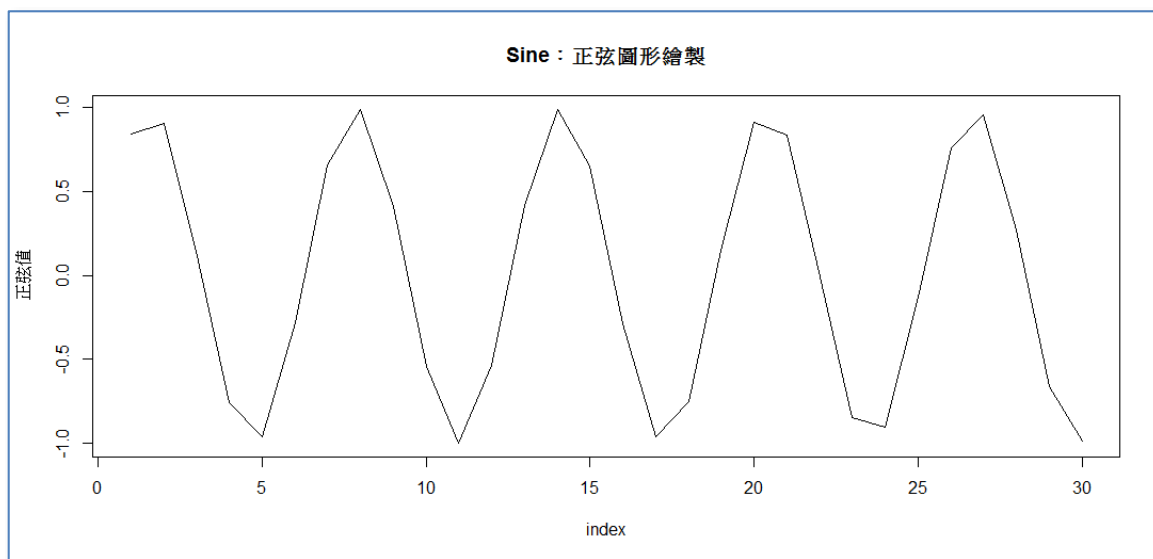
R 語言的繪圖指令可以分成了三個基本的類型：

1. 高階繪圖 (High-level Plotting Functions)：指令在圖形裝置上產生一個新的圖區，它可以包括坐標軸、標籤、標題等等。
2. 低階繪圖 (Low-level Plotting Functions)：指令會在一個已經存在的繪圖上，加上其它的圖形元素，如額外的點、線與標籤等等。
3. 互動式繪圖 (Interactive Graphics Functions)：指令允許互動式地用其他設備（如滑鼠）在一個已經存在的繪圖上，加上圖形資訊或者萃取圖形中的資訊。

### (二)、高階繪圖

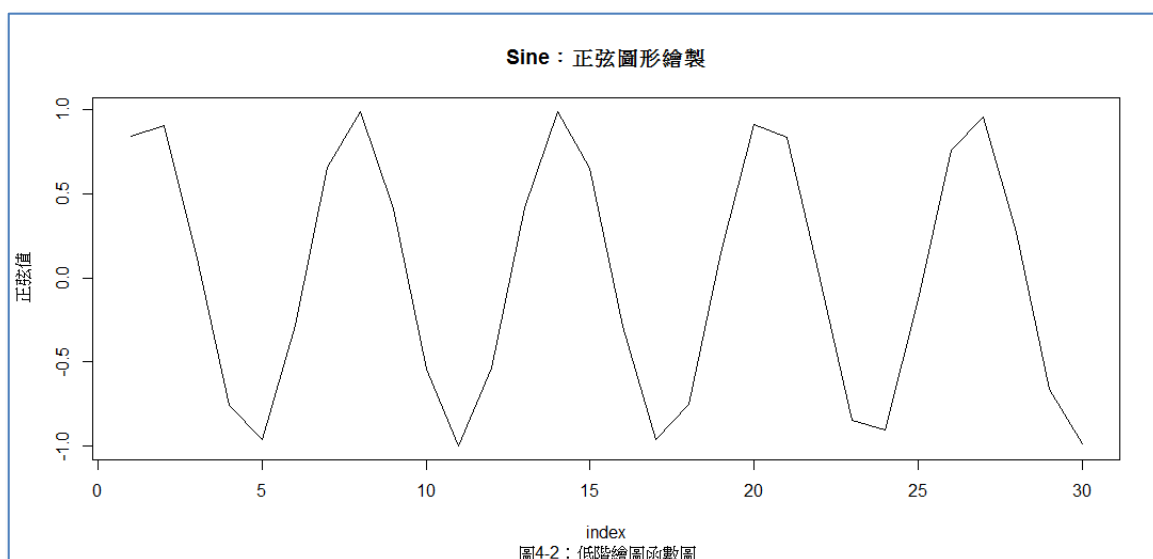
```
> x <- sin(1:30)
```

```
> plot(x, type="l", main="Sine: 正弦圖形繪製", xlab="index", ylab="
正弦值")
```



### (三)、低階繪圖

```
> x <- sin(1:30)
> plot(x, type="l", xlab="index", ylab="正弦值")
> title(main="Sine: 正弦圖形繪製", sub="圖 4-2：低階繪圖函數圖")
```

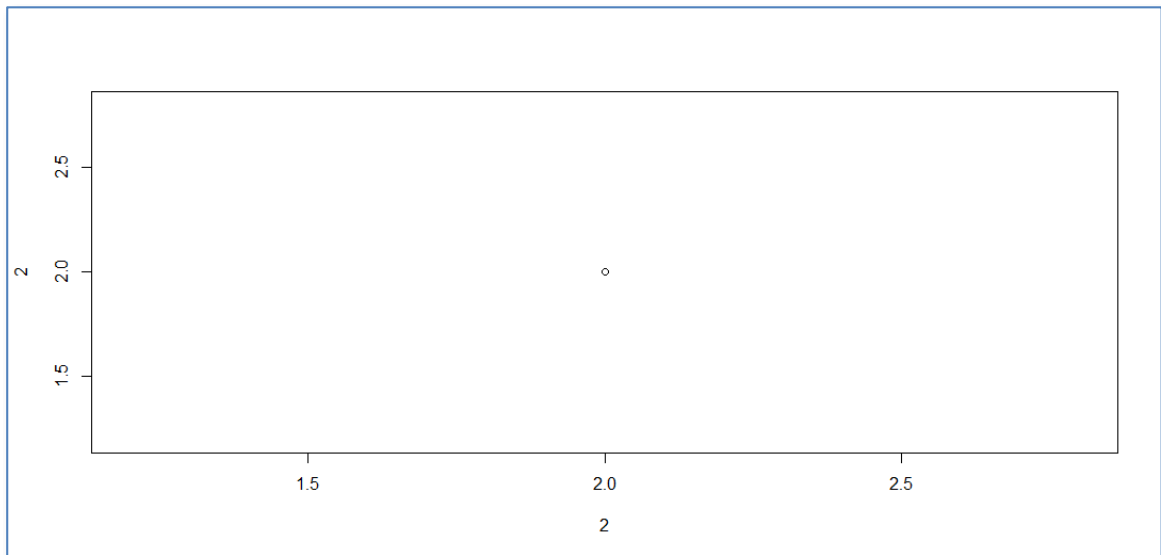


#### (四)、互動式繪圖

**locator()** 函數：可以用來獲取滑鼠點擊在圖上的座標。

```
> plot(2, 2)
```

```
> pts <- locator(n=3) #user 可在圖形中以滑鼠左鍵點選 3 個座標點
```



```
> pts #點選完成後，pts 的值
```

```
$x
```

```
[1] 1.951190 2.083981 2.024764
```

```
$y
```

```
[1] 2.081201 2.147592 1.825852
```

**locator()** 函數：可以用來獲取滑鼠點擊在圖上的座標，比如：**text(locator(1), "標示文字")**，會把"標示文字"放到滑鼠點擊位置。

```
> plot(2, 2)
```

```
> text(locator(1), "第 1 點")
```

```
> text(locator(1), "第 2 點")
```

```
> text(locator(1), "第 3 點")
```

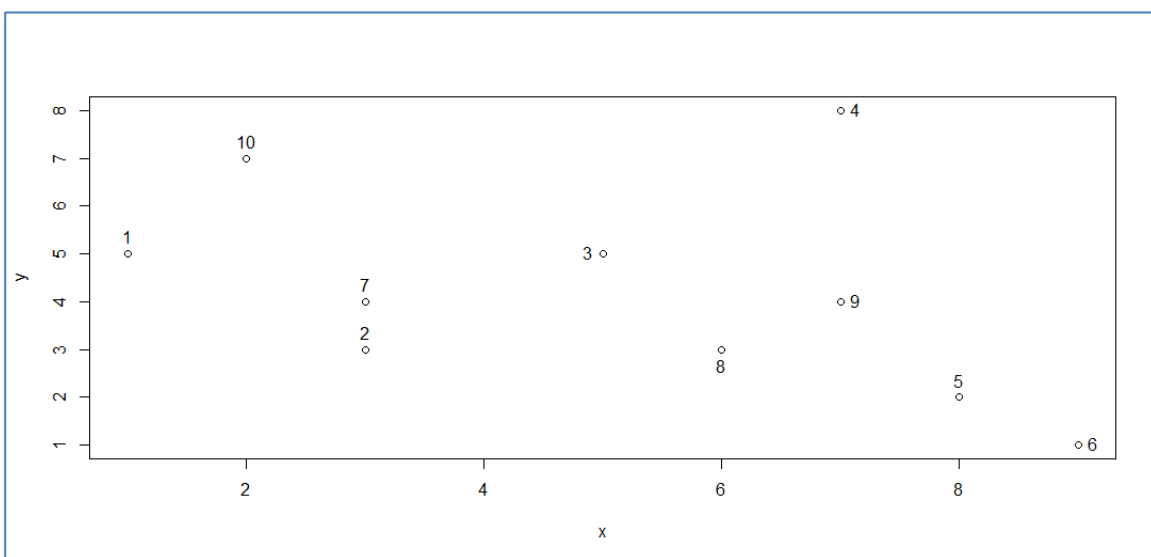
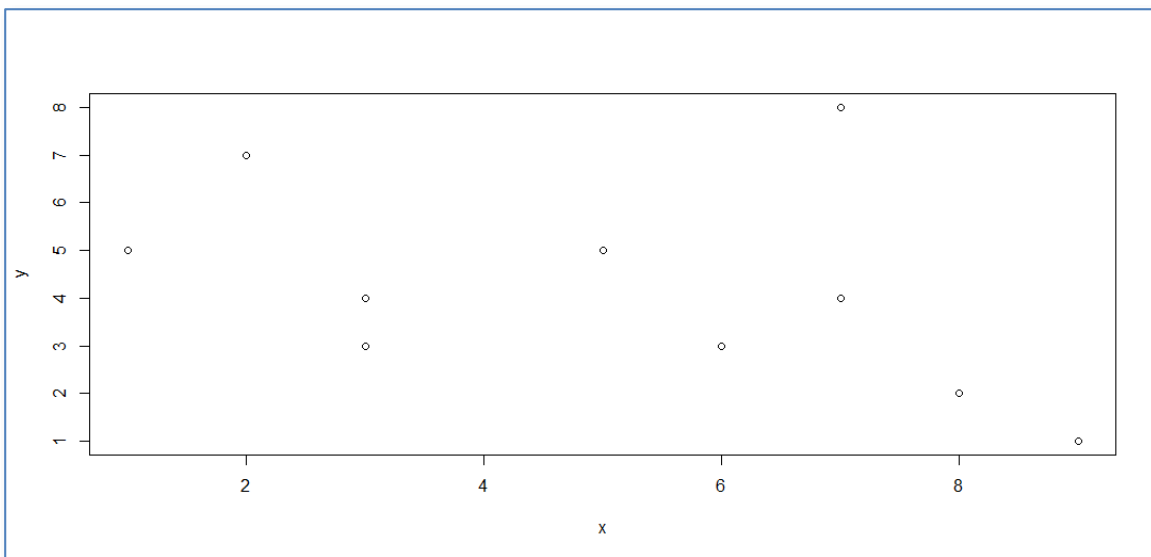
**identify()** 函數：可以識別點擊處的點並標注標籤，格式為 **identify(x, y, labels)**，其中 **(x,y)** 給出可點擊的點的座標，**labels** 是每個點對應的標籤，點擊那個點就在那個點旁邊標對應的標籤。

```
> x <- c(1, 3, 5, 7, 8, 9, 3, 6, 7, 2)
```

```
> y <- c(5, 3, 5, 8, 2, 1, 4, 3, 4, 7)
```

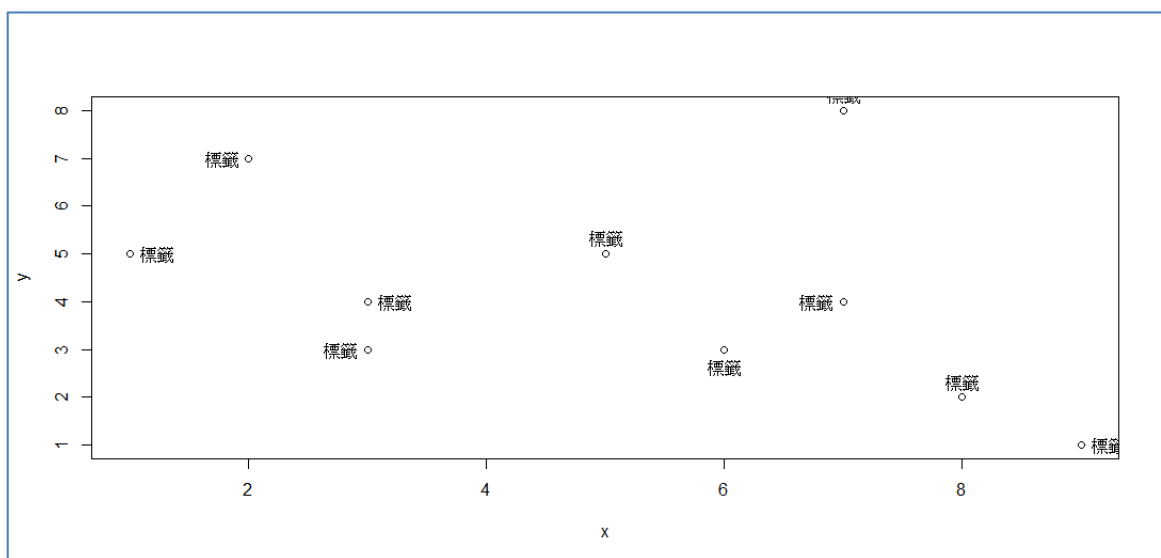
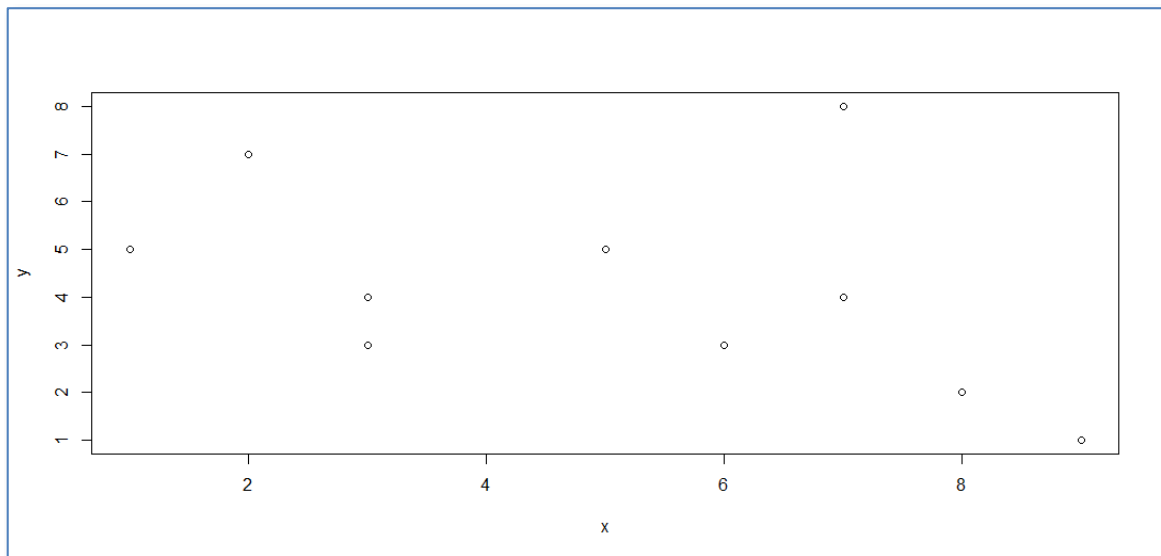
```
> plot(x,y)
```

```
> sel <- identify(x,y)
```



```
> x <- c(1, 3, 5, 7, 8, 9, 3, 6, 7, 2)
> y <- c(5, 3, 5, 8, 2, 1, 4, 3, 4, 7)
> plot(x,y)
> sel <- identify(x,y,"標籤")
```





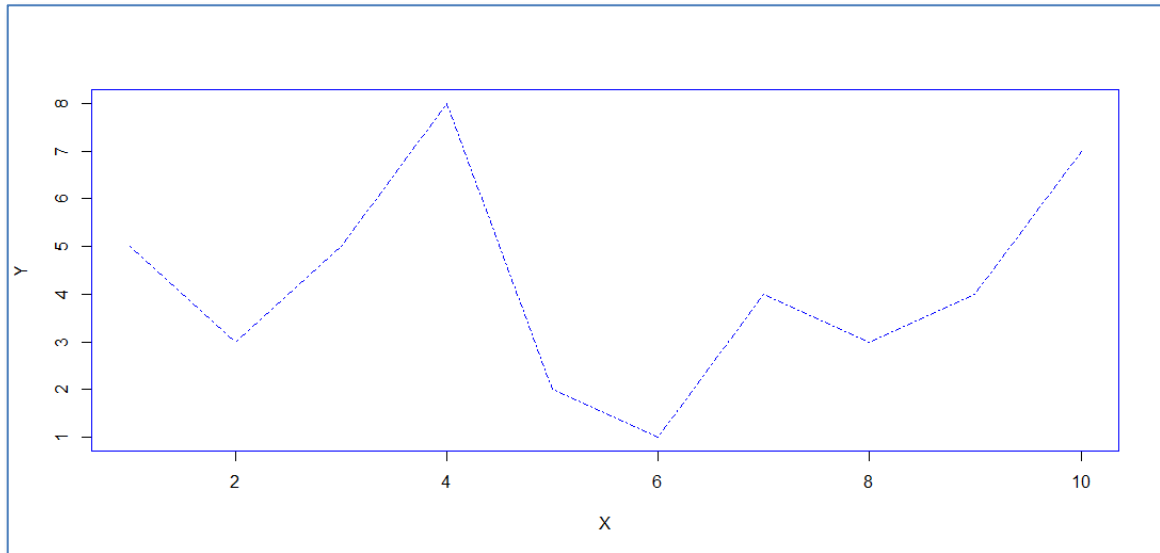
```
> x.sel <- x[sel]
> y.sel <- y[sel]
> x.sel
[1] 1 3 5 7 8 9 3 6 7 2
> y.sel
[1] 5 3 5 8 2 1 4 3 4 7
```

## (五)、圖形參數

用圖形參數可以選擇點的形狀、顏色、線型、粗細、坐標軸做法、邊空、一頁多圖等。有些參數直接用在繪圖函數內，如 `plot` 函數可以用 `pch`、`col`、`cex`、`lty`、`lwd` 等參數。有些圖形參數必須使用 `par()` 函數指定。

**par()** 函數指定圖形參數並返回原來的參數值，所以在修改參數值作圖後通常應該恢復原始參數值，做法如：

```
> x <- c(5, 3, 5, 8, 2, 1, 4, 3, 4, 7)
> par(col=4, lty=4) #col=4: 設置繪圖顏色; lty=4: 短線點
> plot(x, type="l", xlab="x", ylab="Y")
```



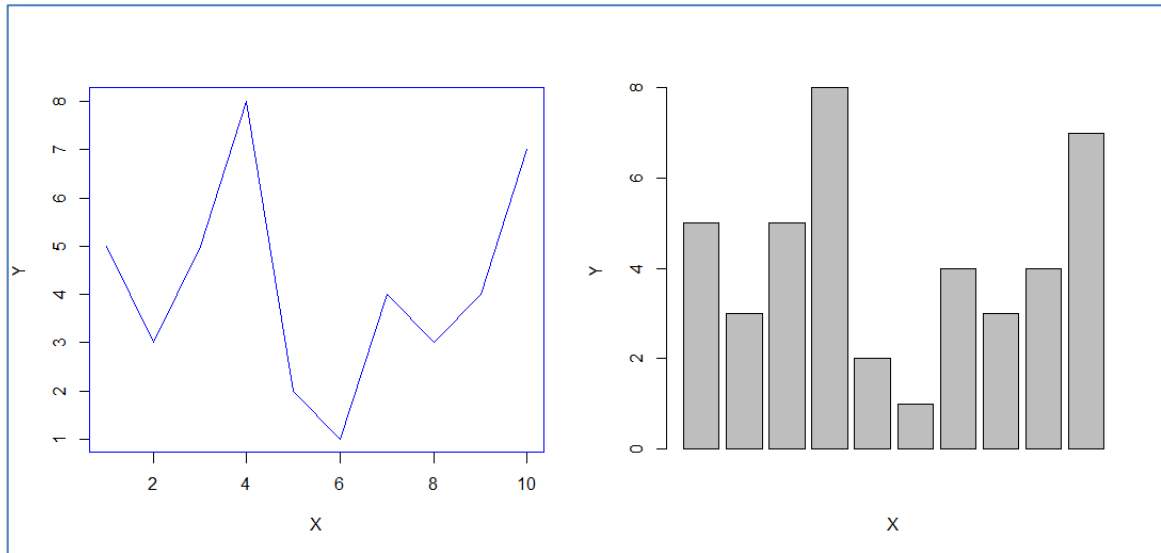
- **bg**：設置繪圖區背景色。預設值為 `bg="transparent"`。
- **col**：設置繪圖顏色。預設值為 `col="black"`。
  - **col.axis**：設置座標標記顏色。預設值為 `col.axis="black"`。
  - **col.lab**：設置坐標軸標題顏色。預設值為 `col.lab="black"`。
  - **col.main**：設置圖標題顏色。預設值為 `col.main="black"`。
  - **col.sub**：設置圖副標題顏色。預設值為 `col.sub="black"`。
- **fg**：設置前景色，主要用於坐標軸、邊框、圖形等，對座標標記與坐標軸標題等週邊無影響。預設值為 `fg="black"`。
- **font**：設置文本字體。`=1`（預設值）：普通字體、`=2`：粗體、`=3`：斜體、`=4`：粗斜體、...。
  - **font.axis**：設置座標標記字體。
  - **font.lab**：設置坐標軸標題字體。
  - **font.main**：設置圖標題字體。
  - **font.sub**：設置圖副標題字體。
- **lty**：設置線的類型。`=0`：空白、`=1`（預設值）：實線、`=2`：短線虛線、`=3`：點虛線、`=4`：短線點虛線、`=5`：長線虛線、`=6`：長短線虛線。
- **pch**：設置點的類型。預設值為 `pch=1`。

~ <http://newtomaso.blogspot.tw/2015/02/par-in-r-rpar.html>

```

> par(mfrow=c(1, 2)) #設置小圖數量與位置
> par(mar=c(5, 4, 4, 2)) #設置圖形空白邊界行數
> par(col=4, lty=1)
> plot(x, type="l", xlab="x", ylab="Y")
> barplot(x, xlab="x", ylab="Y")

```



- **mfcol,mfrow**：設置小圖數量與位置，取值為數值型向量  $c(nr, nc)$ 形式，表示把圖分為  $nr$  行  $nc$  列個小圖，圖形順序按列排（mfcol）或按行排（mfrow）。
- **mar**：設置圖形空白邊界行數， $mar=c(bottom, left, top, right)$ 。預設值為  $mar=c(5.1, 4.1, 4.1, 2.1)$ 。

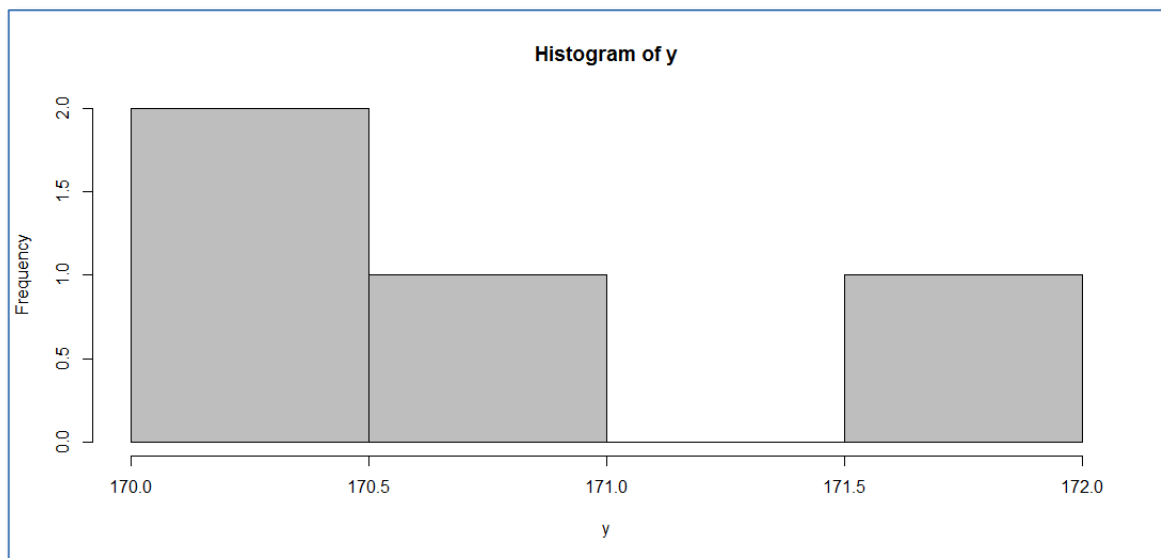
## (六)、基本統計

統計中之敘述統計（Descriptive Statistics）的主要目的乃是透過數值或圖形，呈現資料之特性及瞭解樣本的統計特徵。

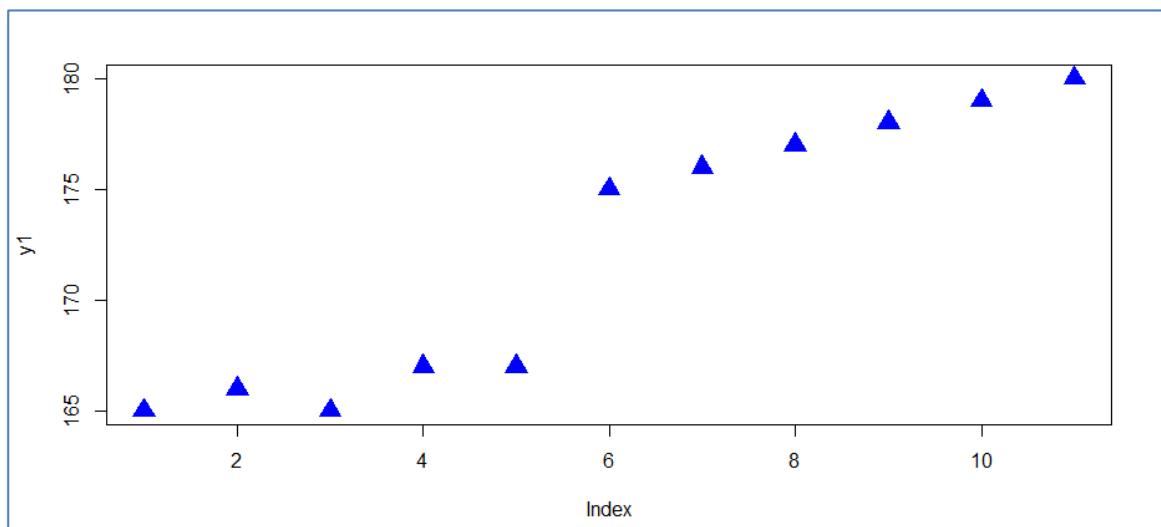
```

> y <- c(170, 170, 171, 172)
> hist(y, col='grey')

```

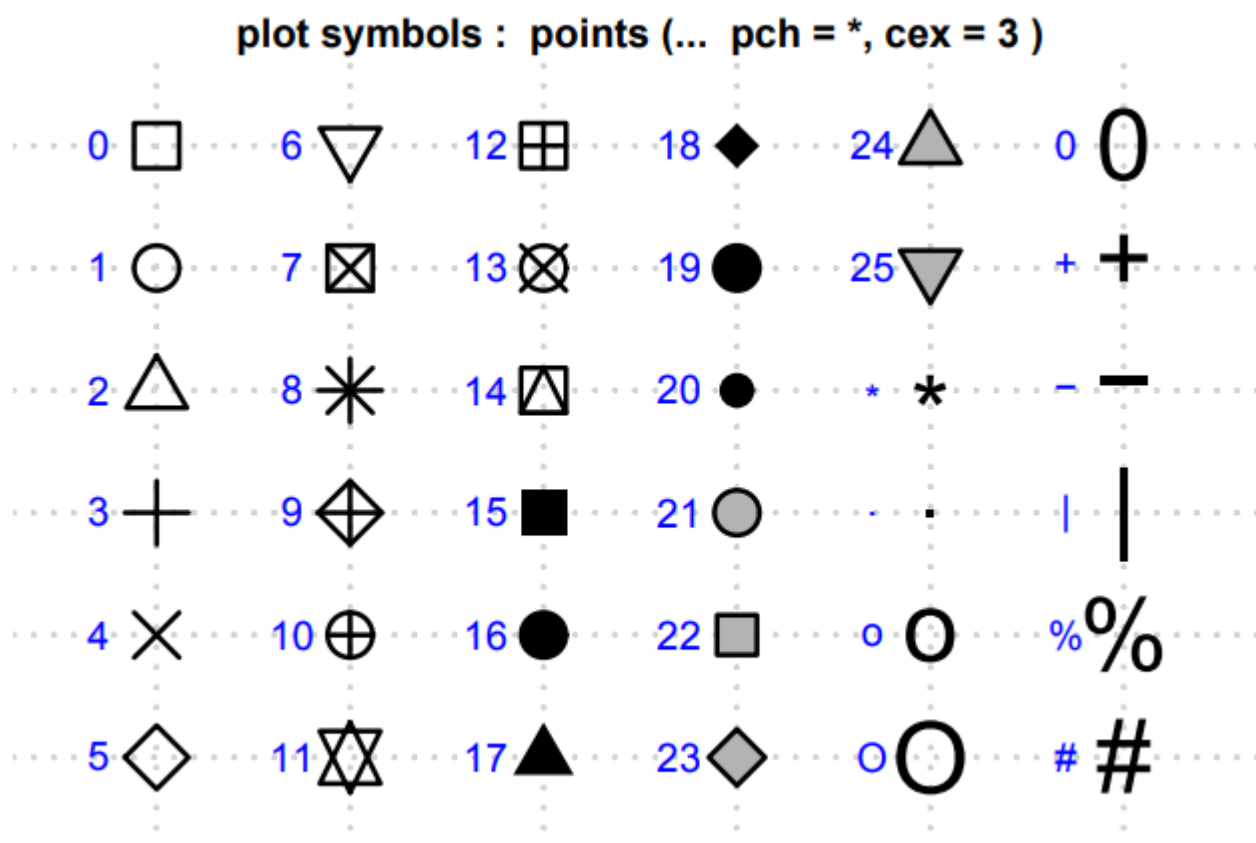


```
> y1 <- c(165,166,165,167,167,175,176,177,178,179,180)
> plot(y1, pch = 17, col = "blue", cex = 2)
```



在 R 語言中，點的樣式由 pch 的取值決定。

- 當 pch 取 0~14 時，其點為空心點，可以用 col (顏色) 參數設置其邊框的顏色。
- 當 pch 取 15~20 時，其點是實心點，可以用 col 參數設置其填充的顏色。
- 當 pch 取 21~25 時，其點也是實心點，既可以用 col 參數設置邊框的顏色，也可以用 bg 參數設置其內部的填充顏色。



圖：pch 引數設定高階繪圖函式在畫點時之顯示方式

```
> #總合
> sum(y1)
[1] 1895

> #平均數(總合除個數)
> mean(y1)
[1] 172.2727
> round(mean(y1))
[1] 172
> round(mean(y1), 1)
[1] 172.3

> max(y1) #最大值
[1] 180

> min(y1) #最小值
[1] 165
```

```
> #全距(最大值減最小值)
```

```
> max(y1)-min(y1)
```

```
[1] 15
```

```
> range(y1)
```

```
[1] 165 180
```

```
> #中位數：將資料由小到大，位置居中者，就是中位數
```

```
> median(y1)
```

```
[1] 175
```

```
> #眾數：一組資料中，出現最多次數的值基本，package 沒有這個功能
```

```
> as.numeric(names(table(y1)))[which.max(table(y1))]
```

```
[1] 165
```

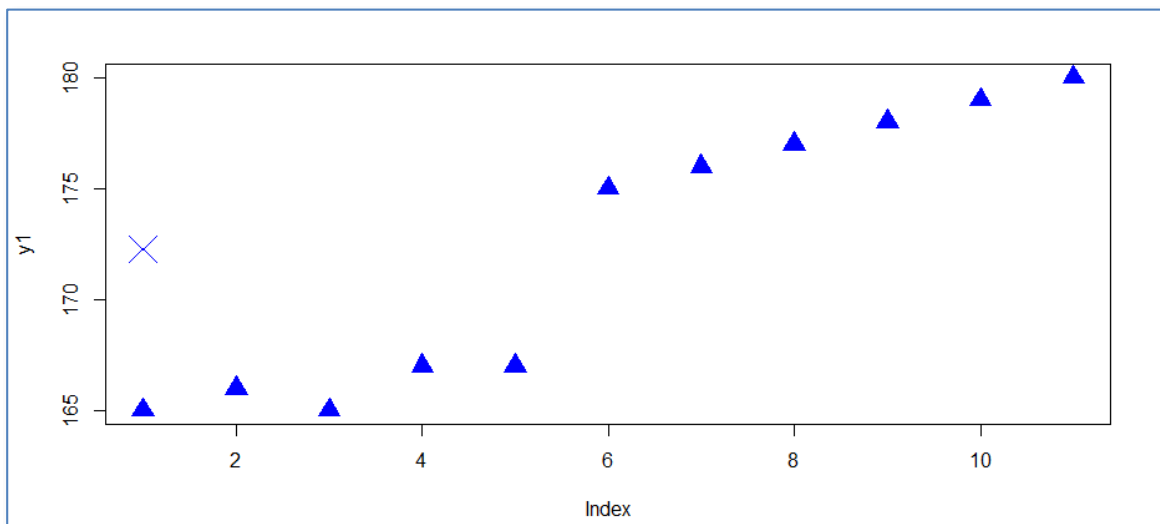
```
> which.max(table(y1))
```

```
165
```

```
1
```

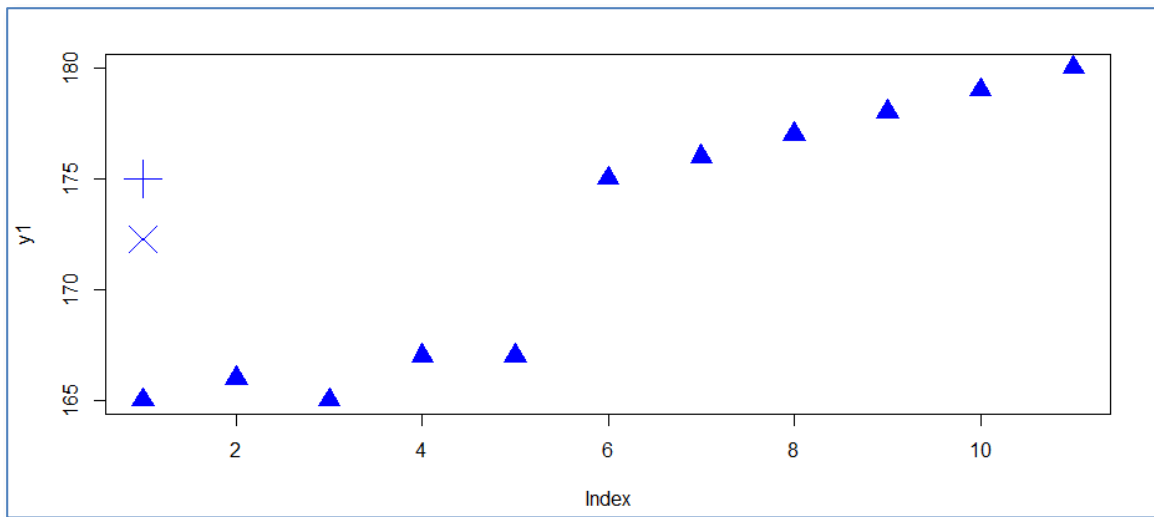
```
> #畫平均數的點
```

```
> points(mean(y1), pch = 4, col = "blue", cex = 3)
```



```
> #畫中位數的點
```

```
> points(median(y1), pch = 3, col = "blue", cex = 3)
```



> #四分位：把資料切分為四等分，中間的三條線就是四分位

> Q1 <- quantile(y1, 1/4) #第 1 個四分位數

> Q1

25%

166.5

> Q2 <- quantile(y1, 2/4) #第 2 個四分位數

> Q2

50%

175

> Q3 <- quantile(y1, 3/4) #第 3 個四分位數

> Q3

75%

177.5

> #IQR = Q3-Q1

> IQR(y1) #四分位數間距

[1] 11

> b <- Q3 - Q1 == IQR(y1) #四分位數間距

> b

75%

TRUE

> #標準差

> sd(y1)

```
[1] 6.182086
```

```
> #變異數
```

```
> var(y1)
```

```
[1] 38.21818
```

```
> sd(y1) ^ 2
```

```
[1] 38.21818
```

```
> #變異係數
```

```
> cv <- 100 * sd(y1) / mean(y1)
```

```
> cv
```

```
[1] 3.588546
```



## 八、探索資料分析

### 資料來源：《輕鬆學習 R 語言》

將資料輸入 R 語言之後，針對陌生的資料進行的會是探索資料分析（Exploratory Data Analysis，EDA），透過這個步驟可以知道資料的外觀、維度以及變數的分佈等資訊。

#### (一)、IRIS 資料集說明及簡單整理：

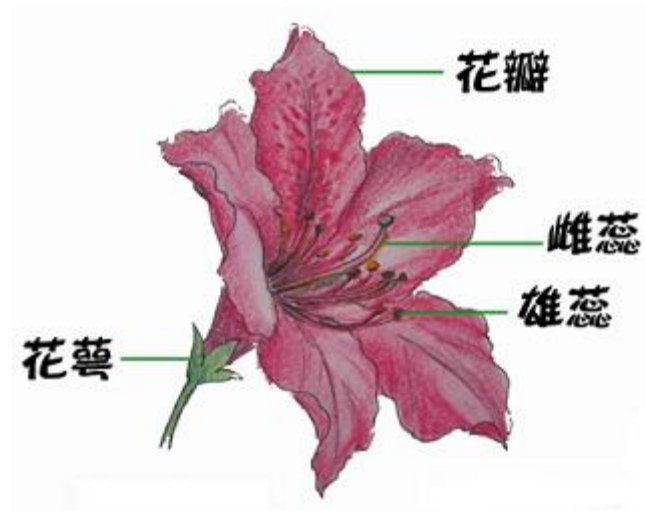
鳶尾花（iris）資料，最早由英國統計學家費雪（R. A. Fisher, 1890~1962）用於多變量分析（multivariate analysis）中的判別分析（discriminant analysis），故常稱為費雪鳶尾花資料。此資料是由美國植物學家安德生（E. S. Anderson, 1897~1969）所收集，故也稱為安德生鳶尾花資料。此資料記錄了鳶尾花三個亞種及其特徵，三亞種分別為山鳶尾（setosa）、變色鳶尾（versicolor）及維吉尼亞鳶尾（virginica），花的特徵則包含花萼（sepal）與花瓣（petal）的長度與寬度。

表 A1：鳶尾花資料

變數名稱	花萼長度	花萼寬度	花瓣長度	花瓣寬度	品種
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
:	:	:	:	:	:
150	5.9	3.0	5.1	1.8	virginica

R 語言內建的鳶尾花資料集（iris）是一組非常著名的生物資訊資料集，取自美國加州大學歐文分校的機械學習資料庫 <http://archive.ics.uci.edu/ml/datasets/Iris>，資料筆數總共有 150 筆，每筆資料有 5 個欄位，各欄位的意義如下：

- Sepal.Length：花萼長度，計算單位是公分。
- Sepal.Width：花萼寬度，計算單位是公分。
- Petal.Length：花瓣長度，計算單位為公分。
- Petal.Width：花瓣寬度，計算單位為公分。
- Species：品種，可分為 Setosa、Versicolor 與 Virginica 三種。



山鳶尾 (Setosa)	變色鳶尾 (Versicolor)	維吉尼亞鳶尾 (Virginica)

## (二)、內建函數

常見的資料讀入之後會以資料框 (Data Frame) 格式儲存，針對資料框有幾個內建函數：

- 資料的維度

使用 `nrow()`、`ncol()` 與 `dim()` 這三個函數可以得知這個資料的維度資訊：

```
> nrow(iris) # 顯示內建資料 iris 有幾個觀測值
[1] 150
```

```
> ncol(iris) # 顯示內建資料 iris 有幾個變數
[1] 5
```

```
> dim(iris) # 顯示內建資料 iris 有幾個觀測值與變數
[1] 150 5
```

其中 `nrow()` 函數命名是 number of rows 的縮寫，`ncol()` 函數命名是 number of columns

的縮寫，dim() 函數命名是 dimensions 的縮寫。

- 資料的外觀

使用 head() 與 tail() 這兩個函數可以得知部分資料的外觀，預設是印出前六或後六個觀測值（含變數名稱）；使用 names() 函數可以得知變數名稱。

```
> head(iris) # 顯示 iris 的前 6 個觀測值（含變數名稱）
```

	Sepal.Length	Sepal.width	Petal.Length	Petal.width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

```
> tail(iris) # 顯示 iris 的後 6 個觀測值（含變數名稱）
```

	Sepal.Length	Sepal.width	Petal.Length	Petal.width	Species
145	6.7	3.3	5.7	2.5	virginica
146	6.7	3.0	5.2	2.3	virginica
147	6.3	2.5	5.0	1.9	virginica
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica

```
> names(iris) # 顯示 iris 的 5 個變數名稱
```

```
[1] "Sepal.Length" "Sepal.width" "Petal.Length" "Petal.width" "Species"
```

- 變數的分佈情況

使用 summary() 函數可以得知每一個變數的描述性統計量：

```
> summary(iris) # 顯示 iris 的變數描述性統計
```

Sepal.Length	Sepal.width	Petal.Length	Petal.width	Species
Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100	se
tosa :50				
1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.300	v
ersicolor:50				
Median :5.800	Median :3.000	Median :4.350	Median :1.300	v
irginica :50				

Mean :5.843 Mean :3.057 Mean :3.758 Mean :1.199

3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800

Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500

- 包含很多資訊的 **str()** 函數

使用 **str()** 函數可以輸出關於資料很多的資訊：像是資料結構的種類、觀測值數、變數個數、變數名稱與前幾筆觀測值等，**str()** 函數命名是 **structure** 的縮寫。

```
> str(iris)
```

```
'data.frame': 150 obs. of 5 variables:
 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1
 1 1 1 1 1 1 1 1 ...
```

### (三)、繪圖

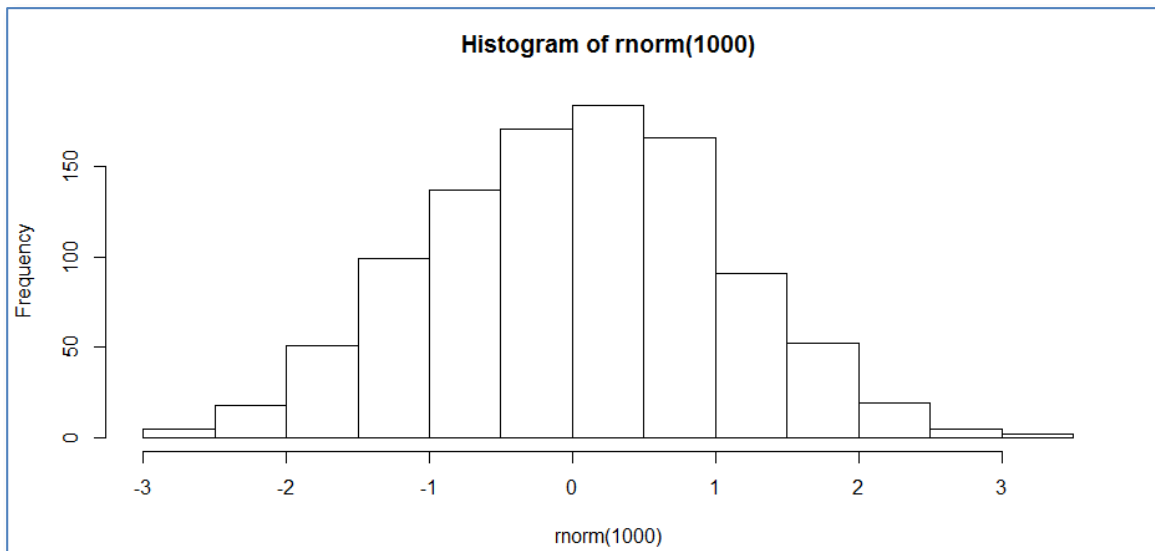
現在要使用 **Base Plotting System** 這個 R 語言內建的繪圖系統來進行探索資料分析，它提供一系列的函數，每一個函數都負責繪製一種圖形，在逐一檢視之前先用下表對這些函數有一個概觀：

函數	圖形	範例
hist()	直方圖	hist(rnorm(1000))
boxplot()	盒鬚圖	boxplot(Sepal.Length ~ Species, data = iris)
plot(..., type = "l")	線圖	plot(AirPassengers, type = "l")
plot()	散佈圖	plot(cars)
barplot()	長條圖	barplot(table(mtcars\$cyl))
curve()	曲線圖	curve(sin, from = 0, to = pi * 2)

- 探索數值分佈

使用 **hist()** 函數繪製直方圖來探索數值的分佈，**hist()** 函數命名是 **histogram** 的縮寫。

```
> hist(rnorm(1000))
```

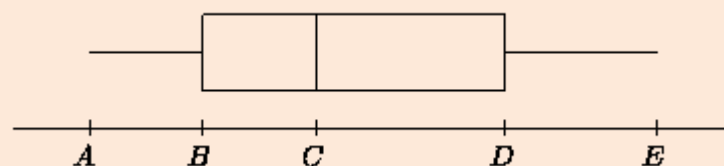


其中 `rnorm()` 函數的作用是為了產生一組符合標準常態分佈（平均值為 0，標準差為 1）的數字來繪圖，其中的 1000 是指定要產生一千個符合指定分佈的數字，這個參數要稍微大一些才能夠看出明顯的鐘型外觀。

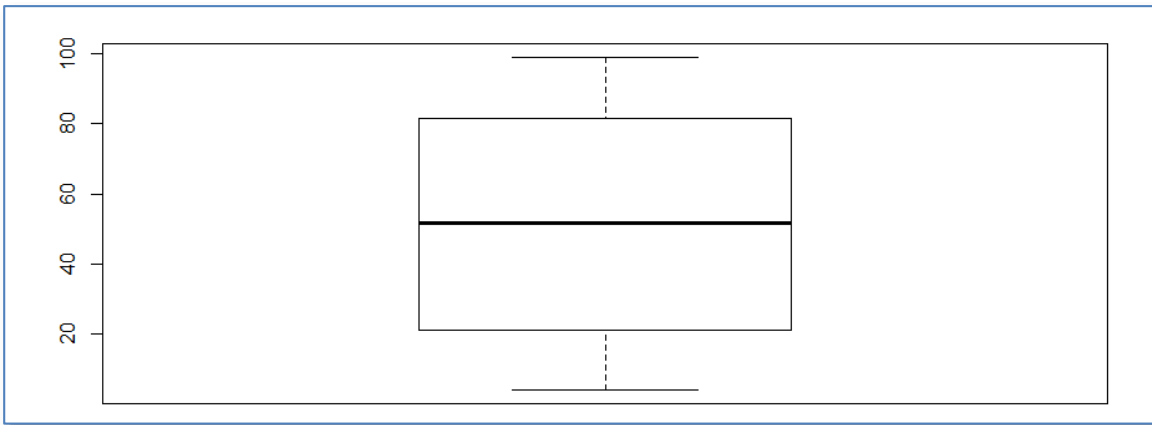
#### ● 探索不同類別與數值分佈的關係

使用 `boxplot()` 函數繪製盒鬚圖（box-and-whisker plot，又稱長鬚圖，簡稱 boxplot）。盒鬚圖可用來瞭解資料的偏斜性（skewness）及離群值（outliers）。

A 稱為資料的最小值，E 稱為最大值，B 與 D 則分別為資料的下四分位數及上四分位數，因此圖中盒子包含資料的中間 50% 部分。又 C 為資料的中位數。上圖包含一個盒子，及二凸出來的鬚，這是此圖命名的由來。

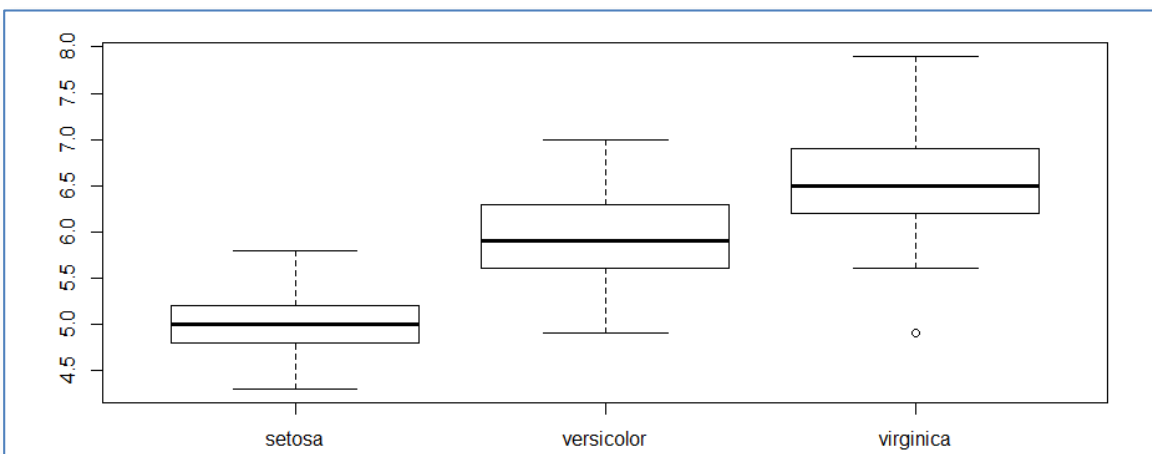


```
> score <- sample(c(0:100), size=60, replace=T)
> score
[1]  7 73 96 19 44 20 93 56 37 16 98 38 49 22 33 65 76 20 28 20 57 16 95
50  4 89 19 18 57 59 97 25 80 13 24 87
[37] 62 28 65 95 46 24 53 75 15 87 65 99 16 20 84 53 11 83 85 31 91 73 89
33
> boxplot(score)
```



盒鬚圖可以用來探索不同類別與數值分佈的關係，以下面這段程式為例，我們探索內建資料 iris 中不同鳶尾花品種（Species 變數）的萼片長度（Sepal.Length 變數）分佈差異：

```
> boxplot(Sepal.Length ~ Species, data = iris)
```



- 探索數值與日期（時間）的關係

使用 `plot(..., type = "l")` 函數繪製線圖來探索數值與日期（時間）的關係：

```
> x <- seq(from = as.Date("2017-01-01"), to = as.Date("2017-01-31"), by = 1)
```

```
> x
```

```
[1] "2017-01-01" "2017-01-02" "2017-01-03" "2017-01-04" "2017-01-05"
"2017-01-06" "2017-01-07" "2017-01-08"
```

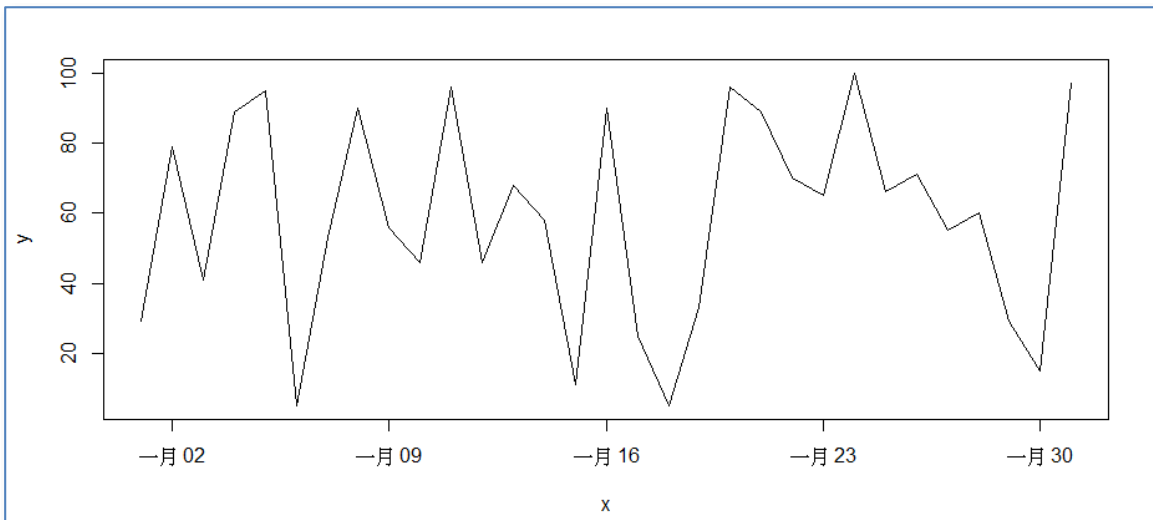
```
[9] "2017-01-09" "2017-01-10" "2017-01-11" "2017-01-12" "2017-01-13"
"2017-01-14" "2017-01-15" "2017-01-16"
```

```
[17] "2017-01-17" "2017-01-18" "2017-01-19" "2017-01-20" "2017-01-21"
"2017-01-22" "2017-01-23" "2017-01-24"
```

```
[25] "2017-01-25" "2017-01-26" "2017-01-27" "2017-01-28" "2017-01-29"
"2017-01-30" "2017-01-31"
```

```
> set.seed(123) # 為了能夠得到一樣的向量 y
```

```
> y <- sample(1:100, size = 31, replace = TRUE)
> y
[1] 29 79 41 89 95 5 53 90 56 46 96 46 68 58 11 90 25
5 33 96 89 70 65 100 66 71 55
[28] 60 29 15 97
> plot(x, y, type = "l")
```

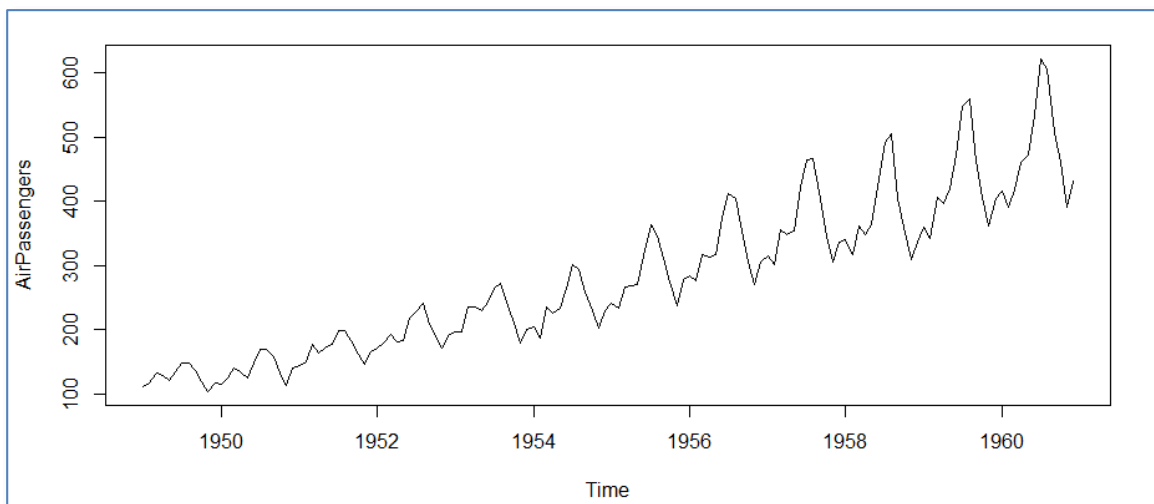


先建立一個向量 `x` 是 2017 年 1 月的 31 天，然後隨機從 1~100 之中選 31 個數值（可以重複）指派給向量 `y`，然後參數指定 `type = "l"`，亦即 line 的縮寫，就能夠順利繪製出線圖。

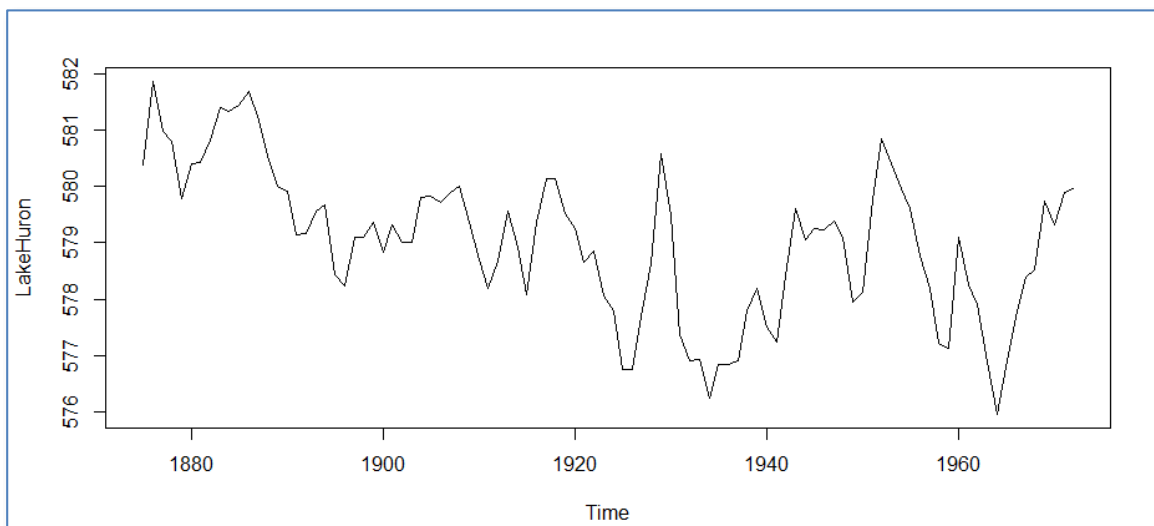
`set.seed()` 函數是設定隨機種子，由於使用了 `sample()` 這個函數做隨機抽樣，為了夠得到一樣的向量 `y`，於是設定了一個整數 123 作為依據，只要在執行 `sample()` 函數之前執行 `set.seed(123)`，每次都會獲得相同的向量 `y`。

在 R 語言的內建資料中，像是 `AirPassengers` 與 `LakeHuron` 等，它們的資料結構是 `ts` 亦即 `time series` 時間序列的縮寫，它們可以直接作為 `plot()` 函數的輸入，就可以完成線圖的繪製（不需另外指派參數 `type = "l"`）。

```
> plot(x, y, type = "l")
> class(AirPassengers)
[1] "ts"
> class(LakeHuron)
[1] "ts"
> plot(AirPassengers)
```



> plot(LakeHuron)

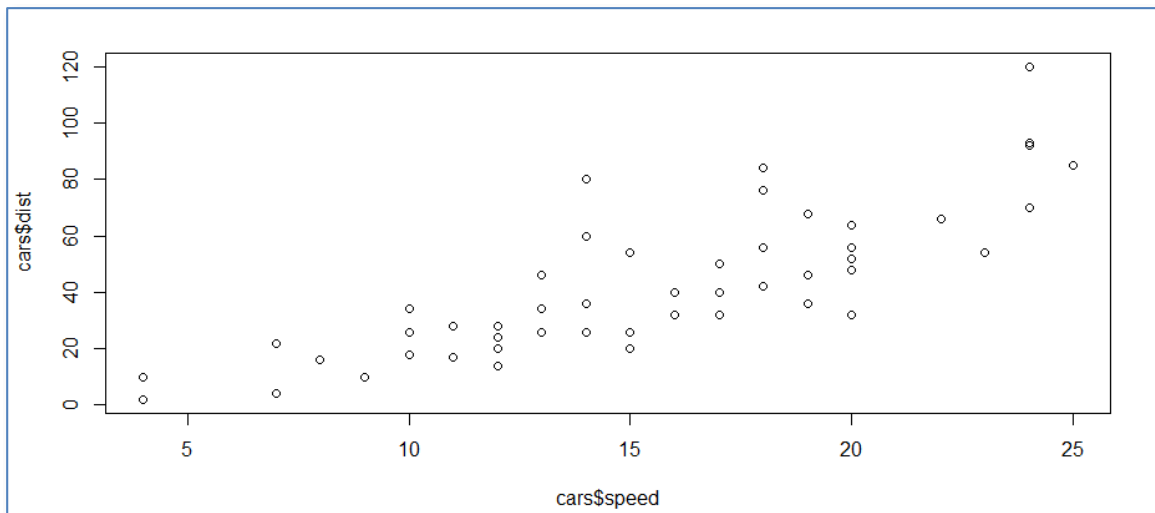


- 探索兩個數值的關係

使用 plot() 函數繪製散佈圖來探索兩個數值的關係：

> plot(cars\$speed, cars\$dist)

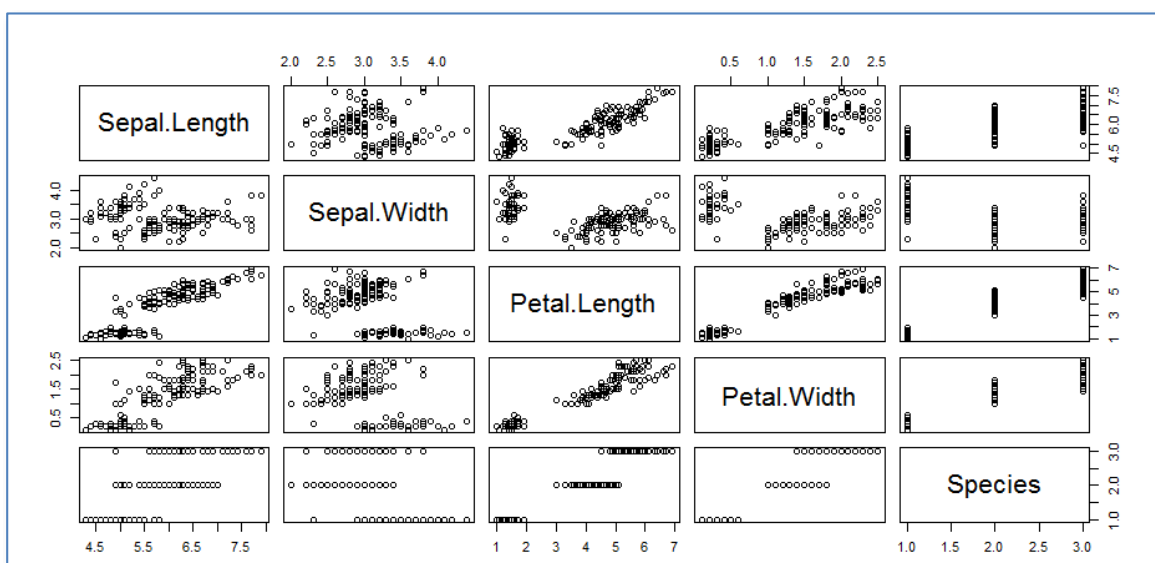




從圖中可以看出資料的軌跡，隨著車速增加，所需要的煞車距離也隨之增加。

plot() 函數除了可以繪製單一散佈圖，還支援散佈圖矩陣 (scatter matrix) 的繪製：

```
> plot(iris)
```



## ● 探索類別

使用 `barplot()` 函數繪製長條圖來探索類別的分佈，假如有一個向量 `ice_cream_flavor` 紀錄了 100 個人最喜歡的霜淇淋口味：

```
> ice_cream_flavor <- rep(NA, times = 100)
```

```
> ice_cream_flavor
```

```
[1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA N  
A NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```



30

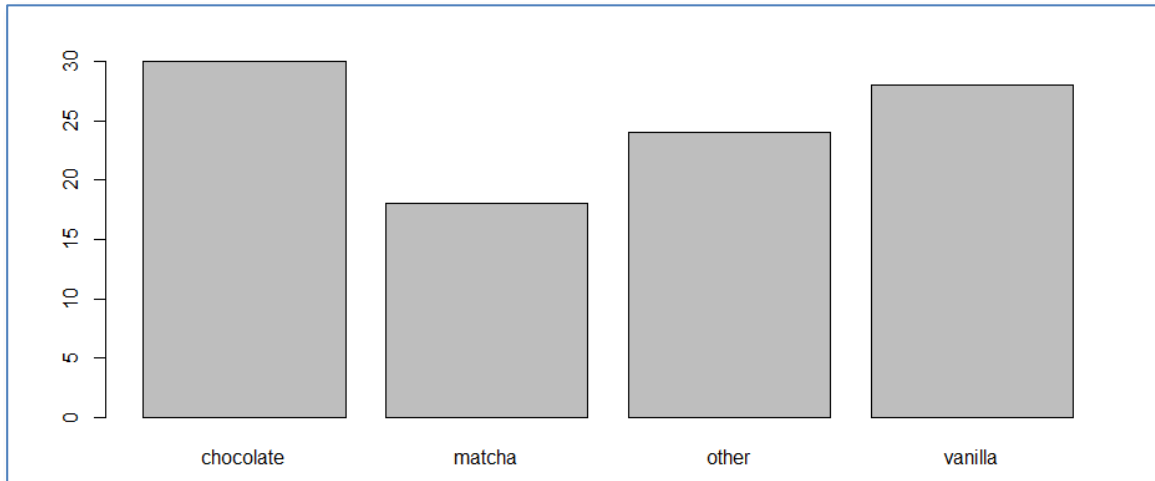
18

24

28

可以一目瞭然，有 30 個人最喜歡的口味是巧克力、18 個人最喜歡抹茶、24 個人喜歡其他的口味而有 28 個人最喜歡香草。這時就可以使用 `barplot()` 函數將這個清晰的結果繪畫成清楚的長條圖：

```
> barplot(table(ice_cream_flavor))
```



## 九、R 語言資料預處理：

### (一)、資料的預處理

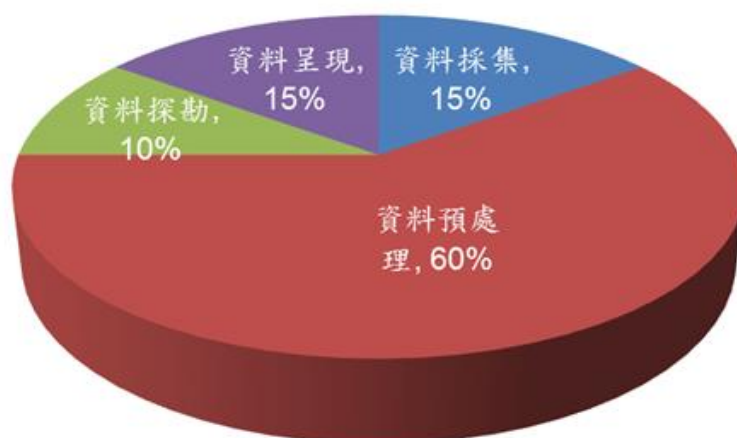
資料分析最消耗的在於：資料的預處理。

資料探勘是從大量的、不完整的、有噪音的，模糊的資料中，提取隱含在其中的有潛在的有用資訊的過程。

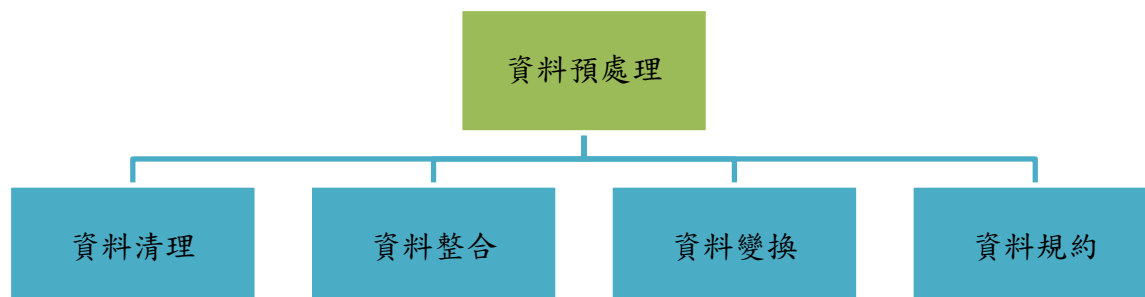
數資料探勘過程：



其工作量占比：



對不同的源資料進行預處理分為四大功能：



實際的資料預處理過程中，這四種功能不一定都用得到，而且，他們的使用也沒有先後順序，某種預處理可能先後要多次進行。

在資料探勘整體過程中，海量的源資料中存在著大量複雜的、重複的、不完整的資料，嚴重影響到資料探勘演算法的執行效率，甚至可能導致探勘結果的偏差，為此，在資料探勘演算法執行之前，必須對收集到的源資料進行預處理，以改進資料的質量，提高資料探勘過程的效率、精度、性能。

### 1. 資料清理

資料清理要去除源資料的噪音資料和無關資料，處理遺漏資料和清洗髒資料、空缺值、識別刪除孤立點等。

- (1). 噪音：噪音是一個測量變量中的隨即錯誤和偏差，包括錯誤的值或偏離期望的孤立點值，對於噪音資料有如下幾種處理方法：分箱法，聚類法識別孤立點，回歸法。
- (2). 空缺值的處理：目前最常用的方法是使用可能的值填充空缺值，如用一個全域常量替換空缺值，使用屬性的平均值填充空缺值或將所有元組按照某些屬性分類，然後用同一類中屬性的平均值填充空缺值。
- (3). 清洗髒資料：異質源資料庫中的資料不一定是正確的，常常不可避免地存在著不完整、不一致、不精確和重複的資料，這些資料統稱為「髒資料」。髒資料能使探勘過程陷入混亂，導致不可靠的輸出。清洗髒資料可採用的方式：
  - 手工實現方式
  - 用專門編寫的應用程式
  - 採用機率統計學遠離查找數值異常的記錄
  - 對重複記錄的檢測和刪除

### 2. 資料整合

- (1). 實體識別問題：在資料整合時候，來自多個資料源的真實世界的實體有時並不一定是匹配的，例如：資料分析者如何才能確信一個資料庫中的 `student_id` 和另一個資料庫中的 `stu_id` 值是同一個實體，通常，可以根據資料庫或者資料倉儲的元資料來區分模式整合中的錯誤。
- (2). 冗餘問題：資料整合往往導致資料冗餘，如同一屬性多次出現，統一屬性命名不一致等，對於屬性間冗餘可以用相關分析檢測到，然後刪除。
- (3). 資料值衝突檢測與處理：對於真實世界的同一實體，來自不同資料源的屬性值可能不同，這可能是因為表示、比例、編碼、資料類型、單位不統一、欄位長度不同。

### 3. 資料變換

資料變換主要是找到資料的特徵表示，用維變換或轉換方法減少有效變量的數目或找到資料的不變式，包括規格化、規約、卻換、旋轉、投影等操作。

規格化是指將元組集按照規格化條件進行合併，也就是屬性值量綱的歸一化處理。規格化定義了屬性的多個取值到給定虛擬值的對應關係，對於不同的數值屬性特點，一般可以分為取值連續和取值分散的數值屬性規格化問題。

#### 4. 資料規約

規約：是指將元組按語義層次結構合併，語義層次結構定義了元組屬性值之間的語義關係，規約化和規約能大量減少元組個數，提高計算效率，同時，規格化和規約過程提高了知識發現的起點，使得一個演算法能夠發現多層次的知識，適應不同應用的需要。

資料規約是將資料庫中的海量資料進行規約，規約之後的資料仍接近於保持原資料的完整性，但資料量相對小的多，這樣進行挖掘的性能和效率會得到很大的提高。

資料規約的策略主要有資料立方體聚集，維規約，資料壓縮，數值壓縮，離散化和概念分層。

- (1). 維規約：通過刪除不相關的屬性（或緯）減少資料量，不僅僅壓縮了資料集，還減少了出現在發現模式上的屬性數目，通常採用屬性子集選擇方法找出最小屬性集，使得資料類的機率分佈儘可能的接近使用所有屬性的原分佈。
- (2). 資料壓縮，資料壓縮分為無損壓縮和有損壓縮，比較流行和有效的有損資料壓縮方法是小波變換和主要成分分析，小波變換對於稀疏或傾斜資料以及具有有序屬性的資料有很好的壓縮效果。
- (3). 數值規約：數值規約通過選擇替代的、較小的資料表示形式來減少資料量。數值規約技術可以是有參的，也可以是無參的。有參方法是使用一個模型來評估資料，只需存放參數，而不需要存放實際資料。有參的數值規約技術有以下兩種，回歸：線性回歸和多元回歸；對數線性模型：近似離散屬性集中的多維機率分佈。無參的數值規約技術有 3 種：直方圖、聚類、選樣。
- (4). 概念分層通過收集並用較高層的概念替換較低層的概念來定義數值屬性的一個離散化。概念分層可以用來規約資料，通過這種概化儘管細節丟失了，但概化後的資料更有意義、更容易理解，並且所需的空間比原資料少。對於數值屬性，由於資料的可能取值範圍的多樣性和資料值的更新頻繁，說明概念分層是困難的。數值屬性的概念分層可以根據資料的分佈分析自動地構造，如用分箱、直方圖分析、聚類分析、基於熵的離散化和自然劃分分段等技術生成數值概念分層。由用戶專家在模式級顯示地說明屬性的部分序或全序，從而獲得概念的分層；只說明屬性集，但不說明它們的偏序，由系統根據每個屬性不同值的個數產生屬性序，自動構造有意義的概念分層。

#### (二)、資料預處理的方法

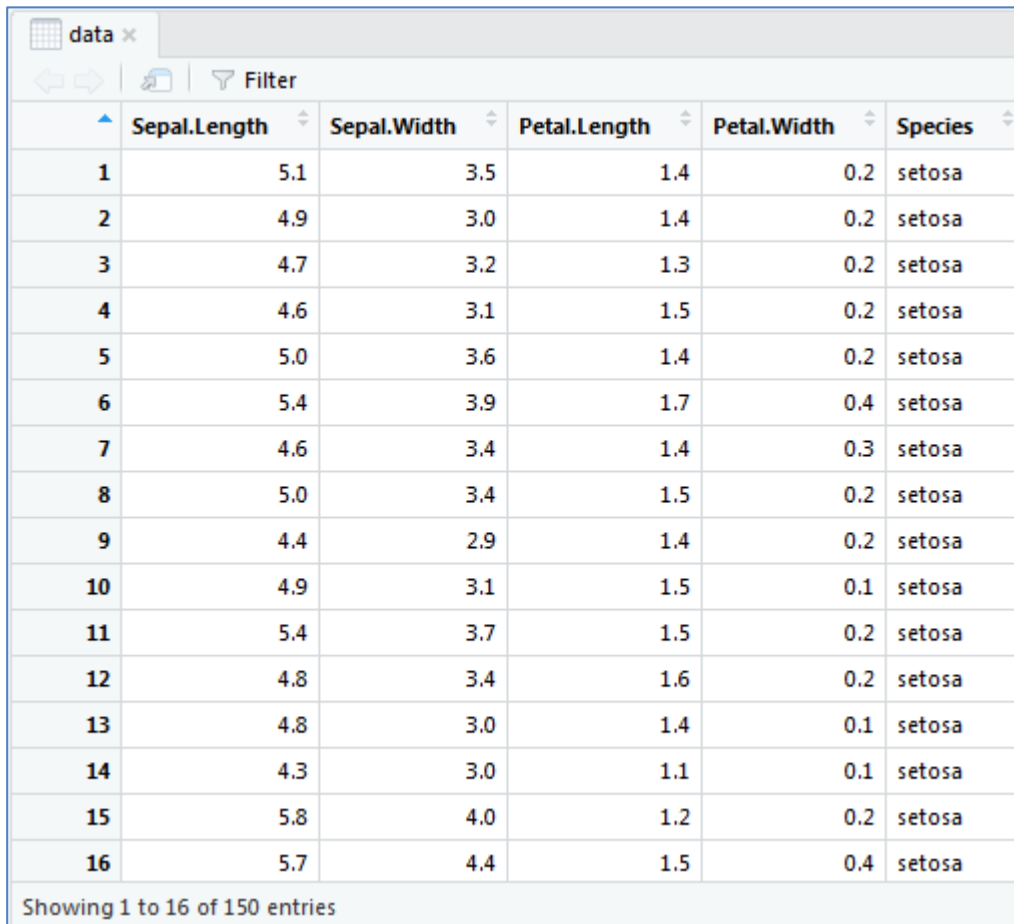
在資料分析的流程中，其實有 60% 以上的時間是在進行「資料預處理」，如果沒有好的資料，後續的分析就可能產生很大的偏誤。在「資料預處理」時，時常會遇到很多問題需要解決。當然，也有很多對應的小技巧，可以幫助處理這些問題。

## 1. 資料分割

想要將一個表格切割成不同的子表格時，會使用到以下的函式：`split()`、`subset()`，以下介紹其函式的用法。

- 使用 `split()` 函式進行資料分割：

```
> require(datasets) # source package
> data <- iris
> View(data)
```



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa

```
> split_data <- split(data, sample(rep(1:2, 75)))
> split_data
$`1`
  Sepal.Length Sepal.Width Petal.Length Petal.Width  Species
2           4.9         3.0         1.4         0.2    setosa
6           5.4         3.9         1.7         0.4    setosa
13          4.8         3.0         1.4         0.1    setosa
.....
149         6.2         3.4         5.4         2.3 virginica
```

```

150          5.9          3.0          5.1          1.8 virginica

$`2`
      Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1           5.1         3.5         1.4         0.2    setosa
3           4.7         3.2         1.3         0.2    setosa
4           4.6         3.1         1.5         0.2    setosa
.....
147          6.3         2.5         5.0         1.9 virginica
148          6.5         3.0         5.2         2.0 virginica

```

由於 `rep(1:2, 75)` 產生 1、2 交錯的向量，但加了前面的 `sample` 則是隨機抽取，所以向量 1、2 會被打亂，`split` 會依照 `sample(rep(1:2, 75))` 分組，都是 1 的會在同一組，都是 2 的也會在同一組。

- 使用 `subset()` 函式進行資料分割：

```

> require(datasets) # source package
> data <- iris
> subset(data, Sepal.Length > 5) # 只會出現 Sepal.Length>5 的資料

```

```

      Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1           5.1         3.5         1.4         0.2    setosa
6           5.4         3.9         1.7         0.4    setosa
11          5.4         3.7         1.5         0.2    setosa
15          5.8         4.0         1.2         0.2    setosa
16          5.7         4.4         1.5         0.4    setosa
.....

```

```

> subset(data, Sepal.Length == 5, select = c("Sepal.Length", "Species")) # 只會出現 Sepal.Length 等於 5 的資料，且欄位只會出現 Sepal.Length 和 Species

```

```

      Sepal.Length   Species
5                5    setosa
8                5    setosa
26               5    setosa
27               5    setosa
36               5    setosa
41               5    setosa
44               5    setosa
50               5    setosa

```



```
61          5 versicolor
94          5 versicolor
```

```
> subset(data, Sepal.Length > 5, select = - Sepal.Length) # select = 負的代表不要出現的欄位
```

	Sepal.Width	Petal.Length	Petal.Width	Species
1	3.5	1.4	0.2	setosa
6	3.9	1.7	0.4	setosa
11	3.7	1.5	0.2	setosa
15	4.0	1.2	0.2	setosa
16	4.4	1.5	0.4	setosa
17	3.9	1.3	0.4	setosa
18	3.5	1.4	0.3	setosa
19	3.8	1.7	0.3	setosa
20	3.8	1.5	0.3	setosa
21	3.4	1.7	0.2	setosa

.....

## 2. 資料合併

想要將兩筆資料合併時，會使用到以下的函式：rbind()、cbind()、merge()，以下介紹其函式的用法。

- 使用 rbind() 函式進行資料合併：

rbind() 可以用來追加資料，需要對應欄位元元（變數）名稱。

```
> # 首先先建立兩個 Data frame
> ID <- c(1,2,3,4)
> Name <- c("A","B","C","D")
> Student1 <- data.frame(ID, Name)
```

```
> ID <- c(5,6,7,8)
> Name <- c("E","F","G","H")
> Student2 <- data.frame(ID, Name)
```

```
> # 透過 row 合併
> rbind(Student1, Student2)
```

	ID	Name
1	1	A
2	2	B
3	3	C

4	4	D
5	5	E
6	6	F
7	7	G
8	8	H

- 使用 `cbind()` 函式進行資料合併：

`cbind()` 可以用來新增變數到原本的資料表格中，不需要對應欄位元元（變數）名稱。

```
> # 首先先建立兩個 Data frame
```

```
> ID <- c(1,2,3,4)
```

```
> Name <- c("A","B","C","D")
```

```
> Score <- c(60,70,80,90)
```

```
> Sex <- c("M","F","M","M")
```

```
> Student1 <- data.frame(ID, Name)
```

```
> Student2 <- data.frame(Score, Sex)
```

```
> # 透過 column 合併
```

```
> cbind(Student1, Student2)
```

	ID	Name	Score	Sex
1	1	A	60	M
2	2	B	70	F
3	3	C	80	M
4	4	D	90	M

- MySQL 的 INNER JOIN、LEFT JOIN、RIGHT JOIN、FULL OUTER JOIN、UNION (1)

MySQL 資料庫

資料庫名稱：employee

資料表格式：table1

```
CREATE TABLE table1 (  
  id char(5) NOT NULL,  
  name varchar(10) NOT NULL,  
  gender char(2) NOT NULL,  
  department varchar(12) NOT NULL,  
  salary int(11) NOT NULL,  
  PRIMARY KEY (id)  
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

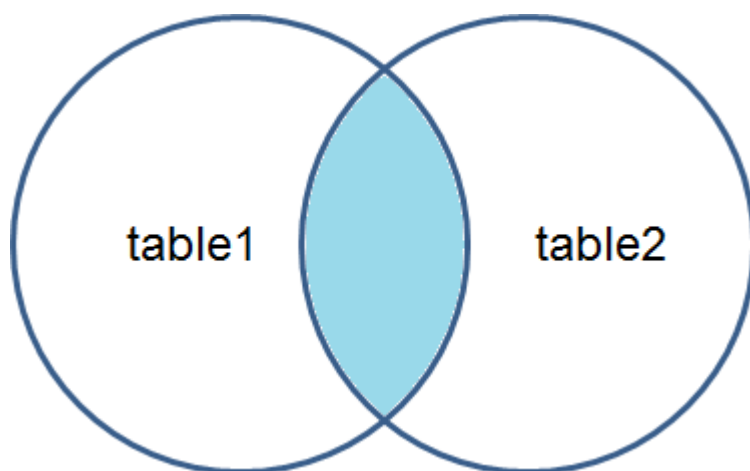
```
INSERT INTO table1 VALUES ('E0001', '蘇義光', '男', '業務部', 26500);
INSERT INTO table1 VALUES ('E0002', '黃定國', '男', '出納課', 31500);
INSERT INTO table1 VALUES ('E0003', '林美金', '女', '業務部', 40500);
INSERT INTO table1 VALUES ('E0004', '陳保川', '男', '業務部', 38000);
INSERT INTO table1 VALUES ('E0005', '李松界', '男', '製造廠', 30500);
```

資料表格式：table2

```
CREATE TABLE table2 (  
    id char(5) NOT NULL,  
    name varchar(10) NOT NULL,  
    gender char(2) NOT NULL,  
    department varchar(12) NOT NULL,  
    salary int(11) NOT NULL,  
    PRIMARY KEY (id)  
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

```
INSERT INTO table2 VALUES ('E0002', '黃定國', '男', '出納課', 31500);  
INSERT INTO table2 VALUES ('E0004', '陳保川', '男', '業務部', 38000);  
INSERT INTO table2 VALUES ('E0006', '王芳元', '女', '業務部', 34500);  
INSERT INTO table2 VALUES ('E0007', '李翔駿', '男', '製造廠', 32500);  
INSERT INTO table2 VALUES ('E0008', '簡麗艷', '女', '業務部', 36000);
```

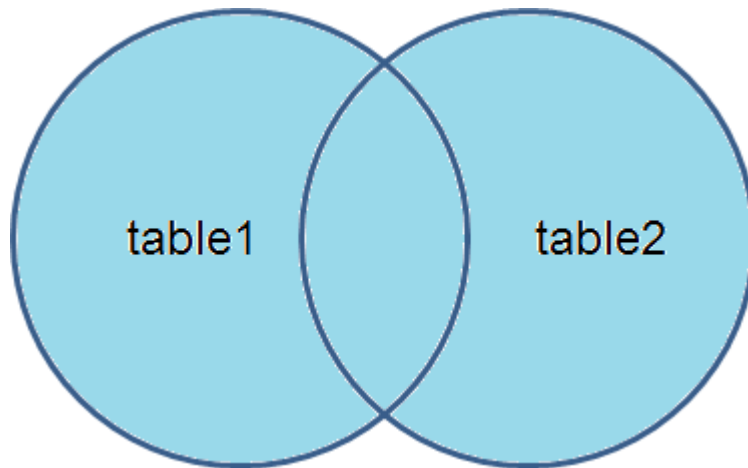
## 1. INNER JOIN



```
SELECT table1.*, table2.*  
FROM table1, table2  
WHERE table1.id = table2.id;
```

←T→									
id	name	gender	department	salary	id	name	gender	department	salary
E0002	黃定國	男	出納課	31500	E0002	黃定國	男	出納課	31500
E0004	陳保川	男	業務部	38000	E0004	陳保川	男	業務部	38000

## 2. FULL OUTER JOIN



MySQL 不支援全連接 (FULL OUTER JOIN)，解決方法：LEFT JOIN + RIGHT JOIN。

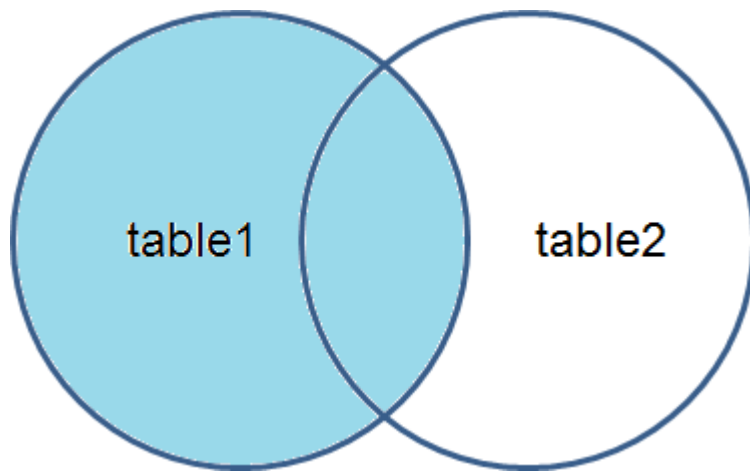
```

SELECT *
FROM table1
LEFT OUTER JOIN table2 ON table2.id = table1.id
UNION
SELECT *
FROM table1
RIGHT OUTER JOIN table2 ON table2.id = table1.id;

```

←T→									
id	name	gender	department	salary	id	name	gender	department	salary
E0001	蘇義光	男	業務部	26500	NULL	NULL	NULL	NULL	NULL
E0002	黃定國	男	出納課	31500	E0002	黃定國	男	出納課	31500
E0003	林美金	女	業務部	40500	NULL	NULL	NULL	NULL	NULL
E0004	陳保川	男	業務部	38000	E0004	陳保川	男	業務部	38000
E0005	李松界	男	製造廠	30500	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	E0006	王芳元	女	業務部	34500
NULL	NULL	NULL	NULL	NULL	E0007	李翔駿	男	製造廠	32500
NULL	NULL	NULL	NULL	NULL	E0008	簡麗艷	女	業務部	36000

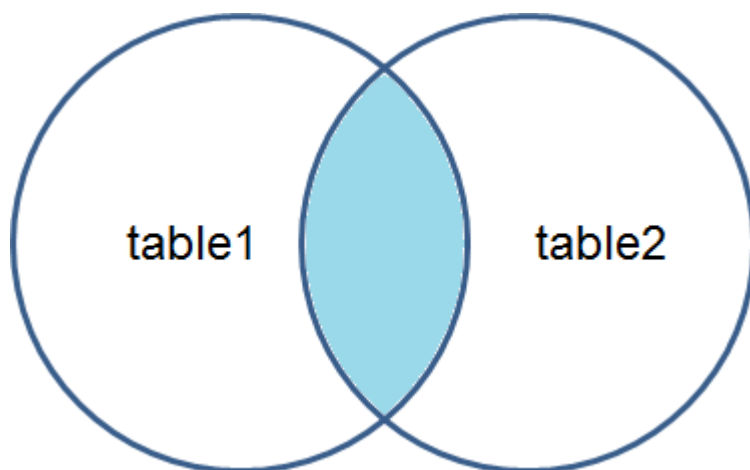
## 3. LEFT OUTER JOIN



```
SELECT *
FROM table1
LEFT JOIN table2 ON table2.id = table1.id;
```

← T →									
id	name	gender	department	salary	id	name	gender	department	salary
E0001	蘇義光	男	業務部	26500	NULL	NULL	NULL	NULL	NULL
E0002	黃定國	男	出納課	31500	E0002	黃定國	男	出納課	31500
E0003	林美金	女	業務部	40500	NULL	NULL	NULL	NULL	NULL
E0004	陳保川	男	業務部	38000	E0004	陳保川	男	業務部	38000
E0005	李松界	男	製造廠	30500	NULL	NULL	NULL	NULL	NULL

#### 4. INNER JOIN

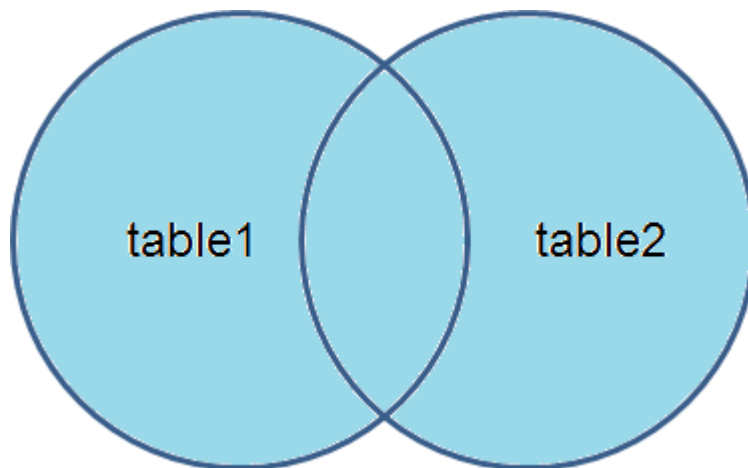


```
SELECT table1.*, table2.*
FROM table1, table2
```

WHERE table1.id = table2.id;

←T→									
id	name	gender	department	salary	id	name	gender	department	salary
E0002	黃定國	男	出納課	31500	E0002	黃定國	男	出納課	31500
E0004	陳保川	男	業務部	38000	E0004	陳保川	男	業務部	38000

## 5. FULL OUTER JOIN

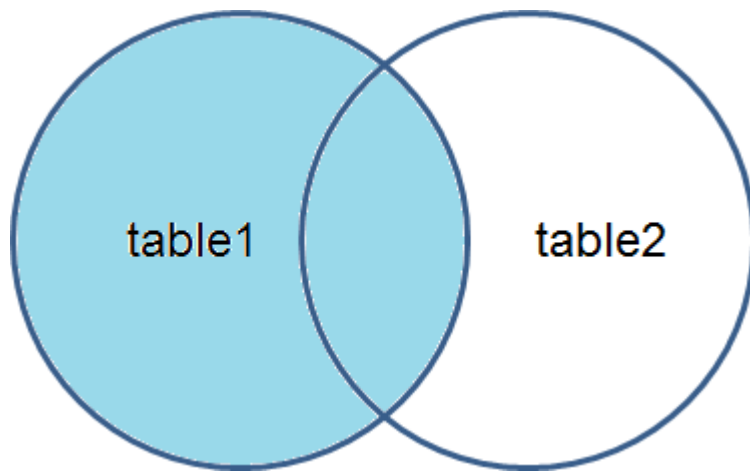


MySQL 不支援全連接 (FULL OUTER JOIN)，解決方法：LEFT JOIN + RIGHT JOIN。

```
SELECT *
FROM table1
LEFT OUTER JOIN table2 ON table2.id = table1.id
UNION
SELECT *
FROM table1
RIGHT OUTER JOIN table2 ON table2.id = table1.id;
```

←T→									
id	name	gender	department	salary	id	name	gender	department	salary
E0001	蘇義光	男	業務部	26500	NULL	NULL	NULL	NULL	NULL
E0002	黃定國	男	出納課	31500	E0002	黃定國	男	出納課	31500
E0003	林美金	女	業務部	40500	NULL	NULL	NULL	NULL	NULL
E0004	陳保川	男	業務部	38000	E0004	陳保川	男	業務部	38000
E0005	李松界	男	製造廠	30500	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	E0006	王芳元	女	業務部	34500
NULL	NULL	NULL	NULL	NULL	E0007	李翔駿	男	製造廠	32500
NULL	NULL	NULL	NULL	NULL	E0008	簡麗艷	女	業務部	36000

## 6. LEFT OUTER JOIN



```
SELECT *  
FROM table1  
LEFT JOIN table2 ON table2.id = table1.id;
```

←T→									
id	name	gender	department	salary	id	name	gender	department	salary
E0001	蘇義光	男	業務部	26500	NULL	NULL	NULL	NULL	NULL
E0002	黃定國	男	出納課	31500	E0002	黃定國	男	出納課	31500
E0003	林美金	女	業務部	40500	NULL	NULL	NULL	NULL	NULL
E0004	陳保川	男	業務部	38000	E0004	陳保川	男	業務部	38000
E0005	李松界	男	製造廠	30500	NULL	NULL	NULL	NULL	NULL

- **MySQL 的 INNER JOIN、LEFT JOIN、RIGHT JOIN、FULL OUTER JOIN、UNION (2)**

MySQL 資料庫

資料庫名稱：training

資料表格式：student

```
CREATE TABLE student (  
    student_id char(5) NOT NULL,  
    name varchar(10) NOT NULL,  
    gender char(2) NOT NULL,  
    department varchar(12) NOT NULL,  
    PRIMARY KEY (student_id)  
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

```
INSERT INTO student VALUES ('E0001', '蘇義光', '男', '資管系');
INSERT INTO student VALUES ('E0002', '黃定國', '男', '資工系');
INSERT INTO student VALUES ('E0003', '林美金', '女', '資管系');
INSERT INTO student VALUES ('E0004', '陳保川', '男', '資工系');
INSERT INTO student VALUES ('E0005', '李松界', '男', '資管系');
INSERT INTO student VALUES ('E0006', '王芳元', '女', '資工系');
INSERT INTO student VALUES ('E0007', '李翔駿', '男', '資管系');
INSERT INTO student VALUES ('E0008', '簡麗艷', '女', '資工系');
```

資料表格式：course

```
CREATE TABLE course (
  course_id char(6) NOT NULL,
  title varchar(20) NOT NULL,
  PRIMARY KEY (course_id)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

```
INSERT INTO course VALUES ('A10011', 'Big Data 大數據分析');
INSERT INTO course VALUES ('B20032', 'AI 人工智慧應用');
INSERT INTO course VALUES ('C30023', 'IoT 物聯網實務');
```

資料表格式：enroll

```
CREATE TABLE enroll (
  course_id char(6) NOT NULL,
  student_id char(5) NOT NULL,
  PRIMARY KEY (course_id, student_id)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

```
INSERT INTO enroll VALUES ('A10011', 'E0001');
INSERT INTO enroll VALUES ('A10011', 'E0002');
INSERT INTO enroll VALUES ('A10011', 'E0003');
INSERT INTO enroll VALUES ('A10011', 'E0005');
INSERT INTO enroll VALUES ('A10011', 'E0007');
INSERT INTO enroll VALUES ('A10011', 'E0008');
```

```
SELECT student.*, enroll.*
FROM student, enroll
WHERE student.student_id = enroll.student_id
ORDER BY student.student_id;
```



student_id ▲ 1	name	gender	department	course_id	student_id
E0001	蘇義光	男	資管系	A10011	E0001
E0002	黃定國	男	資工系	A10011	E0002
E0003	林美金	女	資管系	A10011	E0003
E0005	李松昇	男	資管系	A10011	E0005
E0007	李翔駿	男	資管系	A10011	E0007
E0008	簡麗艷	女	資工系	A10011	E0008

```

SELECT student.*, enroll.*, course.*
FROM student, enroll, course
WHERE student.student_id = enroll.student_id
      AND enroll.course_id = course.course_id
ORDER BY student.student_id;

```

student_id ▲ 1	name	gender	department	course_id	student_id	course_id	title
E0001	蘇義光	男	資管系	A10011	E0001	A10011	Big Data大數據分析
E0002	黃定國	男	資工系	A10011	E0002	A10011	Big Data大數據分析
E0003	林美金	女	資管系	A10011	E0003	A10011	Big Data大數據分析
E0005	李松昇	男	資管系	A10011	E0005	A10011	Big Data大數據分析
E0007	李翔駿	男	資管系	A10011	E0007	A10011	Big Data大數據分析
E0008	簡麗艷	女	資工系	A10011	E0008	A10011	Big Data大數據分析

```

SELECT *
FROM student
LEFT OUTER JOIN enroll ON enroll.student_id = student.student_id
UNION
SELECT *
FROM enroll
RIGHT OUTER JOIN student ON enroll.student_id = student.student_id;

```

student_id	name	gender	department	course_id	student_id
E0001	蘇義光	男	資管系	A10011	E0001
E0002	黃定國	男	資工系	A10011	E0002
E0003	林美金	女	資管系	A10011	E0003
E0005	李松界	男	資管系	A10011	E0005
E0007	李翔駿	男	資管系	A10011	E0007
E0008	簡麗艷	女	資工系	A10011	E0008
E0004	陳保川	男	資工系	NULL	NULL
E0006	王芳元	女	資工系	NULL	NULL
A10011	E0001	E0001	蘇義光	男	資管系
A10011	E0002	E0002	黃定國	男	資工系
A10011	E0003	E0003	林美金	女	資管系
A10011	E0005	E0005	李松界	男	資管系
A10011	E0007	E0007	李翔駿	男	資管系
A10011	E0008	E0008	簡麗艷	女	資工系
NULL	NULL	E0004	陳保川	男	資工系
NULL	NULL	E0006	王芳元	女	資工系

- 使用 merge() 函式進行資料合併：

merge() 能夠依據兩個表格中共同有的欄位元（變數）名稱來合併資料

> # 首先先建立兩個 data frame

```
> df1 <- data.frame(CustomerId = c(1:5), Product = c(rep("Toaster", 3), rep("Radio", 2)))
```

```
> df1
```

```
  CustomerId Product
1          1  Toaster
2          2  Toaster
3          3  Toaster
4          4   Radio
5          5   Radio
```

```
> df2 <- data.frame(CustomerId = c(2, 4, 6), State = c(rep("Alabama", 2), rep("Ohio", 1)))
```

```
> df2
```

```
  CustomerId  State
1          2 Alabama
2          4 Alabama
```

3                      6      Ohio

將兩個 data frame 透過 CustomerId 欄位進行合併：

> # INNER JOIN，保留兩資料集 CustomerId 欄位中，取交集的值來合併

> merge(x = df1, y = df2, by = "CustomerId")

	CustomerId	Product	State
1	2	Toaster	Alabama
2	4	Radio	Alabama

mrge() 函式的第 1、2 參數是指定要合併的資料表，而 by 參數則是指定資料辨識的依據欄位。

> # FULL JOIN，保留兩資料集 CustomerId 欄位中，取聯集的值來合併

> merge(x = df1, y = df2, by = "CustomerId", all = TRUE)

	CustomerId	Product	State
1	1	Toaster	<NA>
2	2	Toaster	Alabama
3	3	Toaster	<NA>
4	4	Radio	Alabama
5	5	Radio	<NA>
6	6	<NA>	Ohio

all 是用來詢問是否顯示所有欄位的資料。

> # LEFT JOIN，保留 x (df1 表格) CustomerId 欄位元中的值來合併

> merge(x = df1, y = df2, by = "CustomerId", all.x = TRUE)

	CustomerId	Product	State
1	1	Toaster	<NA>
2	2	Toaster	Alabama
3	3	Toaster	<NA>
4	4	Radio	Alabama
5	5	Radio	<NA>

> # RIGHT JOIN，保留 y (df2 表格) CustomerId 欄位元中的值來合併

> merge(x = df1, y = df2, by = "CustomerId", all.y = TRUE)

	CustomerId	Product	State
1	2	Toaster	Alabama
2	4	Radio	Alabama
3	6	<NA>	Ohio

要注意，merge() 針對兩筆具有共同變數的資料進行合併，由於 merge() 針對 by 參數所指定的變數做交叉比對，因此該變數的編碼值必須是「單一獨立」且不能「重複」。例如「學號」、「身分證號」等，否則 merge 會出現個案增多的錯誤結果。

### 3. 處理離群值 (outlier)

在資料探勘的流程中，資料中是否存在離群值 (outlier)，可能會嚴重影響到資料分析的結果，甚至會影響到模式建立的正確性。因此判斷離群值的方法便相當重要，以下將介紹兩種以敘述統計為基礎的離群值判斷方法，包括標準化分數、盒鬚圖。

#### ● 標準化分數判斷：

將資料轉成標準化分數 (Z 分數) 進行判斷，根據常態分配的性質，約有 99% 資料的 Z 分數會落在平均值的 3 倍標準差之內，因此 Z 分數大於 3 或小於 -3 的資料將視為離群值 (可自訂其他資料為切割點)。

標準分數是教育統計學名詞。乃一群體中 (如一班學生)，各原始分數與平均數的差數，以標準差除之，就成為以標準差為單位的量數，稱為標準分數或 z 分數 (z score)。

標準分數又稱為 Z 分數或真分數，是以標準差為單位來表示一個分數在團體中所處位置的相對位置量數。

標準化分數的計算方式，可以使用 `scale(raw, center, scale)` 函數

- raw：原始分數，可以是一個數字，也可以是一個數列，如 `c(100, 98, 87, 86, 85, 83, ...)`。
- center：平均數
- scale：標準差，這樣子就可以跑出 Z 分數

#### 【計算標準化分數】

```
> set.seed(10) # 為了能夠得到一樣的向量 y
> score <- sample(c(0:100), size=60, replace=T)
> score
[1] 51 30 43 70 8 22 27 27 62 43 65 57 11 60 36 43 5 26 40 84 87 62 78
35 40 71 84 24 77 35 54 9 17 90 42 75
[37] 83 96 69 50 27 23 1 73 25 16 1 49 10 80 35 94 24 47 19 58 46 47 40
51

> mean(score) # 平均數
[1] 45.9
```

```
> sd(score) # 標準差
[1] 25.70873
```

將上列數字進行標準化：

```
> center <- mean(score) # 設定平均數
> scale <- sd(score) # 設定標準差
> z.score <- round(scale(score, center, scale), 2) # 進行標準化，並
將結果四捨五入至小數點第二位。
```

```
> z.score
      [,1]
[1,]  0.20
[2,] -0.62
[3,] -0.11
[4,]  0.94
[5,] -1.47
.....
[58,]  0.04
[59,] -0.23
[60,]  0.20
attr(,"scaled:center")
[1] 45.9
attr(,"scaled:scale")
[1] 25.70873
```

```
> require(datasets) # source package
> data <- iris
> data <- subset(data, select = - Species) # 去除不是數值型態的 column
```

```
> scale_data <- scale(data, center = TRUE, scale = TRUE) # 標準化
表格中的數值
```

```
> scale_data
      Sepal.Length Sepal.Width Petal.Length  Petal.Width
1    -0.89767388  1.01560199  -1.33575163 -1.3110521482
2    -1.13920048 -0.13153881  -1.33575163 -1.3110521482
3    -1.38072709  0.32731751  -1.39239929 -1.3110521482
4    -1.50149039  0.09788935  -1.27910398 -1.3110521482
```

```
5    -1.01843718  1.24503015 -1.33575163 -1.3110521482
6    -0.53538397  1.93331463 -1.16580868 -1.0486667950
```

.....

```
attr(,"scaled:center")
```

```
Sepal.Length Sepal.Width Petal.Length  Petal.Width
      5.843333      3.057333      3.758000      1.199333
```

```
attr(,"scaled:scale")
```

```
Sepal.Length Sepal.Width Petal.Length  Petal.Width
      0.8280661      0.4358663      1.7652982      0.7622377
```

```
> scale_data <- as.data.frame(scale_data) # 轉成 Data frame 型態
```

```
> scale_data
```

```
      Sepal.Length Sepal.Width Petal.Length  Petal.Width
1    -0.89767388  1.01560199 -1.33575163 -1.3110521482
2    -1.13920048 -0.13153881 -1.33575163 -1.3110521482
3    -1.38072709  0.32731751 -1.39239929 -1.3110521482
4    -1.50149039  0.09788935 -1.27910398 -1.3110521482
5    -1.01843718  1.24503015 -1.33575163 -1.3110521482
6    -0.53538397  1.93331463 -1.16580868 -1.0486667950
```

.....

```
> scale_data <- subset(scale_data, Sepal.Length < 2 & Sepal.Width < 2 & Petal.Length < 2 & Petal.Width < 2) # 留下全部欄位元中，Z 分數小於 2 的值
```

```
> scale_data
```

```
      Sepal.Length Sepal.Width Petal.Length  Petal.Width
1    -0.89767388  1.01560199 -1.33575163 -1.3110521482
2    -1.13920048 -0.13153881 -1.33575163 -1.3110521482
3    -1.38072709  0.32731751 -1.39239929 -1.3110521482
4    -1.50149039  0.09788935 -1.27910398 -1.3110521482
5    -1.01843718  1.24503015 -1.33575163 -1.3110521482
6    -0.53538397  1.93331463 -1.16580868 -1.0486667950
```

.....

```
> scale_data <- subset(scale_data, Sepal.Length > -2 & Sepal.Width > -2 & Petal.Length > -2 & Petal.Width > -2) # 留下全部欄位元中，Z 分數大於 -2 的值
```

```
> scale_data
```

```
      Sepal.Length Sepal.Width Petal.Length  Petal.Width
```

```

1    -0.89767388  1.01560199 -1.33575163 -1.3110521482
2    -1.13920048 -0.13153881 -1.33575163 -1.3110521482
3    -1.38072709  0.32731751 -1.39239929 -1.3110521482
4    -1.50149039  0.09788935 -1.27910398 -1.3110521482
5    -1.01843718  1.24503015 -1.33575163 -1.3110521482
6    -0.53538397  1.93331463 -1.16580868 -1.0486667950
.....

```

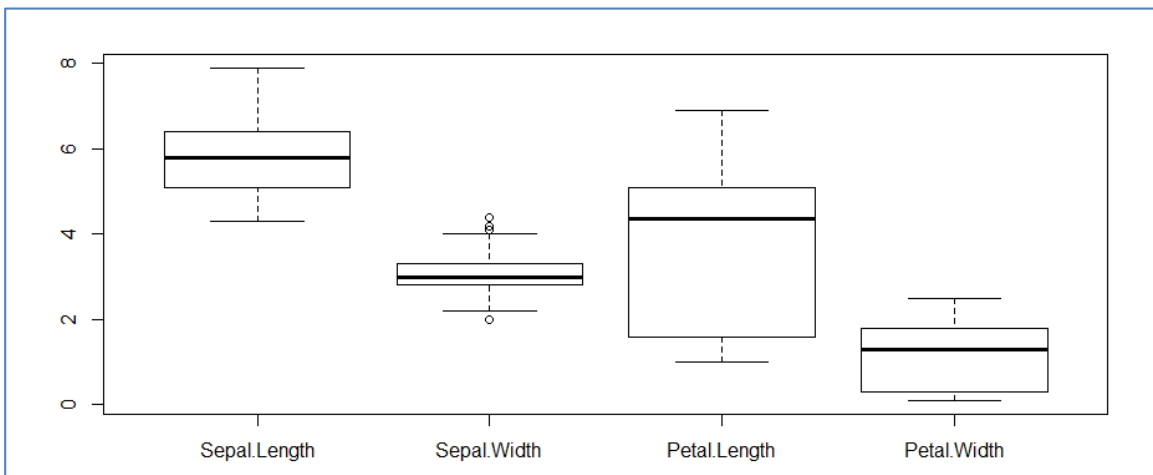
### ● 盒鬚圖判斷：

Tukey (1977) 將變數中任何位於內籬 (inner fence) 與外籬之間的資料視為該變數的潛在離群值。另外，如果變數中有任何資料位於外籬 (outer fence) 之外的，則視它們為該變數的離群值，外籬指的是 Q1 向下延伸或 Q3 向上延伸 1.5 倍 IQR 的距離。

```

> require(datasets) # source package
> data <- iris
> data <- subset(data, select = - Species) # 去除不是數值型態的 column
> boxplot(data) # 繪製盒鬚圖

```



```

> summary(data)

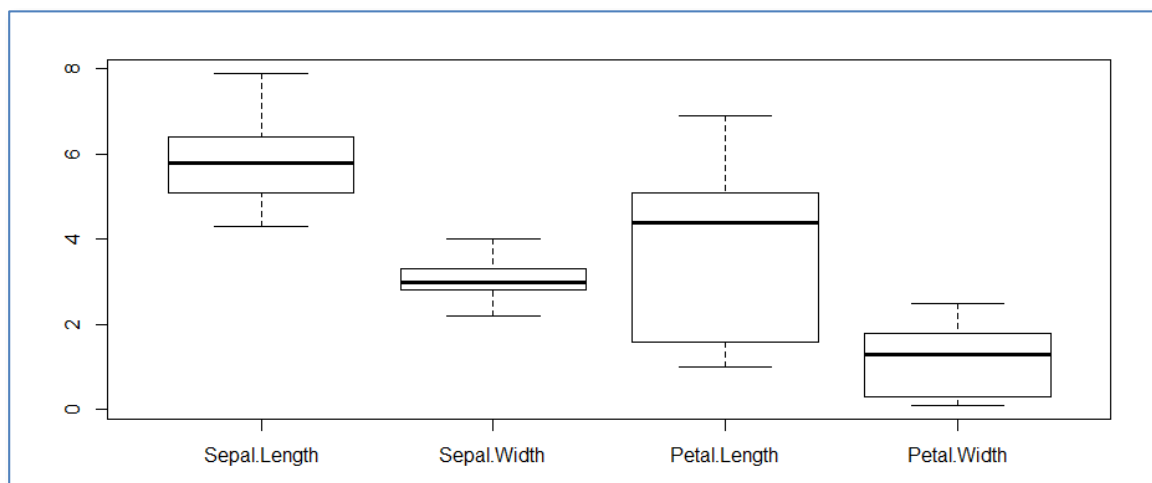
```

Sepal.Length	Sepal.width	Petal.Length	Petal.width
Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100
1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.300
Median :5.800	Median :3.000	Median :4.350	Median :1.300
Mean :5.843	Mean :3.057	Mean :3.758	Mean :1.199
3rd Qu.:6.400	3rd Qu.:3.300	3rd Qu.:5.100	3rd Qu.:1.800
Max. :7.900	Max. :4.400	Max. :6.900	Max. :2.500

```

> 1.5 * IQR(data$Sepal.Width)
[1] 0.75
> boxplot(data)$out # 或使用 boxplot()$out 函式看有哪些離群值
[1] 4.4 4.1 4.2 2.0
> data <- subset(data, Sepal.Width < 4.05 & Sepal.Width > 2.05) #
  留下 Sepal.Width 欄位中，數值小於 4.05 或大於 2.05 的值
> boxplot(data)

```



處理離群值時，首先應考量懷疑為離群值的資料是否可以被解釋，如果可以，則可依合理的原則處理，例如資料完全不合理即可移除；但如果資料經查證後，不但無誤，而且發現該離群值是來自於非常特殊的個案，我們應該深入瞭解其資料為何如此特別，且必須深入探討決定應該刪除抑或保留該資料。

#### 4. 轉虛擬變數 (Dummy variable)

在迴歸分析（線性、羅吉斯...等）中，當自變數為類別變數時，都要先進行轉換虛擬變數 (Dummy variable) 的動作，以人工變數量化類別變數，通常取值為 0 或 1。

- 使用 dummies 套件轉換：

```

> install.packages('dummies')
> require(dummies) # 轉換虛擬變數的套件
Loading required package: dummies
dummies-1.5.6 provided by Decision Patterns
> data <- iris
> alldummy_data <- dummy.data.frame(data)

```

函式會自動抓取表格中，欄位元資料型態屬於 factor 的 column 轉換成虛擬變數。



```
> justdummy_data <- dummy.data.frame(data, all = F)
```

all = F，可以只顯示出轉換後的虛擬變數欄位。

- 使用 `model.matrix` 函式轉換：

```
> data <- iris
```

```
> justdummy_data <- model.matrix(~data$Species-1) # 轉換出只有虛擬變數的欄位
```

```
> alldummy_data <- cbind(data, justdummy_data) # 合併表格 by column
```

注意：利用 `model.matrix` 函式轉換的欄位資料型態要為 `factor`。

引入虛擬變數會使得原本的模型變得更複雜，但對於問題的解釋會更清楚，也就是一個方程式能達到兩個方程式的概念，而且較接近現實。要注意，在模型中引入多個虛擬變數時，虛擬變數的個數要遵守轉換原則：如果在類別變數欄位中有  $n$  種互斥的屬性，則只在模型中引入  $(n-1)$  個虛擬變數。

### (三)、資料缺失

當處理含有缺失值 (NA) 的資料時，可以運用以下幾種最常見的策略：

- 將含有缺失值的案例剔除。
- 根據變數之間的相關性關係填補缺失值。
- 根據案例之間的相似性填補缺失值。
- 使用能夠處理缺失值資料的工具。

下文的例子都用到了 `DMwR` 套件，讀取資料代碼如下：

```
> install.packages('DMwR')
```

```
> library(DMwR)
```

```
Loading required package: lattice
```

```
Loading required package: grid
```

```
> data(algae)
```

```
> algae
```

	season	size	speed	mxPH	mnO2	C1	NO3	NH4	oPO4	P
04	chl a	a1	a2	a3	a4	a5				
1	winter	small	medium	8.000	9.80	60.800	6.238	578.000	105.000	17
	0.000	50.000	0.0	0.0	0.0	0.0	34.2			
2	spring	small	medium	8.350	8.00	57.750	1.288	370.000	428.750	55
	8.750	1.300	1.4	7.6	4.8	1.9	6.7			

```

3  autumn  small medium 8.100 11.40 40.020 5.330 346.667 125.667 1
87.057 15.600 3.3 53.6 1.9 0.0 0.0
4  spring  small medium 8.070 4.80 77.364 2.302 98.182 61.182 13
8.700 1.400 3.1 41.0 18.9 0.0 1.4
5  autumn  small medium 8.060 9.00 55.350 10.416 233.700 58.222 9
7.580 10.500 9.2 2.9 7.5 0.0 7.5

```

.....

```
[ reached getOption("max.print") -- omitted 145 rows ]
```

> View(algae)

### 1. 將缺失部分剔除

將含有缺失值的案例剔除非常容易實現，尤其是當這些記錄所占的比例在可用資料集中非常小的時候，這個選擇就比較合理。因此，在選擇這個方案時先檢查觀測值，或者至少得到這些觀測值的個數。

如果直接刪除所有含有至少一個 NA 的樣本，我們可以輸入：

```

> algae <- na.omit(algae)
> algae
      season  size speed mxPH mnO2      Cl  NO3      NH4      oPO4      P
04  Chla    a1  a2  a3  a4  a5
1  winter  small medium 8.000 9.80 60.800 6.238 578.000 105.000 17
0.000 50.000 0.0 0.0 0.0 0.0 34.2
2  spring  small medium 8.350 8.00 57.750 1.288 370.000 428.750 55
8.750 1.300 1.4 7.6 4.8 1.9 6.7
3  autumn  small medium 8.100 11.40 40.020 5.330 346.667 125.667 1
87.057 15.600 3.3 53.6 1.9 0.0 0.0
4  spring  small medium 8.070 4.80 77.364 2.302 98.182 61.182 13
8.700 1.400 3.1 41.0 18.9 0.0 1.4
5  autumn  small medium 8.060 9.00 55.350 10.416 233.700 58.222 9
7.580 10.500 9.2 2.9 7.5 0.0 7.5
6  winter  small high 8.250 13.10 65.750 9.248 430.000 18.250 5
6.667 28.400 15.1 14.6 1.4 0.0 22.5

```

.....

```
[ reached getOption("max.print") -- omitted 129 rows ]
```

不過這種辦法太過極端，一般不採用。但是對於某些缺失值太多的樣本可以直接剔除，因為他們幾乎是無用的樣本。觀測方法可通過如下代碼：

```
> data(algae)
```



填補缺失資料最簡便和快捷的方法是使用一些代表中心趨勢的值。代表中心趨勢的值反映了變數分佈的最常見值，因此中心趨勢值是最自然的選擇。然而，中心趨勢值也有很多種，如平均值、中位數、眾數等。如何選擇還要由變數的分佈決定。對於接近正態分佈來說，所有的觀測值都較好地聚集在平均值周圍，平均數就是最佳選擇。然而對於偏態分佈，平均值就不適用。另一方面，離群值（極值）的存在會扭曲平均值（這些可以通過箱式圖觀測到）。

- 用平均值填補：

```
> data(algae)
> algae[48,"mxPH"] <- mean(algae$mxPH,na.rm=T)
```

- 用中位數填補：

```
> data(algae)
> algae[is.na(algae$Chla),"Chla"] <- median(algae$Chla,na.rm=T)
```

其中，na.rm 是使計算時忽略缺失資料。當然，例子中用到的套件下也提供了一個函數 `centralImputation()` 可以用資料的中心趨勢值來填補缺失值。對數值型變數使用中位數，對名義變數使用眾數。應用如下：

```
> data(algae)
> algae <- algae[-manyNAs(algae),]
> algae <- centralImputation(algae)
```

上述方法雖然快捷方便，但是它可能導致較大的資料偏差，影響後期的資料分析工作。

### 3. 通過變數的相關關係來填補缺失值

另一種獲得較少偏差填補缺失值的方法是探尋變數之間的相關關係。

```
> symnum(cor(algae[,4:18], use="complete.obs"))
```

```
      mP mO C1 NO NH o P Ch a1 a2 a3 a4 a5 a6 a7
mxPH 1
mnO2   1
C1      1
NO3      1
NH4      , 1
oPO4    . . 1
PO4     . . * 1
Chla .           1
a1      . . . 1
a2     . . . 1
a3      . . 1
```

```

a4      .      .      1
a5      .      .      1
a6      .      .      .      1
a7      .      .      .      1
attr(,"legend")
[1] 0 ' ' 0.3 '.' 0.6 ',' 0.8 '+' 0.9 '*' 0.95 'B' 1

```

函數 `cor()` 的功能是產生變數之間的相關值矩陣，設定參數 `use="complete.obs"` 可以使 R 在計算相關值時忽略含有 NA 的紀錄。而函數 `symnum()` 是用來改善結果的輸出形式的。

在找到相關性較高的兩個變數後，開始尋找他們之間的線性相關關係，如下：

```

> data(algae)
> algae <- algae[-manyNAs(algae),]
> lm(PO4~oPO4, data=algae)

```

```

Call:
lm(formula = PO4 ~ oPO4, data = algae)

```

```

Coefficients:
(Intercept)      oPO4
      42.897      1.293

```

然後，通過線性關係計算缺失值的填補值。

如果 PO4 中存在多個缺失值，也可以通過構造一個函數來完成，如下：

```

> data(algae)
> algae <- algae[-manyNAs(algae),]
> fillPO4 <- function(oP) {
+   if(is.na(oP))
+     return(NA)
+   else return(42.897+1.293*oP)
+ }
> algae[is.na(algae$PO4),"PO4"] <- sapply(algae[is.na(algae$PO
4),"oPO4"],fillPO4)

```

函數 `sapply()` 的第 1 個參數是一個向量，第 2 個參數是一個函數。作用是將函數結果應用到第 1 個參數中向量的每一個元素。

在上面的例子中由於使用的是相關值，所以只用到了數值變數而排除了名義變數與缺失值之間的關係。

#### 4. 通過探索案例之間的相似性來填補缺失值

除了變數之間的相關性外，多個樣本（行）之間的相似性也可以用來填補缺失值。度量相似性的指標有很多，常用的是歐式距離，這個距離可以非正式的定義為任何兩個案例之的觀測值之差的平方和。書中提供的包的函數 `knnImputation()` 可以實現上述操作，這個函數用一個歐式距離的變種來找到任何個案最近的 `k` 個鄰居。使用方法如下：

```
> data(algae)
> algae <- knnImputation(algae,k=10)
```

- 用中位數來填補：

```
> data(algae)
> algae <- knnImputation(algae,k=10, meth="median")
```

#### 5. 缺失值處理

```
> install.packages('lattice')
> install.packages('MASS')
> install.packages('nnet')
> library(lattice)
> library(MASS)
> library(nnet)
> install.packages('mice')
> library(mice)
> nhanes2
```

	age	bmi	hyp	chl
1	20-39	NA	<NA>	NA
2	40-59	22.7	no	187
3	20-39	NA	no	187
4	60-99	NA	<NA>	NA
5	20-39	20.4	no	113
6	60-99	NA	<NA>	184
7	20-39	22.5	no	118
8	20-39	30.1	no	187
9	40-59	22.0	no	238
10	40-59	NA	<NA>	NA
11	20-39	NA	<NA>	NA
12	40-59	NA	<NA>	NA
13	60-99	21.7	no	206

```

14 40-59 28.7 yes 204
15 20-39 29.6 no NA
16 20-39 NA <NA> NA
17 60-99 27.2 yes 284
18 40-59 26.3 yes 199
19 20-39 35.3 no 218
20 60-99 25.5 yes NA
21 20-39 NA <NA> NA
22 20-39 33.2 no 229
23 20-39 27.5 no 131
24 60-99 24.9 no NA
25 40-59 27.4 no 186

```

```
> dim(nhanes)
```

```
[1] 25 4
```

```
> summary(nhanes2)
```

age	bmi	hyp	chl
20-39:12	Min. :20.40	no :13	Min. :113.0
40-59: 7	1st Qu.:22.65	yes : 4	1st Qu.:185.0
60-99: 6	Median :26.75	NA's: 8	Median :187.0
	Mean :26.56		Mean :191.4
	3rd Qu.:28.93		3rd Qu.:212.0
	Max. :35.30		Max. :284.0
	NA's :9		NA's :10

```
> sum(is.na(nhanes2)) # 計算缺失值的數量
```

```
[1] 27
```

```
> sum(complete.cases(nhanes2)) # 計算完整樣本的數量
```

```
[1] 13
```

```
> md.pattern(nhanes2) # 觀測缺失值的情況
```

	age	hyp	bmi	chl	
13	1	1	1	1	0
3	1	1	1	0	1
1	1	1	0	1	1
1	1	0	0	1	2
7	1	0	0	0	3
	0	8	9	10	27

- 插補法：用隨機抽樣填補

```

> sub <- which(is.na(nhanes2[,4]) == TRUE) # 返回數據集第 4 行為 NA
之行
> dataTR <- nhanes2[-sub,] # 將第 4 行不為 NA 的數據存入 dataTR 中
> dataTE <- nhanes2[sub,] # 將第 4 行為 NA 的數據存入 dataTE 中
> dataTE[,4] <- sample(dataTR[,4], length(dataTE[,4]), replace=T)
# 在非缺失值中簡單抽樣
> dataTE
  age  bmi hyp chl
1   1   NA  NA 284
4   3   NA  NA 118
10  2   NA  NA 218
11  1   NA  NA 118
12  2   NA  NA 187
15  1 29.6   1 284
16  1   NA  NA 238
20  3 25.5   2 187
21  1   NA  NA 199
24  3 24.9   1 186

```

- 插補法：用平均值填補

```

> sub <- which(is.na(nhanes2[,4]) == TRUE) # 返回數據集第 4 行為 NA
之行
> dataTR <- nhanes2[-sub,] # 將第 4 行不為 NA 的數據存入 dataTR 中
> dataTE <- nhanes2[sub,] # 將第 4 行為 NA 的數據存入 dataTE 中
> dataTE[,4] <- mean(dataTR[,4]) # 用非缺失值的平均值代替缺失值
> dataTE

```

## R 筆記-(10)遺漏值處理(Impute Missing Value)

[http://www.rpubs.com/skydome20/R-Note10-Missing\\_Value](http://www.rpubs.com/skydome20/R-Note10-Missing_Value)



## 十、R 程式語言設計

```
> a <- c(1,2,3)
> x <- a+2
> x
[1] 3 4 5
```

<https://yijutseng.github.io/DataScienceRBook/io.html#open-data>  
<https://www.tutorialspoint.com/r/index.htm>

```
> x <- a+2
> (x <- a+2)
[1] 3 4 5
```

R 程式是一種表達式或運算式語言 (Expression language) 其任何一個述述句都可以看成是一個表達式或運算式，它有指令形式與傳回結果的函數之運算，**表達式或運算式可以續行，只要前一行不是完整的表達式或運算式，則下一行為上一行的繼續。**表達式或運算式之間以分號分隔或用換行分隔。R 也是一種高階程式語言 (Programming language)，因此提供了其它程式語言共有的條件 (if-else)、轉換 (switch)、迴圈 (loop) 等程式控制結構語法。

多個表達式或運算式可以放在一起，組成一個更大的複合表達式或運算式。多個指令、表達式或運算式可以用大括號 {} 包圍在一起，如：

```
{expr_1; ... ; expr_n}
```

此時，這一組的複合表達式或運算式的結果，是該組中最後一個指令的回傳的值。

```
> {a <- c(1,2,3); x <- a+2}
> x
[1] 3 4 5
```

### (一)、條件控制 if-else 敘述

R 程式語言的基本條件語句形式為

- **if (test\_expression) statement**
- **if (test\_expression) statement1 else statement2**

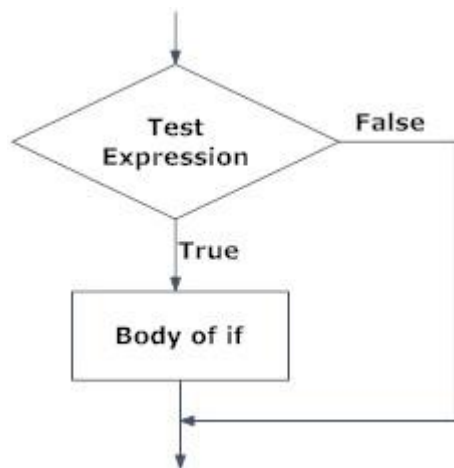
其中 **test\_expression** 是控制條件，**test\_expression** 是一個邏輯操作，判斷控制條件為「真」或「假」(TRUE or FALSE)，**test\_expression** 回傳一個純量，若 **test\_expression** 回傳 TRUE，則執行 **statement1** 之表達式或運算式；若 **test\_expression** 回傳 FALSE，則 else 執行 **statement2** 之表達式或運算式；若無 **statement2**，不進行任何運算，回傳一個

NULL。

- if 敘述的語法

```
if (test_expression) {  
  statement  
}
```

- if 敘述的流程圖



- if 敘述的程式範例 (程式名稱：if\_test-01.R)

```
x <- 5  
if (x > 0) {  
  print("正數")  
}
```

\*執行結果：

```
[1] "正數"
```

---

```
x <- 3.12345  
if (x<5) Y=10  
  
print(x, digits=3)  
print(paste("X =", x))  
print(paste("Y =", Y))  
cat("Y =", Y)
```

\*執行結果：

```
[1] 3.12
```

```
[1] "x = 3.12345"  
[1] "Y = 10"  
Y = 10
```

```
X <- 3; Y <- 1  
if (X<5 && Y<5) {  
  Y <- 10; Z <- 5  
}  
cat("Y =", Y)  
cat("Z =", Z)
```

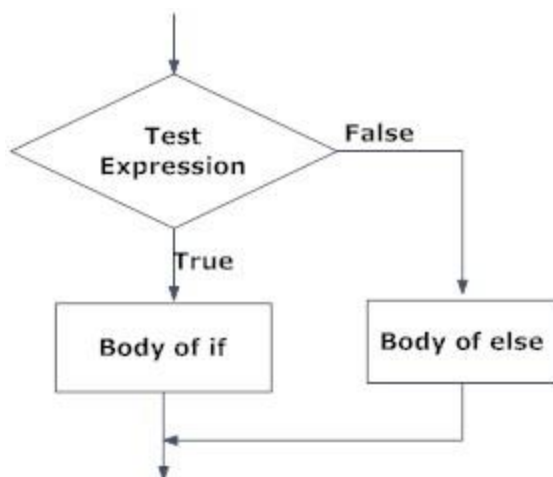
\*執行結果：

```
Y = 10  
Z = 5
```

- if...else...敘述的語法

```
if (test_expression) {  
  statement 1  
} else {  
  statement 2  
}
```

- if...else...敘述的流程圖



- if...else...敘述的程式範例 (程式名稱：if\_test-02.R)

```
X <- 20  
Y = ifelse(X>5, 2, 3)
```

```
cat("Y =", Y)
```

\*執行結果：

```
Y = 2
```

---

```
x <- -5
if (x > 0) {
  print("正數")
} else {
  print("負數")
}
```

\*執行結果：

```
[1] "負數"
```

---

```
x <- 6
if (x>5) y=2 else y=4
cat("y =", y)
```

\*執行結果：

```
y = 2
```

---

```
weather <- sample(c("晴天", "雨天"), size=1)
if (weather == "晴天") {
  print("到戶外做運動！")
} else {
  print("在體育館活動！")
}
```

\*執行結果：

```
[1] "到戶外做運動！"
```

---

```
x <- readline("請輸入一個數值(1~10)：")

if (x>5) {
  y = 2
} else {
  y = 4
}
```

```
cat("運算結果：y =", y)
```

**\*執行結果：**

請輸入一個數值(1~10)：1

運算結果：y = 4

- **if...else...敘述的巢狀應用語法**

```
if (test_expression1) {  
    statement 1  
} else if (test_expression2) {  
    statement 2  
} else if (test_expression3) {  
    statement 3  
} else {  
    statement 4  
}
```

- **if...else...敘述巢狀應用的程式範例（程式名稱：if\_test-03.R）**

```
x <- 0  
if (x < 0) {  
    print("負數")  
} else if (x > 0) {  
    print("正數")  
} else  
    print("零")
```

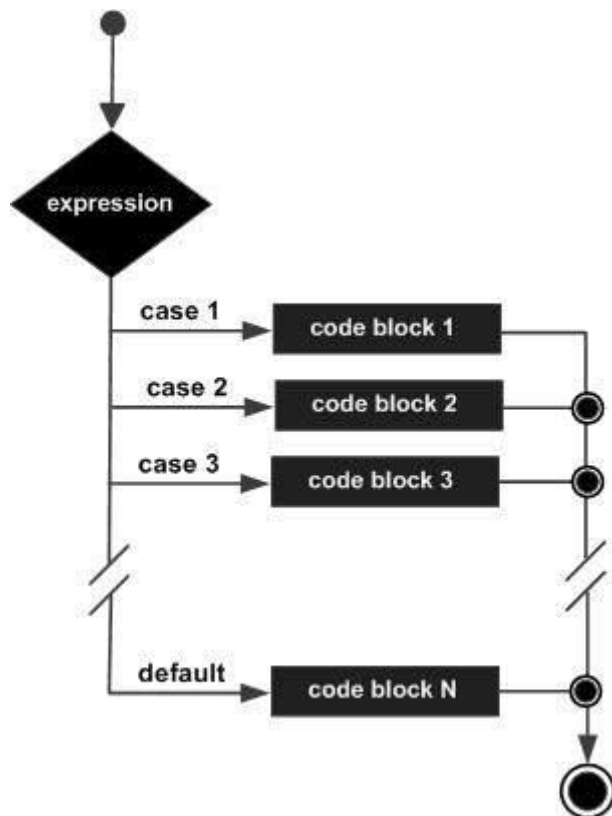
**\*執行結果：**

[1] "零"

## **(二)、條件控制轉換 switch()**

switch() 是一個函數，其使用方式為 switch(condition, expr 1, expr 2, expr 3)，對 switch() 而言，指令中的第一個引數 condition 是條件控制運算式，回傳整數或字串；令所有引數數目 = 所有 expr 加一(condition)，若回傳正整數 k，且回傳之正整數小於所有引數數目減一，則執行運算式 expr k，若回傳整數 k 大於所有引數數目或小於 1，則 switch() 回傳 NULL。例如：

- switch 敘述的語法  
`switch(expression, case1=value1, case2=value2, ..., caseN=valueN)`
- switch 敘述的流程圖



- switch 敘述的程式範例 (程式名稱: **switch\_test-01.R**)

```

X <- 1
switch(X, 5, sum(1:10), rnorm(5))

X <- 2
switch(X, 5, sum(1:10), rnorm(5))

X <- 3
switch(X, 5, sum(1:10), rnorm(5))

X <- 4
switch(X, 5, sum(1:10), rnorm(5))>

```

**\*執行結果：**

```

[1] 5
[1] 55

```

```
[1] 0.914774868 -0.002456267 0.136009552 -0.720153545 -0.1981243
30
```

```
Y <- 1
switch(Y, juice="蘋果", meat="豬肉")
Y <- "juice"
switch(Y, juice="蘋果", meat="豬肉")
```

**\*執行結果：**

```
[1] "蘋果"
[1] "蘋果"
```

### (三)、迴圈控制

在 R 中可以使用的迴圈控制方式主要有 repeat、while 與 for 這三種，雖然大部分的迴圈計算可以使用 R 的向量化運算來處理，不過在重複執行某區段的程式碼時，還是必須使用到這裡的迴圈控制結構。

- repeat 迴圈

repeat 迴圈是最簡單的迴圈控制結構，它會讓 R 不斷得執行指定的程式碼，直到遇到 break 中斷迴圈的執行為止：（程式名稱：repeat\_test-01.R）

```
x <- 0
repeat {
  message("x = ", x)
  x <- x + 1
  if (x == 5) break
}
```

當 R 在執行程式時若遇到 break 會直接跳出整個迴圈的執行，若只想要讓 R 跳過當次的反覆，執行下一次的重複動作，可以使用 next：（程式名稱：repeat\_test-02.R）

```
x <- 0
repeat {
  x <- x + 1
  if (x %% 2 == 0) next
  message("x = ", x)
  if (x > 7) break
}
```

- **while 迴圈**

while 迴圈類似 repeat，不過它會先檢查指定的條件，在條件為 TRUE 的情況下才會執行迴圈的內容。

while 迴圈的外觀架構：

```
while (某種條件) {  
  # 每次反覆要執行的程式  
}
```

- **while 敘述的程式範例（程式名稱：while\_test-01.R）**

```
# 計算 1+2+3+4...+135 的值是多少？  
i <- 1  
result <- 0  
  
# 當符合裡面的條件時，就會一直重複括號內的程式碼，直到不符合為止  
while(i <= 135) {  
  # 迴圈內重複進行的動作  
  result <- result + i  
  i <- i + 1  
}  
  
cat("1+2+3+4+5+.....+135 =", result)
```

```
> month.name #1~12 月的月份名稱
```

```
[1] "January" "February" "March"    "April"  
[5] "May"     "June"     "July"     "August"  
[9] "September" "October"  "November" "December"
```

```
> month.name[5]
```

```
[1] "May"
```

- **for 迴圈**

for 迴圈主要是用在程式執行前就已經知道反覆次數的情況，它在執行時會將輸入向量中的每個元素值逐一指定給反覆變數，然後重複執行迴圈的內容。

for 迴圈的外觀架構：



```
for (i in x) {  
  # 每次反覆要執行的程式  
}
```

- **for 敘述的程式範例 (程式名稱: for\_test-01.R)**

```
for (month in month.name) {  
  print(month)  
}
```

**\*執行結果：**

```
[1] "January"  
[1] "February"  
[1] "March"  
[1] "April"  
[1] "May"  
[1] "June"  
[1] "July"  
[1] "August"  
[1] "September"  
[1] "October"  
[1] "November"  
[1] "December"
```

---

```
for (i in 1:10) {  
  if (!i %% 2){  
    next  
  }  
  print(i)  
}
```

**\*執行結果：**

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

雖然 for 迴圈的寫法跟 R 的向量化運算類似，但實際上 R 向量化運算的效能跟 C 語言層級的迴圈相當，而 R 的 for 迴圈則不同，其執行效率上會差很多，所以在執行大量運算時，

應盡可能避免使用 for 迴圈，改以向量化運算的方式代替。

### ➤ **apply()**

apply 函數是最常用的代替 for 迴圈的函數。apply 函數可以對矩陣、資料框、陣列（二維、多維），按行或列進行迴圈計算，對子元素進行反覆運算，並把子元素以參數傳遞的形式給自訂的 FUN 函數中，並以返回計算結果。

函式定義：

`apply(X, MARGIN, FUN, ...)`

參數列表：

✧ X：陣列、矩陣、資料框。

✧ MARGIN：按行計算或按列計算，1 表示按列(row)，2 表示按行(column)。

✧ FUN：自訂的調用函數。

✧ ...：更多參數，可選。

```
> x<-matrix(1:12, ncol=3)
```

```
> x
```

```
      [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

```
> apply(x, 1, sum)
```

```
[1] 15 18 21 24
```

```
> apply(x, 2, sum)
```

```
[1] 10 26 42
```

### ➤ **lapply()**

lapply()（代表 list apply）與矩陣的 apply() 函數的用法類似，對列表的每個組件執行給定的函數，並返回另一個列表。

```
> x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
```

```
> lapply(x, mean)
```

```
$a
```

```
[1] 5.5
```

```
$beta
```

```
[1] 4.535125
```

```
$logic
```

```
[1] 0.5
```

➤ **sapply()**

sapply()（代表 simplified []apply）可以將結果整理以向量、矩陣、列表的形式輸出。

```
> sapply(x, mean)
```

```
      a      beta      logic
5.500000 4.535125 0.500000
```

```
> sapply(x, quantile) #每一個對應組件輸出 5 個元素，所以為 5 行，像矩陣一樣，豎着來的。
```

```
      a      beta      logic
0%    1.00  0.04978707  0.0
25%    3.25  0.25160736  0.0
50%    5.50  1.00000000  0.5
75%    7.75  5.05366896  1.0
100%  10.00 20.08553692  1.0
```

```
> sapply(2:4, seq)
```

```
[[1]]
[1] 1 2
```

```
[[2]]
[1] 1 2 3
```

```
[[3]]
[1] 1 2 3 4
```

➤ **vapply()**

vapply() 與 sapply() 相似，他可以預先指定的返回值類型。使得得到的結果更加安全。

```
> vapply(x, quantile, c(1,2,5,6,8)) #它需要一個 5 個長度的向量來告訴他返回的類型，向量裏面的內容可以變換
```

```
      a      beta      logic
0%    1.00  0.04978707  0.0
25%    3.25  0.25160736  0.0
50%    5.50  1.00000000  0.5
75%    7.75  5.05366896  1.0
```

```
100% 10.00 20.08553692 1.0
```

#### ➤ **tapply()**

tapply(x, f, g) 需要向量 x (x 不可以是數據框)，因子或因子列表 f 以及函數 g。

tapply() 執行的操作是：暫時將 x 分組，每組對應一個因子水平，得到 x 的子向量，然後這些子向量應用函數 g

```
> a <- c(24,25,36,37)
> b <- c('q', 'w', 'q', 'w')
> tapply(a, b, mean)
  q  w
30 31
```

### (四)、自訂函數

#### 自訂函數的外觀架構

一個自訂函數的外觀架構長得像這個樣子：

```
function_name <- function(輸入 1, 輸入 2, ..., 輸入 n, 參數 1, 參數 2, ..., 參數 n) {
  # 呼叫函數後執行的程式
  return(輸出)
}
```

#### ■ 計算數值的平方 (程式名稱：myFun-01.R)

```
# 程式名稱：myFun-01.R
# 自訂函數
my_squared <- function(x) {
  y <- x^2
  return(y)
}

# 呼叫函數
my_squared(2)
```

**\*執行結果：**

```
[1] 4
```

#### ■ 計算圓的面積 (程式名稱：myFun-02.R)

實作一個自訂函數叫做 **circle\_area()**，當使用者輸入一個圓形的半徑，它會計算出

該圓形的面積。

```
# 程式名稱：myFun-02.R
# 自訂函數
circle_area <- function(r) {
  area <- pi*r^2 #R 語言有內建圓周率 pi
  return(area)
}

# 呼叫函數
circle_area(3)
circle_area(5)
```

**\*執行結果：**

```
[1] 28.27433
```

```
[1] 78.53982
```

#### ■ 計算圓的周長（程式名稱：myFun-03.R）

實作一個自訂函數叫做 **circle\_circum()**，當使用者輸入一個圓形的半徑，它會計算出該圓形的周長。

```
# 程式名稱：myFun-03.R
# 自訂函數
circle_circum <- function(r) {
  circum <- 2*pi*r #R 語言有內建圓周率 pi
  return(circum)
}

# 呼叫函數
circle_circum(3)
circle_circum(5)
```

**\*執行結果：**

```
[1] 18.84956
```

```
[1] 31.41593
```

#### ■ 讓使用者決定要計算圓面積或周長（程式名稱：myFun-04.R）

把前面兩個自訂函數的功能整合起來到一個函數裡面，這個時候會正式納入參數（parameters）的觀念到這個自訂函數中，使用一個邏輯值參數 **is\_area** 來讓使用者呼叫

時決定要計算面積或者周長。

```
# 程式名稱：myFun-04.R
# 自訂函數
circle_calculator <- function(r, is_area) {
  area <- pi*r^2
  circum <- 2*pi*r
  if (is_area == TRUE) {
    return(area)
  } else {
    return(circum)
  }
}

# 呼叫函數
circle_calculator(3, is_area=TRUE)
circle_calculator(3, is_area=FALSE)
```

**\*執行結果：**

```
[1] 28.27433
```

```
[1] 18.84956
```

#### ■ 計算 BMI (程式名稱：myFun-05.R)

```
# 程式名稱：myFun-05.R
# 自訂函數
BMI <- function(height,weight) {
  bmivalue <- round(weight/(height/100)^2, 1)
  return(bmivalue)
}

h = c(185,162,122,224,154)
w = c(68,70,65,74,69)

people <- data.frame(h,w)
people$bmi <- BMI(people$h,people$w)

print(people)
```

**\*執行結果：**

```
[1] 28.27433
```

```
[1] 18.84956
```

## (五)、實務演練-統計學員 BMI

### 1. 計算 BMI 之 1

```
# 程式名稱：myBMI-02.R
```

```
weight <- sample(60:98, 10, replace = FALSE) #體重(公斤)
```

```
height <- round(rnorm(10, mean = 1.72, sd = 0.156), 2) #身高(公尺)
```

```
cat("* 資料筆數:", length(weight)) #計算資料總筆數
```

```
bmi <- round(weight/height^2, 1) # 新增變數 bmi
```

```
cat("BMI 資料一覽表:", bmi)
```

```
average <- mean(bmi)
```

```
cat("BMI 平均值:", round(average, 1))
```

```
if (average > 24) {  
  print("* BMI-均值過重!")  
} else if (average >= 18.5) {  
  print("* BMI-均值標準!")  
} else {  
  print("* BMI-均值太輕!")  
}
```

### 2. 計算 BMI 之 2

```
# 程式名稱：myBMI-03.R
```

```
#install.packages("dplyr") #安裝一次即可
```

```
#install.packages("rio") #安裝一次即可
```

```
library(dplyr)
```

```
library(rio)
```

```
name <- c("王克雄", "黃中木", "王芳元", "李麗瑕", "莊寅竣", "簡麗艷", "童益興", "  
莊淑寬", "鄭碧海", "林政雄")
```

```
gender <- c("M", "M", "F", "F", "M", "F", "M", "F", "M", "M")
```

```

height <- round(rnorm(10, mean = 172, sd = 12), 1) #身高(公分)
weight <- sample(50:98, 10, replace = FALSE) #體重(公斤)

myEmployee.df = data.frame(name, gender, height, weight)
myEmployee.df

avgH <- round(mean(myEmployee.df$height), 1)
avgW <- round(mean(myEmployee.df$weight), 1)

cat("* 資料筆數 :", length(weight)) #計算資料總筆數
cat("* 身高均值 :", avgH, "(公分)") #計算身高均值
cat("* 體重均值 :", avgW, "(公斤)") #計算體重均值

myBMI.df <- mutate(myEmployee.df, BMI = round((weight/(height/100)^
2), 1))
myBMI.df

#將資料輸出為 csv 檔案
export(myBMI.df, "myBMI.csv", format = "csv")

#查詢某位員工 BMI 資料
subset(myBMI.df, regexpr("莊寅竣", myBMI.df$name) > 0)

#簡單統計應用：排序
myBMI.df <- arrange(myBMI.df, gender, BMI)
myBMI.df

#簡單統計應用：統計男/女性員工資料
myBMI_M.df <- subset(myBMI.df, regexpr("M", myBMI.df$gender) > 0)
myBMI_F.df <- subset(myBMI.df, regexpr("F", myBMI.df$gender) > 0)
#簡單統計應用：顯示男性員工之欄位統計資訊
summary(myBMI_M.df)
#簡單統計應用：顯示女性員工之欄位統計資訊
summary(myBMI_F.df)

#顯示體重超過 90 公斤的員工資料
filter(myBMI.df, weight > 90)
#顯示體重超過 60 公斤且身高低於 170 公分的員工資料
filter(myBMI.df, weight > 60 & height < 170)

```



```
#顯示身高超過 170 公分的女性員工資料
filter(myBMI.df, height > 170 & gender == "F")
#顯示身高超過 170 公分或性別為女性的員工資料
filter(myBMI.df, height > 170 | gender == "F")
```

### 3. 計算年齡

```
# 程式名稱：myAge-01.R
#install.packages("lubridate") #安裝一次即可
library(lubridate)

calc_age <- function(birthDate, refDate = Sys.Date()) {
  #Sys.Date()指令可取得系統日期
  require(lubridate)
  period <- as.period(interval(birthDate, refDate), unit = "year")
  age <- period$year
  cat("Age =", age)
}

calc_age("2000/12/09")
```

### 4. 計算 BMI 之 3

```
# 程式名稱：myBMI-04.R
#install.packages("lubridate") #安裝一次即可
library(lubridate)

calc_age <- function(birthDate, refDate = Sys.Date()) {
  require(lubridate)
  period <- as.period(interval(birthDate, refDate), unit = "year")
  age <- period$year
}

calc_ageGroup <- function(age) {
  sapply(age, function(x) if(x >= 66) "老年"
        else if (x >= 41) "中年"
        else if (x >= 18) "青年")
}
```

```

    else "少年")
}

name <- c("王克雄","黃中木","王芳元","李麗瑕","莊寅竣","簡麗艷",
        "童益興","莊淑寬","鄭碧海","林政雄")
gender <- c("M","M","F","F","M","F","M","F","M","M")
birthday <- c("1966/01/05","1976/03/29","1972/07/26",
             "1980/03/08","1982/10/27","1977/09/28",
             "2001/10/10","1983/10/25","1979/04/04",
             "1996/11/09")
age <- calc_age(birthday)
ageGroup <- calc_ageGroup(age)
height <- round(rnorm(10, mean = 172, sd = 12), 1) #身高(公分)
weight <- sample(50:98, 10, replace = FALSE) #體重(公斤)

myEmployee.df = data.frame(name, birthday, age, ageGroup, gender, height, weight)
myEmployee.df
summary(myEmployee.df)

```