第 15 章: 資料處理

15: Data Management

統計分析的第一步就是資料清理與資料處理, 通常可佔據整體統計分析的 70%-80% 時間. 資料清理包含同一個體重覆輸入觀測值, 錯誤輸入觀測值, 缺失值檢視與處理, 連續變數語類別變數的區分, 選取適當的個體, 資料排序與分組等等, R 有許多資料處理的函式與套件, 但是 R 資料儲存有許多不同類型與格式, 包含 向量, 矩陣, 陣列, 列表等, 且 R 個別函式與套件, 統計模型接受的資料類型與格式各不相同, 因此讓使用者容易混淆而造成困擾. R 提供 `apply()` 系列函式, 功能強大, 但因接受的資料類型與格式各不相同, 讓初學者相當困惑, 參見 表 15.1. 套件 `plyr`, `dplyr`, `reshape`, `reshape2`, `tidyr`, `tidyverse` 等, 提供較 R 容易理解的資料處理函式, 但是計算速度有差異, 且經常更新與改名, 也讓初學者相當困惑.

表 15.1: R Basic `apply()` Function

	Output Object			
Input object	array	data frame	list	nothing
array	apply	.	.	.
data frame	.	aggregate	by	.
list	sapply	.	lapply	.
replicates	replicate	.	replicate	.
function	arguments	mapply	mapply	.

資料物件通常以 矩陣, 陣列, 列表 與 資料框架 型式儲純數值, 統計資料物件內個別維度的成分 (變數) 通常是向量物件, 在統計分析資料時, 必須對資料物件內各別的成分 (變數) 進行運算, 在 R 中, 許多函式通常利用向量做運算, 但在統計分析資料時, 更多時候必須對許多向量使用同一函式做重複運算, 例如, 資料框架內的每一變數, 欄向量 (行向量), 計算平均值與變異數. 在統計分析資料時, 有時候會針對一個變數, 依照另外一個類別變數不同的類別水準, 分別使用同一函式做運算, 例如. 根據不同治療的組別, 分別計算血壓的樣本平均值與樣本變異數. 在上述的情形, 許多程式語言常常須用到迴圈 (loop), 或是自行寫作向量操作函式. 在 R 中, 迴圈演算法的執行非常無效率, 應儘量避免, 因此, 在 R 中許多函式直接利用向量做運算, 且在 R 中有些函式, 例如, `apply()`, `lapply()`, `sapply()`, `tapply()`, `mapply()`, `Vectorize()`, `by()`, `sweep()`, `aggregate()`, `replicate()` 等, 可以對不同向量進行重複的運算, 如此避免以迴圈演算法進行重複的運算, 可以更有效率的執行類似迴圈的函式指令.

表 15.2: 常見資料物件操作函式

函式	輸入	輸出
`by`	df	list
`aggregate`	df	df
`replicate`	r	array, list
`apply`	matrix, array, df	vector, array, list
`dendrapply`	dendogram dendogram	
`eapply`	environment list	
`lapply`	list, df, vector	list
`mapply`	list, df, vectors	vector, matrix, list
`rapply`	list	vector, list,
`sapply`	list, df, vector	vector, matrix, list
`tapply`	df, categories	array, list
`vapply`	list, df, vector	vector, matrix, list

15.1 資料物件操作函式: apply() 函式

使用函式 apply() 對一個 資料框架 (data frame), 矩陣 (matrix) 陣列物件 (array) 內之任一維度的邊際 (margins of an array) 執行同一個函式運算, 並回傳一向量, 陣列或列表. 函式指令爲

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

其中的主要引數分別爲:

- X 爲所要執行之陣列物件, 若資料物件 X 爲 矩陣 或 資料框架, 則函式 apply() 會將該 矩陣 或 資料框架 轉換成 2-維度 陣列 (array).
- MARGIN 爲陣列物件 X 的邊際維度序號.
  - MARGIN = 1 會對陣列 X 的第 1 個維度 (row) 執行函式, 例如, 對 矩陣 或 資料框架 的每一列 (row) 做運算.
  - MARGIN = 2 會對陣列 X 的第 2 個維度 (column) 執行函式, 例如, 對 矩陣 或 資料框架 的每一欄 (行, column) 做運算等等.
  - MARGIN = k 會對陣列 X 的第 k 個維度 執行函式.
- FUN 爲執行同一函式的名稱.

例 **15.1.** 安眠藥臨床試驗

醫學中心一位醫師針對一新開發之 安眠藥 T 的藥效進行一研究, 這位醫師想證明 安眠藥 T 比 安慰劑 C 可以有效地治療失眠患者. 因此這位醫師進行一個先導研究, 共 10 位失眠患者同意參予研究. 這 10 位受試者, 在第 1 天給予 安慰劑 C, 並測量第 1 天當夜的連續睡眠時間. 然後第 2 天給予安眠藥 T, 再次測量第 2 天當夜的連續睡眠時間. 研究的主要反應變數爲受試者的連續睡眠的時間, 若其第 2 天睡眠的時間遠大於第 1 天連續睡眠的時間, 表示新開發之安眠藥 T 可以有效地治療失眠患者.

表 15.3 顯示研究結果 (單位: 小時). 研究的結果是否有足夠的證據力支持研究者的想法?

表 15.3: 安眠藥的藥效臨床試驗研究: 服藥後的睡眠時間

受試者	1	2	3	4	5	6	7	8	9	10
C	2.0	3.6	4.0	3.9	4.0	1.3	3.5	3.9	3.8	1.5
T	5.0	4.8	2.5	6.2	3.6	7.3	4.9	2.6	3.7	7.4

```
> ## apply() + matrix
> x.mat <- matrix(1:12, nrow = 4, byrow = T)
> x.mat
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> apply(X = x.mat, MARGIN = 1, FUN = mean)
[1]  2  5  8 11
> apply(x.mat, 2, mean)
[1] 5.5 6.5 7.5
> #
> ## apply() + data.frame
> setwd("C:/RData/")
> sleep <- read.table(file = "sleepwidePaired.csv",
                              header = TRUE, sep = ",", dec = ".",
                              row.names = NULL)
> head(sleep)
  id Tdrug Cdrug
1  1   5.0   2.0
2  2   4.8   3.6
3  3   2.5   4.0
4  4   6.2   3.9
5  5   3.6   4.0
6  6   7.3   1.3
> apply(sleep[, 2:3], 1, mean)
 [1] 3.50 4.20 3.25 5.05 3.80 4.30 4.20 3.25 3.75 4.45
> apply(sleep[, 2:3], 2, mean)
Tdrug Cdrug
 4.80  3.15
 > #
> ## more apply() + matrix
> set.seed(100)
> x.mat <- matrix(rnorm(1000), 200, 5)
> apply(x.mat, 2, quantile, probs = c(0.25, 0.75))
          [,1]       [,2]       [,3]       [,4]       [,5]
```

```
25% -0.5056721 -0.6637388 -0.6670268 -0.7151312 -0.6835946
75%  0.5273765  0.6685950  0.6429354  0.7748576  0.8415970
>
> y.mat <- matrix(rnorm(10000), 2000, 5)
> apply(y.mat, 2, quantile, probs = c(0.25, 0.75))
          [,1]       [,2]       [,3]       [,4]       [,5]
25% -0.7058193 -0.6644742 -0.6559814 -0.6828271 -0.6520564
75%  0.6618302  0.6722488  0.6733151  0.6641776  0.6773927
> #
> z.mat <- matrix(rnorm(10000), 5, 2000)
> apply(z.mat, 1, quantile, probs = c(0.25, 0.75))
          [,1]       [,2]       [,3]       [,4]       [,5]
25% -0.6717053 -0.6963323 -0.6643946 -0.7375793 -0.6294552
75%  0.6154957  0.6828310  0.6504301  0.6262656  0.7088369
>
>
> ## apply() + array
> b.arr <- array(1:24, dim = c(4, 3, 2),
                 dimnames = list(c("x1", "x2", "x3", "x4"),
                                 c("y1", "y2", "y3"),
                                 c("z1", "z2")))
> b.arr
, , z1
   y1 y2 y3
x1  1  5  9
x2  2  6 10
x3  3  7 11
x4  4  8 12

, , z2
   y1 y2 y3
x1 13 17 21
x2 14 18 22
x3 15 19 23
x4 16 20 24

> #
> b.arr[1, , ]
   z1 z2
y1  1 13
y2  5 17
y3  9 21
> mean(b.arr[1, , ])
[1] 11
> #
> apply(b.arr, 1, mean) # average by x
x1 x2 x3 x4
11 12 13 14
> rowMeans(b.arr, dims = 1)
x1 x2 x3 x4
```

```
11 12 13 14
> #
> apply(b.arr, c(2, 3), mean) # average over x
     z1   z2
y1  2.5 14.5
y2  6.5 18.5
y3 10.5 22.5
> colMeans(b.arr, dims = 1)
     z1   z2
y1  2.5 14.5
y2  6.5 18.5
y3 10.5 22.5
> #
> apply(b.arr, c(1, 2), mean)
   y1 y2 y3
x1  7 11 15
x2  8 12 16
x3  9 13 17
x4 10 14 18
> rowMeans(b.arr, dims = 2)
   y1 y2 y3
x1  7 11 15
x2  8 12 16
x3  9 13 17
x4 10 14 18
> #
> # array average
> a.arr <- array(rnorm(2*2*10), c(2, 2, 10))
> apply(a.arr, c(1, 2), mean)
           [,1]        [,2]
[1,] 0.2426878 -0.41321519
[2,] 0.1213732  0.09303838
> rowMeans(a.arr, dims = 2)
           [,1]        [,2]
[1,] 0.2426878 -0.41321519
[2,] 0.1213732  0.09303838
> #
> apply(a.arr, c(1, 3), mean)
           [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
[1,] 0.4570282 -0.4108178 -0.5218928 -0.4031859 -0.1281254 -0.2967553
[2,] 0.3604596  0.1664159  0.2622479  0.2454029 -0.4436532  0.7604331
            [,7]       [,8]      [,9]     [,10]
[1,] -0.1859059 -0.5653128 0.5192811 0.6830499
[2,] -1.0262100 -0.5717037 1.1088832 0.2097824
> rowMeans(a.arr, dims = 1)
[1] -0.08526367  0.10720581
> colMeans(a.arr, dims = 1)
           [,1]       [,2]       [,3]       [,4]       [,5]      [,6]
[1,] 0.2399494  0.1379897  0.7800541  0.1544253 -0.3909275 0.2618471
[2,] 0.5775384 -0.3823917 -1.0396990 -0.3122082 -0.1808511 0.2018306
```

```
          [,7]       [,8]      [,9]     [,10]
[1,] -0.1266432 -0.4104021 0.4224371 0.7515755
[2,] -1.0854728 -0.7266143 1.2057272 0.1412568
> #
> colMeans(a.arr, dims = 2)
 [1]  0.40874388 -0.12220098 -0.12982247 -0.07889147 -0.28588930  0.23183887
 [7] -0.60605798 -0.56850821  0.81408215  0.44641616
```

15.2 資料物件操作函式: lapply() 與 sapply()

使用函式 `lapply()` 對一個列表物件 (list) 內的每一成分 (component), 執行同一個函式做運算, 並回傳一個列表物件 (list), 且此回傳的列表物件長度與原有的列表物件長度相同.

函式 `sapply()` 是相對於函式 `lapply()` 較容易使用的替代函式, 使用函式 `sapply()` 對一個列表物件 (list) 內的每一成分 (component), 執行同一個函式運算, 並且回傳一 向量 或 矩陣. 例如, 使用函式指令 `sapply(X.df)` 可以對資料框架 (列表物件) `X.df` 執行同一個函式做運算, 是指對每一個變數 (欄位), 做運算. 函式 `vapply()` 是類似於函式 `sapply()`. 函式指令爲

```
> lapply(X, FUN, ...)
> sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
> vapply(X, FUN, FUN.VALUE, ..., USE.NAMES = TRUE)
```

其中的主要引數分別爲:

- `X` 爲所要執行之 列表物件 或 資料框架.
- 對列表每一個成分執行同一函式.
- `FUN` 爲執行同一函式的名稱.
- `FUN.VALUE` 爲 執行函數 `FUN` 回傳的物件名稱.
- `USE.NAMES = TRUE`: 若 `X` 物件成分已有命名, 內設直接使用 `X` 命名.

```
> ## lapply(), sapply() + list
> set.seed(100)
> x.list <- list(a = 1:5, b = rnorm(10, mean = 0, sd = 1))
> lapply(x.list, mean)
$a
[1] 3

$b
[1] -0.01795716
> #
> sapply(x.list, mean)
          a           b
 3.00000000 -0.01795716
> #
> x.list <- list(a = 11:15, b = rnorm(10), c = rnorm(1000, mean = 5, sd =  2))
> lapply(x.list, mean)
$a
[1] 13

$b
[1] 0.2336915

$c
[1] 5.033755
> #
> sapply(x.list, mean)
         a          b          c
13.0000000  0.2336915  5.0337552
> #
> x <- 1:2
> lapply(x, runif)
[[1]]
[1] 0.5156047

[[2]]
[1] 0.4649941 0.5249516
> #
> sapply(x, runif)
[[1]]
[1] 0.1502433

[[2]]
[1] 0.7110009 0.3702982
> #
> x <- 1:2
> lapply(x, runif, min = 0, max = 10)
[[1]]
[1] 4.533344

[[2]]
```

```
[1] 6.179135 2.836679
> #
> sapply(x, runif, min = 0, max = 10)
[[1]]
[1] 2.069829

[[2]]
[1] 6.077585 6.662716
> #
> x.list <- list(a = matrix(1:4, 2, 2), b = matrix(1:6, 3, 2))
> x.list
$a
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$b
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> #
> lapply(x.list, function(elt) elt[,1])
$a
[1] 1 2

$b
[1] 1 2 3
> #
> sapply(x.list, function(elt) elt[,1])
$a
[1] 1 2

$b
[1] 1 2 3
> #
> c.list <- list(a = 1:20,
                 beta = exp(-2:2),
                 logic = c(TRUE, FALSE, FALSE, TRUE, FALSE))
> lapply(c.list, mean)
$a
[1] 10.5

$beta
[1] 2.322111

$logic
[1] 0.4
> #
> sapply(c.list, mean)
```

```
        a      beta     logic
10.500000  2.322111  0.400000
> #
> # sapply()+ list = simplify
> sapply(c.list, mean, simplify = FALSE)
$a
[1] 10.5

$beta
[1] 2.322111

$logic
[1] 0.4
> #
> sapply(c.list, mean, simplify = TRUE)
        a      beta     logic
10.500000  2.322111  0.400000
> sapply(c.list, mean)
        a      beta     logic
10.500000  2.322111  0.400000
> #
> ## lapply() + data.frame
> setwd("C:/RData/")
> sleep <- read.table(file = "sleepwidePaired.csv",
                      header = TRUE, sep = ",", dec = ".",
                      row.names = NULL)
> #
> lapply(sleep, mean)
$id
[1] 5.5

$Tdrug
[1] 4.8

$Cdrug
[1] 3.15
> #
> sapply(sleep, mean)
   id Tdrug Cdrug
 5.50  4.80  3.15
> #
> ## vapply() + list + FUN
> d.list <- sapply(3:5, seq) # list of vectors
> d.list
[[1]]
[1] 1 2 3

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

```

```
[[3]]
[1] 1 2 3 4 5
> #
> sapply(d.list, fivenum)
     [,1] [,2] [,3]
[1,]  1.0  1.0    1
[2,]  1.5  1.5    2
[3,]  2.0  2.5    3
[4,]  2.5  3.5    4
[5,]  3.0  4.0    5
> #
> vapply(d.list, fivenum,
         c(Min = 0, "Q1" = 0,
           Median = 0, "3rd Qu." = 0, "Max" = 0))
        [,1] [,2] [,3]
Min      1.0  1.0    1
Q1       1.5  1.5    2
Median   2.0  2.5    3
3rd Qu.  2.5  3.5    4
Max      3.0  4.0    5
```

15.3 資料物件操作函式: tapply()

使用函式 `tapply()` 可以對一參差不齊的陣列 (ragged array), 依照類別變數的 類別水準 或 組別 的指標值, 分別使用同一函式做運算. 例如, 計算分組平均值. 函式指令爲

```
> tapply(X, INDEX, FUN = NULL, ..., simplify = TRUE)
```

其中的主要引數分別爲:

- `X` 是一個原型物件 (atomic object), 通常向量.
- `INDEX` 通常是一個 類別變數, 包含 類別水準 或 組別 所形的 因子向量 (或 陣列), 長度與 `X` 相同.

例 **15.2.** 抽菸, 膽固醇與血壓調查研究

一個早期大規模的抽樣調查, 以問卷方式調查 抽菸, 膽固醇 與 血壓 的關聯性研究, 隨機抽取了近 1.5 萬位成年人進行問卷訪問, 資料變數在表 **??**, 部分個體資料在

表 **??**, 檔案在 BPsurvey.csv. 研究主要描述一般族群的健康狀況, 探討 疾病 與 發生原因 之關聯性, 例如, 分析 抽菸, 膽固醇 與 血壓 的關聯性.

```
> ## tapply() + data.frame
> setwd("C:/RData/")
> bp.df <- read.table(file = "BPsurvey.csv",
                      header = TRUE, sep = ",", dec = ".",
                      row.names = NULL)
> dim(bp.df)
[1] 15643    10
> head(bp.df)
  id sex race smoke age wt  ht sbp dbp chol
1  3   1    1     1  21 81 179 120  67  268
2  4   0    1     1  32 61 162 126  86  160
3  9   0    1     2  48 68 157 131  73  236
4 10   1    1     1  35 92 177 130  82  225
5 11   1    1     2  48 70 168 120  70  260
6 19   1    2     3  44 86 178 133  85  187
>#
> tapply(bp.df$wt, bp.df$sex, mean)
       0        1
70.56511 80.18642
> tapply(bp.df$wt, bp.df$sex, mean, simplify = FALSE)
$`0`
[1] 70.56511

$`1`
[1] 80.18642
> #
> # sapply() by pass arguments into tapply()
> sapply(bp.df[, 5:10], FUN = tapply, bp.df$sex, mean)
       age       wt       ht      sbp      dbp     chol
0 47.64641 70.56511 160.1776 123.7655 72.19050 207.8440
1 48.44739 80.18642 173.3462 128.4596 76.79267 203.6927
```

15.4 資料物件操作函式: mapply()

函式 `mapply()` 是函式 `apply()` 的多變量型式, 對一個列表物件 (list) 內的每一成分 (component), 執行同一個函式運算, `mapply()` 回傳一向量, 矩陣 或 列表. 函式 `mapply()` 是函式 `Map` 的簡化. 函式 `mapply()`指令爲

```
> mapply(FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE,
         USE.NAMES = TRUE)
```

其中的主要引數分別爲:

- `FUN()` 爲執行同一函式的名稱, 且 `FUN()` 本身可以含有引數. `FUN()` 的引數以 `FUN(ARG1 = arg1, ARG2 = arg2, ...)` 等放入原有函式 `mapply()` 之中.
- 執行 `FUN` 函式的運算時, 使用 `ARG1 = arg1` 的第 1 元素, `ARG2 = arg2` 的第 2 元素, $\cdots$, 等等, 作爲 `FUN` 的引數.
- `MoreArgs = list()`, 爲執行 `FUN` 函式所需要用列表物件, 且引數可以循環使用.

```
> ## mapply()
> list(rep(1, 3), rep(2, 2), rep(3, 1))
[[1]]
[1] 1 1 1

[[2]]
[1] 2 2

[[3]]
[1] 3
> #
> mapply(rep, 1:3, 3:1)
[[1]]
[1] 1 1 1

[[2]]
[1] 2 2

[[3]]
[1] 3
> #
> mapply(rep, times = 1:3, x = 3:1)
[[1]]
[1] 3

[[2]]
[1] 2 2

[[3]]
[1] 1 1 1
> #
> mapply(rep, times = 1:3, MoreArgs = list(x = 7))
[[1]]
```

```
[1] 7

[[2]]
[1] 7 7

[[3]]
[1] 7 7 7
> ##
> noise <- function(n, mean, sd) {
    rnorm(n, mean, sd)
  }
> set.seed(100)
> noise(n = 3, mean = 1, sd = 2)
[1] -0.004384701  1.263062331  0.842165820
> noise(n = 1:3, mean = 1:3, sd = 2)
[1] 2.773570 2.233943 3.637260
> mapply(noise, n = 1:3, mean = 1:3, sd = 2)
[[1]]
[1] -0.1635814

[[2]]
[1] 3.4290654 0.3494811

[[3]]
[1] 2.280276 3.179772 3.192549
> #
> list(noise(1, 1, 2), noise(2, 2, 2),
+      noise(3, 3, 2), noise(4, 4, 2),
+      noise(5, 5, 2))
[[1]]
[1] 0.5967321

[[2]]
[1] 3.479681 2.246759

[[3]]
[1] 2.941367 2.222292 4.021713

[[4]]
[1] 2.172372 8.620594 3.123820 5.528121

[[5]]
[1] 5.523923 6.546809 3.371242 4.123099 3.559557
```

15.5 資料物件操作函式: Vectorize()

函式 Vectorize() 是 mapply() 的的包裝封套, 回傳一新函式, 有如呼叫 mapply() 函式. 函式指令爲

```
> Vectorize(FUN, vectorize.args = arg.names, SIMPLIFY = TRUE,
            USE.NAMES = TRUE)
```

其中的主要引數分別爲:

- FUN() 爲執行同一函式的名稱, 且 FUN() 本身可以含有引數. FUN() 必須是正式存在的函式名.
- vectorize.args 爲文字向量, 須將 FUN() 引數轉換爲文字向量.

```
> ## Vectorize()
> mapply(rep, 1:3, 3:1)
[[1]]
[1] 1 1 1

[[2]]
[1] 2 2

[[3]]
[1] 3
> #
> vrep <- Vectorize(rep)
> vrep(1:3, 3:1)
[1] 1 1 1 2 2 3
> #
> mapply(rep, times = 1:3, x = 3:1)
[[1]]
[1] 3

[[2]]
[1] 2 2

[[3]]
[1] 1 1 1
> #
> vrep(times = 1:3, x = 3:1)
[1] 3 2 2 1 1 1
> #
> vrep <- Vectorize(rep.int)
> vrep(times = 1:3, x = 45)
```

```
[[1]]
[1] 45

[[2]]
[1] 45 45

[[3]]
[1] 45 45 45
> #
> vrep <- Vectorize(rep.int, "times")
> vrep(times = 1:3, x = 45)
[[1]]
[1] 45

[[2]]
[1] 45 45

[[3]]
[1] 45 45 45
> #
> # error
> # not a formal function name
> vrep <- Vectorize(rep, "times")
Error in Vectorize(rep, "times") :
  must specify names of formal arguments for 'vectorize'
> help(rep) # rep.int as a formal function mane
```

15.6 資料物件操作函式: by()

使用函式 by() 可將 資料框架 依不同的組別因子, 計算統計量或執行同一個函式做運算. 函式指令爲

```
> by(data, INDICES, FUN, ..., simplify = TRUE)
```

其中的主要引數分別爲:

- data 爲 資料框架 或 矩陣.
- INDICES 爲因子向量代表類別變數的組別, 其長度與資料框架變數的長度 (nrows(data)) 相同.

- FUN() 是指所要計算的統計量或執行同一函式的名稱, FUN() 的引數必須是資料框架 或 矩陣 進行操作的函式.
- by() 回傳列表物件 (list).

```
> ## by()
> setwd("C:/RData/")
> bp.df <- read.table(file = "BPsurvey.csv",
                      header = TRUE, sep = ",", dec = ".",
                      row.names = NULL)
> head(bp.df)
  id sex race smoke age wt  ht sbp dbp chol
1  3   1    1     1  21 81 179 120  67  268
2  4   0    1     1  32 61 162 126  86  160
3  9   0    1     2  48 68 157 131  73  236
4 10   1    1     1  35 92 177 130  82  225
5 11   1    1     2  48 70 168 120  70  260
6 19   1    2     3  44 86 178 133  85  187
> #
> by(data = bp.df[, c(8:10)], INDICES = bp.df$sex, FUN = summary)
bp.df$sex: 0
      sbp             dbp              chol
 Min.   : 69.0   Min.   : 20.00   Min.   : 74.0
 1st Qu.:108.0   1st Qu.: 65.00   1st Qu.:175.0
 Median :119.0   Median : 72.00   Median :203.0
 Mean   :123.8   Mean   : 72.19   Mean   :207.8
 3rd Qu.:136.0   3rd Qu.: 78.00   3rd Qu.:235.0
 Max.   :237.0   Max.   :142.00   Max.   :676.0
------------------------------------------------------------
bp.df$sex: 1
      sbp             dbp              chol
 Min.   : 80.0   Min.   : 40.00   Min.   : 59.0
 1st Qu.:116.0   1st Qu.: 70.00   1st Qu.:174.0
 Median :125.0   Median : 76.00   Median :201.0
 Mean   :128.5   Mean   : 76.79   Mean   :203.7
 3rd Qu.:137.0   3rd Qu.: 83.00   3rd Qu.:229.0
 Max.   :244.0   Max.   :134.00   Max.   :702.0
> # by(bp.df[, c(5:10)], bp.df$sex, summary)
> #
> ##
> tmp <- by(bp.df, bp.df[, "sex"],
            function(x) lm(sbp ~ age, data = x))
> sapply(tmp, coef)
                     0           1
(Intercept) 88.5392872 105.5821478
age          0.7393261   0.4722124
> #
> # NOT WORK ANY MORE
> # BECAUSE MEAN FUN DID NOT TAKE df OBJECT
```

```
> by(data = bp.df[, c(8:10)], bp.df$sex, mean)
bp.df$sex: 0
[1] NA
------------------------------------------------------------
bp.df$sex: 1
[1] NA
Warning messages:
.............................
```

15.7 資料物件操作函式: aggregate()

函式 aggregate() 可將 資料框架 分成不同的組別, 分別計算統計量或執行同一個函式做運算. 函式 aggregate() 與函式 by() 有類似功能. 函式 aggregate() 指令爲

```
> ## S3 method for class 'data.frame'
> aggregate(x, by, FUN, ..., simplify = TRUE, drop = TRUE)
> ## S3 method for class 'formula'
> aggregate(formula, data, FUN, ...,
            subset, na.action = na.omit)
```

其中的主要引數分別爲:

- x 爲資料框架.
- formula 爲統計模型.
- by = f.list, 是 列表物件 (list), 長度與 資料框架 X.df 相同的 因子變數. 可用來分組分別計算統計量.
- FUN 是指所要計算的統計量或執行同一個函式名稱.
- aggregate() 中 FUN() 的限制較 by() 少, 但 aggregate() 回傳 資料框架 (data.frame) 與 by() 回傳列表 (list) 不同.

```
> ## aggregate()
> aggregate(bp.df[, 5:10], by = list(SEX = bp.df$sex), mean)
  SEX      age       wt       ht      sbp      dbp     chol
1   0 47.64641 70.56511 160.1776 123.7655 72.19050 207.8440
```

```
2   1 48.44739 80.18642 173.3462 128.4596 76.79267 203.6927
> class(aggregate(bp.df[, 5:10], by = list(SEX = bp.df$sex), mean))
[1] "data.frame"
> #
> # transpose a data.frame
> t(aggregate(bp.df[, 9:10], by = list(SEX = bp.df$sex), summary))
                [,1]   [,2]
SEX             0.00   1.00
dbp.Min.       20.00  40.00
dbp.1st Qu.    65.00  70.00
dbp.Median     72.00  76.00
dbp.Mean       72.19  76.79
dbp.3rd Qu.    78.00  83.00
dbp.Max.      142.00 134.00
chol.Min.      74.00  59.00
chol.1st Qu. 175.00 174.00
chol.Median  203.00 201.00
chol.Mean    207.80 203.70
chol.3rd Qu. 235.00 229.00
chol.Max.    676.00 702.00
> #
> ## compare aggregate() and sapply()
> aggregate(bp.df[, 8:10], by = list(SEX = bp.df$sex), mean)
  SEX      sbp      dbp     chol
1   0 123.7655 72.19050 207.8440
2   1 128.4596 76.79267 203.6927
> sapply(bp.df[, c(8:10)], tapply, bp.df$sex, mean)
       sbp      dbp     chol
0 123.7655 72.19050 207.8440
1 128.4596 76.79267 203.6927
```

15.8 資料物件操作函式: sweep()

使用函式 `sweep()` 可以對一個 陣列 物件, 針對邊際維度, 計算一個統計量或執行同一個函式做運算. 函式指令爲

```
> sweep(x, MARGIN, STATS, FUN = "-", check.margin = TRUE, ...)
```

其中的主要引數分別爲:

- `X` 爲一個 陣列 或 矩陣 物件.
- `FUN` 是指所要計算的統計量或執行同一函式的名稱.

- 函式 sweep() 自動內設爲 (FUN = "-") (減法運算), 若執行特定符號運算, 例如, 此特定符號運算爲 + (加號), 則符號須加用雙引號成爲 "+".

```
> ## sweep()
> b.data <- as.data.frame(matrix(c(1:24), nrow = 6, byrow = T))
> (b.mean <- apply(b.data, 2, mean))
V1 V2 V3 V4
11 12 13 14
> sweep(b.data, 2, b.mean)
   V1  V2  V3  V4
1 -10 -10 -10 -10
2  -6  -6  -6  -6
3  -2  -2  -2  -2
4   2   2   2   2
5   6   6   6   6
6  10  10  10  10
> #
> (b.mean <- apply(b.data, 1, mean))
[1]  2.5  6.5 10.5 14.5 18.5 22.5
> sweep(b.data, 1, b.mean)
    V1   V2  V3  V4
1 -1.5 -0.5 0.5 1.5
2 -1.5 -0.5 0.5 1.5
3 -1.5 -0.5 0.5 1.5
4 -1.5 -0.5 0.5 1.5
5 -1.5 -0.5 0.5 1.5
6 -1.5 -0.5 0.5 1.5
```

15.9 資料物件操作函式: replicate()

使用函式 replicate() 與函式 sapply() 類似, 爲函式 sapply() 的包裝封套, 可以重覆執行同一函式做運算. 函式指令爲

```
> replicate(n, expr, simplify = "array")
```

其中的主要引數

- n 所要重覆執行的次數.
- expr 爲所要重覆執行的同一運算式.
- 函式 replicate() 通常是執行產生隨機變數.

```
> ## replicate random variables from exponential distribution
> replicate(5, rexp(10))
            [,1]       [,2]      [,3]       [,4]         [,5]
 [1,] 0.06436724 1.49382223 0.1169804 0.39905662 0.0312094139
 [2,] 4.05790204 0.08366908 2.2931382 0.81557964 0.7972878694
 [3,] 0.52927286 3.49740175 0.2327129 0.75613704 0.4220017293
 [4,] 0.08287350 0.35182542 1.3666231 0.74424509 1.1743398280
 [5,] 0.14477947 0.53206901 1.1084216 0.33239908 0.8897708109
 [6,] 1.57175982 1.86628848 4.9939808 0.38849803 2.0421579144
 [7,] 0.02001488 0.16144714 1.5584840 1.37843723 0.2155170352
 [8,] 0.52199820 1.34512130 0.4799255 0.47659392 0.3325432939
 [9,] 0.08017182 0.55292033 2.6603417 1.65020506 1.9962155669
[10,] 0.23903690 1.18049660 1.1706547 0.04541023 0.0007613422
> replicate(5, mean(rexp(10)))
[1] 1.0176443 0.7254602 1.0603147 1.0598029 1.4094343
> #
> hist(replicate(100, mean(rexp(10))))
```

15.10 資料框架分割與合併

許多時候需要對資料框架分割 (split) 與合併 (merge), 可以分別使用 函式 `split()` 與 函式 `merge()` 進行分割與合併.

15.10.1 資料框架分割函式: split()

使用函式 `split()` 可以對資料框架分割, 傳回列表物件, 使用函式指令如下:

```
> split(x, f, drop = FALSE, ...)
> split(x, f, drop = FALSE, ...) <- value
> unsplit(value, f, drop = FALSE)
```

其中的主要引數分別爲:

- `x` 爲 1 個所要分割之資料框架.
- `f` 爲 1 個類別變數, 因子物件 或 因素物件 (factor), 或包含因子物件的交互作用項. 可以使用用 `as.factor(f)` 將變數 `f` 轉換成因子變數.

- `drop = FALSE` 爲 1 個邏輯指令, 自動內設爲 `FALSE`, 是否要排除那些未在 `f` 因子之水準內的資料.
- `value` 爲 1 個分割後之資料框架或列表物件向量.

例 **15.3.** R 內建資料 Puromycin

R 內建資料 `data(Puromycin)`, 比較 Puromycin 與 Placebo (`state` = "treated" 或 "untreated") 在不同 substrate 濃度 (`conc`) 下的酵素化學反應速率.

```
> ## split()
> data(Puromycin)
> head(Puromycin)
conc rate   state
1 0.02   76 treated
2 0.02   47 treated
3 0.06   97 treated
4 0.06  107 treated
5 0.11  123 treated
6 0.11  139 treated
> #
> (df.split <- split(Puromycin, Puromycin$state))
$treated
   conc rate   state
1  0.02   76 treated
2  0.02   47 treated
3  0.06   97 treated
.....................

$untreated
   conc rate     state
13 0.02   67 untreated
14 0.02   51 untreated
15 0.06   84 untreated
.....................
> #
> lapply(df.split, summary)
$treated
      conc            rate              state
 Min.   :0.020   Min.   : 47.0   treated  :12
 1st Qu.:0.060   1st Qu.:104.5   untreated: 0
 Median :0.165   Median :145.5
 Mean   :0.345   Mean   :141.6
 3rd Qu.:0.560   3rd Qu.:193.2
 Max.   :1.100   Max.   :207.0

$untreated
```

```
      conc             rate             state
 Min.   :0.0200   Min.   : 51.0   treated  : 0
 1st Qu.:0.0600   1st Qu.: 85.0   untreated:11
 Median :0.1100   Median :115.0
 Mean   :0.2764   Mean   :110.7
 3rd Qu.:0.3900   3rd Qu.:137.5
 Max.   :1.1000   Max.   :160.0
> #
> unsplit(df.split, Puromycin$state)
   conc rate     state
1  0.02   76   treated
2  0.02   47   treated
3  0.06   97   treated
.....................
13 0.02   67 untreated
14 0.02   51 untreated
15 0.06   84 untreated
.....................
> #
> split(Puromycin$rate, Puromycin$state)
$treated
 [1]  76  47  97 107 123 139 159 152 191 201 207 200

$untreated
 [1]  67  51  84  86  98 115 131 124 144 158 160
> #
> p.split <- split(Puromycin$rate, Puromycin$state)
> lapply(p.split, mean)
$treated
[1] 141.5833

$untreated
[1] 110.7273
> #
> unsplit(p.split, Puromycin$state)
 [1]  76  47  97 107 123 139 159 152 191 201 207 200  67  51  84
[16]  86  98 115 131 124 144 158 160
```

15.10.2 合併資料框架函式: merge()

使用函式 `merge()` 可以對資料框架合併, 傳回列表物件. 函式指令使用如下:

```
> ## S3 method for class 'data.frame'
> merge(x, y, by = intersect(names(x), names(y)),
        by.x = by, by.y = by, all = FALSE, all.x = all, all.y = all,
        sort = TRUE, suffixes = c(".x",".y"),
        incomparables = NULL, ...)
```

其中引數爲:

- x, y 爲要合併的 資料框架.
- by, by.x = "x.anme", by.y = "y.name"
  設定資料框架 x, y 合併後的資料框架中, 所要依據之共同之變數的欄位名稱 (common columns), 例如, by.x = "x.name", by.y = y.name", 內設相同的變數名 by.
- all = FALSE, all.x = all, all.y = all
  all.x 爲 1 個邏輯指令, 當all.x = TRUE 時, 在所要合併之資料框架 x, 若與資料框架 y 沒有相配之 by.y 列位 (rows), 會附加在合併後之資料框架最後, 並設定在相對應之 by.y 變數爲 NA. all.y 爲 1 個邏輯指令, 作用與 all.x 類似. all 表示 all.x, all.y 有相同之邏輯指令.
- sort = TURE 回傳結果是否須依照 column 排序.
- suffixes = c(".x",".y")]
  在 x, y 合併之資料框架中, 非合併所依據之共同之變數的其餘變數欄位名稱, 若有仍有相同的變數欄位名稱, 則會附加文字在其餘仍有相同名稱變數欄位.
- incomparables 設定一些數值不能被配對.

```
> ## merge()
> x <- data.frame(k1 = c(NA, NA, 3, 4, 5), k2 = c(1, NA, NA, 4, 5), id = 1:5, id1 = 11:15)
> y <- data.frame(k1 = c(NA, 2, NA, 4, 5), k2 = c(NA, NA, 3, 4, 5), id = 1:5, id2 = 11:15)
> cbind(x, y)
  k1 k2 id id1 k1 k2 id id2
1 NA  1  1  11 NA NA  1  11
2 NA NA  2  12  2 NA  2  12
3  3 NA  3  13 NA  3  3  13
4  4  4  4  14  4  4  4  14
5  5  5  5  15  5  5  5  15
> #
> merge(x, y, by = c("id"))
  id k1.x k2.x id1 k1.y k2.y id2
1  1   NA    1  11   NA   NA  11
2  2   NA   NA  12    2   NA  12
3  3    3   NA  13   NA    3  13
```

```
4  4    4    4  14    4    4  14
5  5    5    5  15    5    5  15
> #
> merge(x, y, by.x = c("id1"), by.y = c("id2"))
  id1 k1.x k2.x id.x k1.y k2.y id.y
1  11   NA    1    1   NA   NA    1
2  12   NA   NA    2    2   NA    2
3  13    3   NA    3   NA    3    3
4  14    4    4    4    4    4    4
5  15    5    5    5    5    5    5
> #
> merge(x, y, by = c("k1", "k2")) # NA's match
  k1 k2 id.x id1 id.y id2
1  4  4    4  14    4  14
2  5  5    5  15    5  15
3 NA NA    2  12    1  11
> #
> merge(x, y, by = "k1") # NA's match, so 6 rows
  k1 k2.x id.x id1 k2.y id.y id2
1  4    4    4  14    4    4  14
2  5    5    5  15    5    5  15
3 NA    1    1  11   NA    1  11
4 NA    1    1  11    3    3  13
5 NA   NA    2  12   NA    1  11
6 NA   NA    2  12    3    3  13
> #
> merge(x, y, by = "k2", incomparables = NA) # 2 rows
  k2 k1.x id.x id1 k1.y id.y id2
1  4    4    4  14    4    4  14
2  5    5    5  15    5    5  15
> #
> merge(x, y, by = "k1", all = TRUE)
  k1 k2.x id.x id1 k2.y id.y id2
1  2   NA   NA  NA   NA    2  12
2  3   NA    3  13   NA   NA  NA
3  4    4    4  14    4    4  14
4  5    5    5  15    5    5  15
5 NA    1    1  11   NA    1  11
6 NA    1    1  11    3    3  13
7 NA   NA    2  12   NA    1  11
8 NA   NA    2  12    3    3  13
> #
> merge(x, y, by = "k1", all = FALSE)
  k1 k2.x id.x id1 k2.y id.y id2
1  4    4    4  14    4    4  14
2  5    5    5  15    5    5  15
3 NA    1    1  11   NA    1  11
4 NA    1    1  11    3    3  13
5 NA   NA    2  12   NA    1  11
6 NA   NA    2  12    3    3  13
```

```
> #
> merge(x, y, by = "k1", all.x = TRUE)
  k1 k2.x id.x id1 k2.y id.y id2
1  3   NA    3  13   NA   NA  NA
2  4    4    4  14    4    4  14
3  5    5    5  15    5    5  15
4 NA    1    1  11   NA    1  11
5 NA    1    1  11    3    3  13
6 NA   NA    2  12   NA    1  11
7 NA   NA    2  12    3    3  13
> #
> merge(x, y, by = "k1", all.x = FALSE)
  k1 k2.x id.x id1 k2.y id.y id2
1  4    4    4  14    4    4  14
2  5    5    5  15    5    5  15
3 NA    1    1  11   NA    1  11
4 NA    1    1  11    3    3  13
5 NA   NA    2  12   NA    1  11
6 NA   NA    2  12    3    3  13
> #
> merge(x, y, by = "k1", all.y = TRUE)
  k1 k2.x id.x id1 k2.y id.y id2
1  2   NA   NA  NA   NA    2  12
2  4    4    4  14    4    4  14
3  5    5    5  15    5    5  15
4 NA    1    1  11   NA    1  11
5 NA    1    1  11    3    3  13
6 NA   NA    2  12   NA    1  11
7 NA   NA    2  12    3    3  13
> #
> merge(x, y, by = "k1", all.y = FALSE)
  k1 k2.x id.x id1 k2.y id.y id2
1  4    4    4  14    4    4  14
2  5    5    5  15    5    5  15
3 NA    1    1  11   NA    1  11
4 NA    1    1  11    3    3  13
5 NA   NA    2  12   NA    1  11
6 NA   NA    2  12    3    3  13
> #
> x.id <- c(1:6)
> x.f1 <- rep(c(1, 2), 3)
> x.f2 <- rep(c("m", "f"), each = 3)
> x.f3 <- rep(c("L", "M" , "H"), length.out = 6)
> x.df <- data.frame(x.id = x.id, x.f1 = x.f1, x.f2 = x.f2, x.f3 = x.f3)
> #
> y.id <- c(3:8)
> x.f1 <- rep(c(1, 2),3)
> y.f2 <- rep(c("m", "f"), each = 3)
> y.f3 <- rep(c("L", "M" , "H", "E"), length.out = 6)
> y.df <- data.frame(y.id = y.id, x.f1 = x.f1, y.f2 = y.f2, y.f3 = y.f3)
```

```
> #
> cbind(x.df, y.df)
  x.id x.f1 x.f2 x.f3 y.id x.f1 y.f2 y.f3
1    1    1    m    L    3    1    m    L
2    2    2    m    M    4    2    m    M
3    3    1    m    H    5    1    m    H
4    4    2    f    L    6    2    f    E
5    5    1    f    M    7    1    f    L
6    6    2    f    H    8    2    f    M
> #
> merge(x.df, y.df, by.x = "x.id", by.y = "y.id", sort = FALSE)
  x.id x.f1.x x.f2 x.f3 x.f1.y y.f2 y.f3
1    3      1    m    H      1    m    L
2    4      2    f    L      2    m    M
3    5      1    f    M      1    m    H
4    6      2    f    H      2    f    E
> #
> merge(x.df, y.df, by.x = "x.id", by.y = "y.id", all = TRUE, sort = FALSE)
  x.id x.f1.x x.f2 x.f3 x.f1.y y.f2 y.f3
1    3      1    m    H      1    m    L
2    4      2    f    L      2    m    M
3    5      1    f    M      1    m    H
4    6      2    f    H      2    f    E
5    1      1    m    L     NA <NA> <NA>
6    2      2    m    M     NA <NA> <NA>
7    7     NA <NA> <NA>      1    f    L
8    8     NA <NA> <NA>      2    f    M
> #
> merge(y.df, x.df, by.x = "y.id", by.y = "x.id", all = TRUE, sort = TRUE)
  y.id x.f1.x y.f2 y.f3 x.f1.y x.f2 x.f3
1    1     NA <NA> <NA>      1    m    L
2    2     NA <NA> <NA>      2    m    M
3    3      1    m    L      1    m    H
4    4      2    m    M      2    f    L
5    5      1    m    H      1    f    M
6    6      2    f    E      2    f    H
7    7      1    f    L     NA <NA> <NA>
8    8      2    f    M     NA <NA> <NA>
> #
> merge(x.df, y.df, by.x = "x.id", by.y = "y.id",
      sort = FALSE, all = TRUE, suffixes = c(".X",".Y"))
  x.id x.f1.X x.f2 x.f3 x.f1.Y y.f2 y.f3
1    3      1    m    H      1    m    L
2    4      2    f    L      2    m    M
3    5      1    f    M      1    m    H
4    6      2    f    H      2    f    E
5    1      1    m    L     NA <NA> <NA>
6    2      2    m    M     NA <NA> <NA>
7    7     NA <NA> <NA>      1    f    L
8    8     NA <NA> <NA>      2    f    M
```

15.11 資料框架函式: with()

使用資料框架函式 with() 直接使用 資料框架 data.name 內的變數, 可以在使用資料框架 data.name 內的變數 var.name 時, 避免使用 data.name$var.name 的繁複形式, 也可避免使用 attach() 所帶來不可預測的變數名稱衝突. 函式 with() 的指令爲

```
> with(data, expr, ...)
> within(data, expr, ...)
```

其中引數爲:

- data 爲 資料框架名.
- expr 對 資料框架 data 進行的運算式.

```
> ## with ()
> data(Puromycin)
> with(Puromycin, mean(rate))
[1] 126.8261
> with(Puromycin, fivenum(rate))
[1]  47.0  91.5 124.0 158.5 207.0
> with(Puromycin, t.test(rate ~ state))

        Welch Two Sample t-test
data:  rate by state
t = 1.6375, df = 19.578, p-value = 0.1175
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -8.504864 70.216985
sample estimates:
  mean in group treated mean in group untreated
               141.5833                 110.7273
> #
> ## try
> with(Puromycin, plot(x = conc, y = rate))
> plot(rate ~ conc, data = Puromycin)
```