第 17 章: 基礎統計函式

17: Basic Statistical Function

R 有許多統計函式, 可以對向量物件進行統計分析, R 內建所有常見的基礎敘述統計量, 例如累加與乘積函式, `sum()`, `cumsum()`, `diff()`, `prod()`, `cumprod()`; 樣本統計量函式, 例如, `mean()`, `median()`, `var()`, `sd()`, `range()`, `min()`, `max()`, `quantile()`, `sample(x)` 等, 可參見表 17.1 中之簡要說明.

使用 R 的統計函式進行資料分析時, 須可別注意缺失值, 若資料物件中有缺失值, 一些 R 的統計函式須另外做特別處裡, 可以對資料物件中的缺失值, 先使用函式 `na.omit()` 處理, 然後再進行資料分析, 例如, `mean(na.omit(x))`. 另外, 可使用統計函式內的通用引數, `na.rm = T`, 例如, `mean(x, na.rm = T)` 然後再進行資料分析, 對矩陣或資料框架物件做運算, 若資料物件中有缺失值, 常會有不可預期的結果必須小心.

17.1 敘述統計函式

Functions for Descriptive Statistics

R 內建一些基本敘述統計, 如計算平均值與變異等. `z <- range(x)` 函式回傳一個向量, 二個元素 $[\min(x), \max(x)]$; 極大值與極小值分別爲 `min(x)`, `max(x)`; 求取百分位值可以用 `quantile()`, `fivenum(x)` 回傳向量 $[\max, Q_1, \text{median}, Q_3, \max]$. 參見表 17.1 說明.

表 17.1: 常見敘述性統計函式

格式	說明	
`sum(x)`	加總和 (scalar)	$y = \sum_i x_i$
`cumsum(x)`	累計加總和 (vector)	$z_j = \sum_{i \le j} x_i$
`diff(x)`	`x[i+1]-x[i]`	$z_i = x_{i+1} - x_i$
`prod(x)`	乘積 (product)	$y = \prod_i x_i$
`cumprod(x)`	累計乘積	$z_j = \prod_{i \le j} x_i$
`mean(x)`	平均值 (mean)	$\bar{x} = \frac{1}{n} \sum_i x_i)$
`median(x)`	中位數 (median)	0.5 quantile, 50^{th} percentile
`var(x)`	變異數, 共變異數	$s^2 = \frac{1}{n-1} \sum_i (x_i - \bar{x})^2$
`sd(x)`	標準差 (SD)	$s = \sqrt{s^2}$
`range(x)`	範圍 (range)	$[\min(x), \max(x)]$
`min(x)`	最小值	
`max(x)`	最大值	
`quantile(x)`	百分位數	
`fivenum(x)`	五數摘要 (five-number summary)	$[\max, Q_1, \text{median}, Q_3, \max]$
`sample(x)`	隨機抽樣	random sample

```
> # descriptive statistics
> (x <- seq(-2, 3, 0.3))
 [1] -2.0 -1.7 -1.4 -1.1 -0.8 -0.5 -0.2  0.1  0.4  0.7  1.0  1.3  1.6
[14]  1.9  2.2  2.5  2.8
> sum(x)
```

```
[1] 6.8
> cumsum(x)
 [1] -2.0 -3.7 -5.1 -6.2 -7.0 -7.5 -7.7 -7.6 -7.2 -6.5 -5.5 -4.2 -2.6
[14] -0.7  1.5  4.0  6.8
> diff(x)
 [1] 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3
> prod(x)
[1] -0.713814
> cumprod(x)
 [1] -2.0000000  3.4000000 -4.7600000  5.2360000 -4.1888000  2.0944000
 [7] -0.4188800 -0.0418880 -0.0167552 -0.0117286 -0.0117286 -0.0152472
[13] -0.0243956 -0.0463516 -0.1019735 -0.2549337 -0.7138144
>
> mean(x)
[1] 0.4
> median(x)
[1] 0.4
> var(x)
[1] 2.295
> sd(x)
[1] 1.51493
> range(x)
[1] -2.0  2.8
> min(x)
[1] -2
> max(x)
[1] 2.8
>
> (y <- quantile(x, probs = c(0.05, 0.25, 0.5, 0.75, 0.95)))
   5%   25%   50%   75%   95%
-1.76 -0.80  0.40  1.60  2.56
> # quantile range
> y[4]-y[2]
75%
2.4
>
> y[4]-y[2]
75%
2.4
>
> fivenum(x)
[1] -2.0 -0.8  0.4  1.6  2.8
>
> # missing values
> x[3] <- NA
> x[7] <- NA
> x
 [1] -2.0 -1.7   NA -1.1 -0.8 -0.5   NA  0.1  0.4  0.7  1.0  1.3  1.6
[14]  1.9  2.2  2.5  2.8
> mean(x)
```

```
[1] NA
> mean(na.omit(x))
[1] 0.56
> var(x, na.rm = T)
[1] 2.33829
```

17.2 機率函式與亂數生成函式

Probability Function

R 具有詳盡的機率函式與亂數生成函式參見表 17.2, 如令 X 爲一隨機變數 (random variable), 定義

累積機率分配函數, cumulative distribution function (17.2.1)

$$p = F(q) = P[X \leq q] \tag{17.2.2}$$

分位數函數, quantile function (17.2.3)

$$q = Q(u)F^{-1}(p),\ p \leq P[X \leq q] \tag{17.2.4}$$

機率密度分配函數, probability density function (17.2.5)

$$d = f(x) = F'(x) = P[X = x] \tag{17.2.6}$$

隨機亂數函數, random number (17.2.7)

$$r = R(r) = f^{-1}(x),\ f = f(X = x) \tag{17.2.8}$$

在 R 中, 相同的機率函數 (probability function), 使用相同的機率名稱, `ProbFun`, 在機率名稱, `ProbFun`, 之間, 加上 4 種不同的小寫字母, `p`, `q`, `d`, `r`, 產生不同的機率

函式 (17.2.1 – 17.2.8). 例如, fProbFun), 表示用相同的機率函式含義, 產生上述不同之結果.

p 表示 累積機率分配函數 (cumulative distribution function, CDF).

q 表示 分位數 (quantile), 符合 $u \leq P(X <= x)$ 的最小 x.

d 表示 機率密度函數 (probability density function, pdf).

r 表示 隨機模擬 或 "(偽) 隨機亂數", "隨機亂數" 生成函式 (pseudo-random number generation function, random number).

dProbFun 的第一個參數是 x.

pProbFun 的第一個參數是 q.

qProbFun 的第一個參數是 p.

rProbFun 的的第一個參數是 n, 生成亂數之數目.

pProbFun 和 qProbFun 函數 都有邏輯引數 lower.tail 和 log.p.

若 lower.tail = TRUE (default), 機率計算爲 $P[X <= x]$.

若 lower.tail = FALSE 機率計算爲 $P[X > x]$.

若 log.p = TRUE, 機率 p 是以 log(p) 輸入與輸出.

dProbFun 也有一個邏輯引數 log, 用來計算所要的對數機率值.

非中央參數 (non-centrality parameter), ncp 現在僅用於累積分配函數, 此外還有函是 ptukey() 和 qtukey() 計算 Studentized Range Distribution. 所有細節的內容可以參考輔助文檔.

```
> # normal distribution
> pnorm(1.96)
[1] 0.9750021
> qnorm(0.975)
[1] 1.959964
> dnorm(1.96)
[1] 0.05844094
> rnorm(5, mean = 0, sd = 1)
[1]  0.26489830 -1.73168534  0.02819288  1.04172067  1.47740736
```

```
> rnorm(5, mean = 3, sd = 3)
[1]  3.7804748 -0.1957517  5.8409855  6.5761660  0.5526837
>
> # t distribution
> qt(0.995, df = 2)
[1] 9.924843
> 2*pt(-1.96, df = 2)
[1] 0.1890573
> 2*pt(-1.96, df = 30)
[1] 0.05934231
>
> # upper 1% point for an F(1, 2) distribution
> sqrt(qf(0.99, 1, 2))
[1] 9.924843
>
> # Simulation different shapes of distribution
> par(mfrow = c(2,2))
> histSYM <- hist(rnorm(10000),nclas = 100,freq = FALSE,
+  main = "Symmetric Distribution", xlab = "")
> histFLAT <- hist(runif(10000),nclas = 100,freq = FALSE,
+  main = "Symmetric Flat Distribution", xlab = "")
> histSKR <- hist(rgamma(10000,shape = 2,scale = 1),freq = FALSE, nclas = 100,
+  main = "Skewed to Right", xlab = "")
> histSKL <- hist(rbeta(10000,8,2),nclas = 100,freq = FALSE,
+  main = "Skewed to Left", xlab = "")
> par(mfrow = c(1,1))
```

通常產生亂數序列希望是不會重復的, 實際上, R 在現在操作視窗下, 第一次產生時亂數時, 從當下時間 (current time), 生成一個 種子 (seed) 出發, 不斷迭代更新產生隨機均等分配亂數 (uniform random number), 所以不同時間下執行 R, 啟用不同的種子, 隨後內部的隨機種子就已經改變了, 模擬亂數是不會重復的, 有時我們需要模擬結果是可重復的亂數序列, 此時需要用函式 `set.seed()`, 在每次產生僞隨機亂數之前, 把種子設定種子爲某一特定正整數即可.

```
> # seed
> runif(5)
[1] 0.50747820 0.30676851 0.42690767 0.69310208 0.08513597
> runif(5)
[1] 0.2254366 0.2745305 0.2723051 0.6158293 0.4296715
> set.seed(10)
> runif(5)
[1] 0.50747820 0.30676851 0.42690767 0.69310208 0.08513597
> set.seed(10)
> runif(5)
[1] 0.50747820 0.30676851 0.42690767 0.69310208 0.08513597
> # norm
```

```
> rnorm(5)
[1] -0.7539600 -0.6058564 -0.1772105  0.1706176  0.2428141
> rnorm(5)
[1] -0.1794061 -0.6305186  0.9786930  0.2932970 -0.3703290
> set.seed(10)
> rnorm(5)
[1]  0.01874617 -0.18425254 -1.37133055 -0.59916772  0.29454513
> set.seed(10)
> rnorm(5)
[1]  0.01874617 -0.18425254 -1.37133055 -0.59916772  0.29454513
```

有時統計模擬需要的計算量很大, 很多的時候甚至要計算幾天的時間. 對於這種問題要把問題拆分成可以單獨計算的小問題, 然後單獨計算每個小問題, 把結果儲存在 R 物件中或文字檔案中, 最後綜合每個小問題之結果得到最終結果.

表 17.2: R 機率與亂數生成函式

機率分配	R 函式名 (ProbFun)	引數
beta	`beta`	shape1, shape2, ncp
binomial	`binom`	size, prob
Cauchy	`cauchy`	location, scale
chi-squared	`chisq`	df, ncp
exponential	`exp`	rate
F	`f`	df1, df1, ncp
gamma	`gamma`	shape, scale
geometric	`geom`	prob
hypergeometric	`hyper`	m, n, k
log-normal	`lnorm`	meanlog, sdlog
logistic	`logis`	location, scale
negative binomial	`nbinom`	size, prob
normal	`norm`	mean, sd
Poisson	`pois`	lambda
Student's	`t`	t df, ncp
uniform	`unif`	min, max
Weibull	`weibull`	shape, scale
Wilcoxon	`wilcox`	m, n

17.3 隨機抽樣: sample() 函式

在 R 中有一個常用的隨機抽樣函式, `sample()`, 用法如下

```
> sample(x, size, replace = FALSE, prob = NULL)
```

其中引數

`x`

`x` 爲一長度大於 1 的任意向量, 或是一個正整數.

`size = k`

`size = k` 設定所要抽出之樣本數.

`prob`

`prob` 設定每一個個體被抽取之相對應機率或比率之向量, 若無設定值, 則每一個個體被抽取之相對應機率爲相等.

`replace = FALSE`

`replace = FALSE` 1 個邏輯指令, 設定是否可重複抽取.

```
> # sample
> # a random sampling (permutation)
> # sampling 5 subjects from 10 subjects
> # without replacement
> x <- 1:10
> sample(x, size = 5, replace = FALSE)
[1]  7 10  9  8  3
>
> # equal rpobability
> x <- 1:10
> sample(x, size = 5, replace = FALSE, prob = c(1:10))
[1]  2  7  3  8 10
> sample(x, size = 5, replace = FALSE, prob = c(rep(1,10)/10.0))
[1]  5  8 10  6  7
>
> # unequal rpobability
> y <- 1:3
> sample(y, size = 2, replace = FALSE, prob = c(0.1, 0.6, 0.3))
[1] 2 3
> sample(y, size = 10, replace = TRUE, prob = c(0.25, 0.3, 0.45))
```

```
 [1] 3 2 2 3 3 1 3 3 3 1
> # clinical trials
> # randomization
> # random assign to two groups, total 20 subjects
> # random assigning treatment groups
> sample(2, size = 20, replace = TRUE)
 [1] 2 1 1 2 1 2 1 2 2 1 1 1 1 1 1 1 1 2 1 1
>
> # random choose 10 subjects to group 1
> sample(20, size = 10, replace = FALSE)
 [1] 10 17 18  9  4 20  5 14  3  7
>
> # block randomization
> # total 3 blocks, block size 4, choose 2 subjects to group 1
> replicate(3, sample(c(1:4), size = 2, replace = FALSE))
     [,1] [,2] [,3]
[1,]    2    4    4
[2,]    4    2    1
>
> # bootstrap sampling -- only if length(x) > 1 !
> sample(1:10,replace = TRUE)
 [1]  5 10  2  9  9  4  4  2  1  3
>
> # 20 Bernoulli trials
> sample(c(0,1), size = 20, replace = TRUE)
 [1] 0 1 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1 1
>
> ## More careful bootstrapping --  Consider this when using sample()
> ## programmatically (i.e., in your function or simulation)!
> # sample()'s surprise -- example
> x <- 1:10
> sample(x[x >  8]) # length 2
[1]  9 10
> sample(x[x >  9]) # oops -- length 10!
 [1]  3 10  1  4  6  2  9  8  7  5
> try(sample(x[x > 10]))# error!
Error: sample(length(x), size, replace, prob) :
        'x' incorrect argument
```