第 8 章: 文字函式

8: Functions for Characters and Strings

傳統上, 統計人員較少直接處裡文字或字串資料, 多數時候是由資料管理人元處理後, 轉換成數值資料, 然後交由統計人員進行後續分析. 由於大數據時代來臨包含者多樣性的資料型態, 統計人員必須必須直接處裡文字或字串資料的機會也越來越多. R 內有許多函數可以處理文字型態的資料物件 或 文字資料 (Character Data), 常用之文字函式有 `paste()`, `substr()`, `substring()`, `grep()`, `gsub()`, `strsplit()` 等. R 套件 `stringr` 有更多處理文字或字串資料函式.

8.1 文字合併函式: paste()

文字合併函式 `paste()` 可以合併 2 個文字向量中的元素使其合而爲 1 個文字向量.

```
> paste (..., sep = " ", collapse = NULL)
> paste0(..., collapse = NULL)
```

其中引數爲

- `sep = " "` 設定 2 個字串向量合併時, 各別元素中間分隔使用的文字或符號.

  回傳字串長度, 長度爲原有字串向量的長度, R 內設爲 1 各空白鍵.

- collapse 先依照 sep 設定回傳字串長度後, 然後再次合併此字串向量的元素後成爲回傳單一字串, 長度爲 1. collapse 設定最後元素中間分隔使用的文字或符號.

```
> ## paste()
> paste(1:5) # same as as.character(1:5)
[1] "1" "2" "3" "4" "5"
> paste("Today is", date())
[1] "Today is Sat Mar 11 11:37:40 2017"
> #
> paste("A", 1:5, sep = "")
[1] "A1" "A2" "A3" "A4" "A5"
> paste("A", 1:5, sep = " ")
[1] "A 1" "A 2" "A 3" "A 4" "A 5"
> paste("A", 1:5, sep = "#")
[1] "A#1" "A#2" "A#3" "A#4" "A#5"
> #
> paste(c("X", "Y"), 1:5)
[1] "X 1" "Y 2" "X 3" "Y 4" "X 5"
> paste(c("X", "Y"), 1:5, sep = " ")
[1] "X 1" "Y 2" "X 3" "Y 4" "X 5"
> paste(c("X", "Y"), 1:5, sep = "")
[1] "X1" "Y2" "X3" "Y4" "X5"
> paste(c("X", "Y"), 1:5, sep = "+")
[1] "X+1" "Y+2" "X+3" "Y+4" "X+5"
> paste(c("X", "Y"), 1:5, sep = "", collapse = NULL)
[1] "X1" "Y2" "X3" "Y4" "X5"
> paste(c("X", "Y"), 1:5, sep = "", collapse = " + ")
[1] "X1 + Y2 + X3 + Y4 + X5"
```

8.2 萃取文字函式: substr() 與 substring()

文字函式 substr() 與 substring() 可在 1 個文字向量中, 萃取傳回部分字串, 或替換部分字串.

```
> substr(x.char, start, stop)
> substring(x.text, first, last = 1000000L)
> substr(x.char, start, stop) <- new.char
> substring(x.text, first, last = 1000000L) <- new.char
```

其中引數

- x.char 爲 1 個文字字串向量,

- start, 爲 1 個正整數, 表示所要萃取文字字串的第 1 個文字位置,
- stop, 爲 1 個正整數, 表示所要萃取文字字串的最後 1 個文字位置.
- x.text 爲 1 個文字字串向量.
- first 爲 1 個正整數, 表示所要萃取文字字串的第 1 個文字位置.
- last 爲 1 個正整數, 表示所要萃取文字字串的最後 1 個文字位置.
- 使用其它文字字串來取代原先所萃取之文字字串:
  - substr(x.char, start, stop) <- new.char
  - substring(x.text, first, last = 1000000) <- new.text

  可以指派新的文字字串 new.char 或 new.text 取代原先所萃取之文字字串之位置.

利用 R 內建資料框架 state, 關於美國 50 洲的一些相關資料, 包含向量物件, state.abb, state.area, state.center, state.division, state.name, state.region, 以及矩陣物件 state.x77, 在此使用 substr() 與 substring() 函式, 分析文字向量物件 state.name.

```
> ## substr()
> data(state)
> state.name
 [1] "Alabama"       "Alaska"      "Arizona"       "Arkansas"      "California"
 [6] "Colorado"      "Connecticut" "Delaware"      "Florida"       "Georgia"
[11] "Hawaii"        "Idaho"       "Illinois"      "Indiana"       "Iowa"
[16] "Kansas"        "Kentucky"    "Louisiana"     "Maine"         "Maryland"
[21] "Massachusetts" "Michigan"    "Minnesota"     "Mississippi"   "Missouri"
[26] "Montana"       "Nebraska"    "Nevada"        "New Hampshire" "New Jersey"
[31] "New Mexico"    "New York"    "North Carolina" "North Dakota"  "Ohio"
[36] "Oklahoma"      "Oregon"      "Pennsylvania"  "Rhode Island"  "South Carolina"
[41] "South Dakota"  "Tennessee"   "Texas"         "Utah"          "Vermont"
[46] "Virginia"      "Washington"  "West Virginia" "Wisconsin"     "Wyoming"
> state.name[47:50]
[1] "Washington"    "West Virginia" "Wisconsin"     "Wyoming"
> substr(state.name[47:50], start = 1, stop = 4)
[1] "Wash" "West" "Wisc" "Wyom"
> substr(state.name[47:50], start = 1, stop = 4) <- "AAA"
> state.name[47:50]
```

```
[1] "AAAhington"    "AAAt Virginia" "AAAconsin"     "AAAming"
> #
> ## substring()
> data(state)
> state.name[47:50]
[1] "Washington"    "West Virginia" "Wisconsin"     "Wyoming"
> substring(state.name[47:50], first = 3, last = 1000)
[1] "shington"    "st Virginia" "sconsin"     "oming"
> substring(state.name[47:50], first = 3, last = 1000) <- "BBB"
> state.name[47:50]
[1] "WaBBBngton"    "WeBBBVirginia" "WiBBBnsin"     "WyBBBng"
```

8.3 簡化文字函式: abbreviate()

文字函式 abbreviate() 方便用來簡化文字字串, 使用如下:

```
> abbreviate(names.arg, minlength = 4, use.classes = TRUE,
             dot = FALSE, strict = FALSE,
             method = c("left.kept", "both.sides"))
```

其中引數

- names.arg 爲 1 個向文字字串向量.
- minlength = 4 爲簡化後文字字串的字元數目.
- dot = FALSE 爲邏輯向量, 是否須要附加 "." 在簡化的文字字串內.
- strict = FALSE 爲邏輯向量, 爲邏輯向量, 是否須要絕對遵循 minlength 的設定, 若 strict = TRUE 則可能不會形成唯一的簡化, 某些元素原本是不同文字, 這些元素在簡化後可能會變成相同的文字.

```
> ## abbreviate()
> data(state)
> state.name[47:50]
[1] "Washington"    "West Virginia" "Wisconsin"     "Wyoming"
> abbreviate(state.name[47:50], minlength = 2, dot = TRUE)
  Washington West Virginia     Wisconsin       Wyoming
       "Wsh."         "WV."        "Wsc."         "Wy."
>
> x.chr <- c("abcd", "efgh", "abce")
```

```
> abbreviate(x.chr, 2)
  abcd   efgh   abce
"abcd"   "ef" "abce"
> abbreviate(x.chr, 2, strict = TRUE)  # 1st and 3rd are = =  "ab"
abcd efgh abce
"ab" "ef" "ab"
> abbreviate(x.chr, 2, strict = FALSE) #
  abcd   efgh   abce
"abcd"   "ef" "abce"
>
> x.abb = abbreviate(state.name[47:50], minlength = 2, method = "left.kept")
> y.abb = abbreviate(state.name[47:50], minlength = 2, method = "both")
> cbind(x.abb, y.abb)
               x.abb y.abb
Washington     "Wsh" "Ws"
West Virginia  "WV"  "WV"
Wisconsin      "Wsc" "Wn"
Wyoming        "Wy"  "Wy"
```

8.4 文字搜尋函式: grep()

文字處理必須要有一個重要的功能, 就是在字串中 搜尋配對, 修正 與 取代 關鍵字 或 特定型式字串, 或取得特定型式字串所在的下標位置. 特定型式字串 R 稱爲 regular expression (Regular expression is a pattern that describes a set of strings), 程式語言 Perl 與 Python 都具備此功能. R 也可用 Perl 的搜尋配對或取代方式. regular expression 是由搜尋一個單字逐步形成搜尋特定型式字串, 執行搜尋取代功能前有些事項必須先注意, 尤其是 metacharacters (特殊符號字元).

- single character: 搜尋或取代單字.
- character class: 同時搜尋或取代多個單字.
- concatenation: 搜尋單字且合併形成爲特定型式字串.
- repetition: 搜尋或取代特定型式字串的次數.
- alternation: 交互輪流, 在多個特定型式字串中, 搜尋或取代某一特定型式字串.

- 使用萬用字元符號或是搜尋標點符號.
- metacharacters: 特殊符號字元 以及 R 保留的關鍵字. 搜尋取代特殊符號字元必須在特殊符號字元前加上雙反斜線 "\\".

```
1 ! " # $ % & âĂŹ ( ) * + , - . / : ; < = > ? @ [ \ ] ^ - ‘ {  }
```

萬用字元符號與 特殊符號字元參見 表 8.1 - 表 8.2.

```
1  > grep(pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE,
2        fixed = FALSE, useBytes = FALSE, invert = FALSE)
3
4  > grepl(pattern, x, ignore.case = FALSE, perl = FALSE,
5         fixed = FALSE, useBytes = FALSE)
6
7  > sub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
8       fixed = FALSE, useBytes = FALSE)
9
10 > gsub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
11       fixed = FALSE, useBytes = FALSE)
12
13 > regexpr(pattern, text, ignore.case = FALSE, perl = FALSE,
14          fixed = FALSE, useBytes = FALSE)
15
16 > gregexpr(pattern, text, ignore.case = FALSE, perl = FALSE,
17           fixed = FALSE, useBytes = FALSE)
18
19 > regexec(pattern, text, ignore.case = FALSE,
20          fixed = FALSE, useBytes = FALSE)
```

其中引數

- pattern: 特定型式字串.
- x, text: 文字向量或物件.
- ignore.case = FALSE: 設定大小寫有差異不可忽略.
- perl = FALSE: 使用 perl 程式語言的特定型式字串.
- value = FALSE: 回傳向量正整數下標, value = TRUE: 回傳向量符合特定型式的元素.
- fixed = FALSE: 設定特定型式字串是否須完全相符合.

- useBytes = FALSE: 設定搜尋取代爲 character-by-character 而非 byte-by-byte.
- invert = TRUE: 設定搜尋取代爲反向操作, 傳回不符合特定型式字串的下標位置或文字.

8.4.1 搜尋或取代單字

任何搜尋或取代方式都是由搜尋一個單字逐步形成搜尋特定型式字串. 例如, 在文字向量 x.chr 搜尋單字 x, 使用函式 grep() 在萃取文字向量 x.chr 中, 尋找的某一個特定式樣 (pattern), 並且傳回文字向量的下標 (index).

```
> ## single character
> ## grep()
> x.chr <- c("y", "ya bc", "abc", "ay bc", "ab yc", "ab cy")
> grep("y", x.chr)
[1] 1 2 4 5 6
> x.chr[grep("y", x.chr)] # index by position
[1] "y"     "ya bc" "ay bc" "ab yc" "ab cy"
```

特定型式字串的前一位置 (pre-pattern) 使用 ^ 字元 (caret, 脫字符號, 倒 V 符號), 則僅搜尋 1 行字串的第 1 個字元是否符合特定型式字串.

```
> ## ^ (caret) line first
> x.chr <- c("y", "ya bc", "abc", "ay bc", "ab yc", "ab cy")
> grep("^y", x.chr)
[1] 1 2
> x.chr[grep("^y", x.chr)]
[1] "y"     "ya bc"
```

特定型式字串的後一位置 (pattern-post) 使用 $ 字元 (dollar sign, 金錢符號), 則僅搜尋 1 行字串的最後 1 個字元是否符合特定型式字串. metacharacters.

```
> ## $ (dollar sign) line last
> x.chr <- c("y", "ya bc", "abc", "ay bc", "ab yc", "ab cy")
> grep("y$", x.chr)
[1] 1 6
> x.chr[grep("y$", x.chr)]
[1] "y"     "ab cy"
```

特殊符號字元 "." (句點, period), 搜尋任何符合特定型式字串的字串, 包含空格. ^ 字元, $ 字元 與 "." 字元 是所謂的特殊符號字元

```
> x.chr[grep(".bc", x.chr)]
[1] "ya bc" "abc"   "ay bc"
```

僅搜尋文字 (word) 的第 1 個字元是否符合特定型式字串, 但不要搜尋 1 行字串的第 1 個字元, 則特定型式字串的前一位置增加 1 空格. 同樣的, 僅搜尋文字 (word) 的最後 1 個字元是否符合特定型式字串, 但不要搜尋 1 行字串的最後 1 個字元, 則特定型式字串的後一位置增加 1 空格.

```
> ## word first, except line first
> x.chr <- c("y", "ya bc", "abc", "ay bc", "ab yc", "ab cy")
> grep(" y", x.chr)
[1] 5
> x.chr[grep(" y", x.chr)]
[1] "ab yc"
> #
> ## word last, except line last
> x.chr <- c("y", "ya bc", "abc", "ay bc", "ab yc", "ab cy")
> grep("y ", x.chr)
[1] 4
> x.chr[grep("y ", x.chr)]
[1] "ay bc"
```

8.4.2 同時搜尋或取代多個單字

R 在 regular expression 中定義 character class 是由中括號包圍的特定型式字串 [and], 且 grep() 搜尋時, 只要中括號包圍內的字元中的任一個單字符合, 都會回傳. 例如, 若特定型式字串爲 "[cgr]" 1 個字元只要與 "c", "g", "r" 任一個單字符合, 都會回傳. 若特定型式字串爲 "^[cgr]", 則文字向量內的 1 行的第 1 個字只要與 "c", "g", "r" 任一個單字符合, 都會回傳. 反之, 若 "^[^cgr]" 則文字向量內的 1 行的第 1 個字元不含任一個單字 "c", "g", "r" 才會回傳. 參見 表 8.1 - 表 8.2.

```
> ## character class
> x.chr <- c("cat", "bar", "rat", "sac", "mug", "gate")
> grep("[cgr]", x.chr)
[1] 1 2 3 4 5 6
> x.chr[grep("[cgr]", x.chr)]
[1] "cat"  "bar"  "rat"  "sac"  "mug"  "gate"
> #
> x.chr <- c("cat", "bar", "rat", "sac", "mug", "gate")
```

```
> grep("^[cgr]", x.chr)
[1] 1 3 6
> x.chr[grep("^[cgr]", x.chr)]
[1] "cat"  "rat"  "gate"
> #
> x.chr <- c("cat", "bar", "rat", "sac", "mug", "gate")
> grep("^[^cgr]", x.chr)
[1] 2 4 5
> x.chr[grep("^[^cgr]", x.chr)]
[1] "bar" "sac" "mug"
```

例如, "^.[^a].+" 搜尋任何第 1 個單字, 接著是 非 a 單字, 接續爲至少 1 個或多個單字.

```
> x.chr <- c("cat", "bar", "rat", "sac", "mug", "gate")
> grep("^.[^a].+", x.chr)
[1] 5
> x.chr[grep("^.[^a].+", x.chr)]
[1] "mug"
```

文字 搜尋時, 也可使用文字範圍設定, 例如,

- [0-9]: 搜尋數字 0-9.
- [A-Z]: 搜尋大寫字母 A-Z.
- [a-z]: 搜尋小寫字母 a-z.
- [0-9A-Za-z]: 同時搜尋數字與字母.

表 8.1 顯示更多的用法.

8.4.3 合併成特定字串

搜尋單一字母可以搜尋特定字合併使用, 例如, "^[cgr]at$", 搜尋 1 行的第 1 個字元爲任 1 個 c, g, r, $ 搜尋 1 行字串的最後字元爲 "at", 因此可搜尋出 "cat", "rat" 等字. 若要 3 個字元, 可使用 "^[ct]a[br]$".

```
> ##
> x.chr <- c("cat", "car", "bar", "rat", "sac", "mug", "gate", "tab")
> grep("^[ct]a[br]$", x.chr)
[1] 2 8
```

```
> x.chr[grep("^[bct]a[br]$", x.chr)]
[1] "car" "bar" "tab"
> #
> x.chr <- c("cat", "car", "bar", "rat", "sac", "mug", "gate", "tab")
> grep("^[ct]a[br]$", x.chr)
[1] 2 8
```

句點 (period) "." 爲 metacharacters, 作爲萬用字元, 可以搜尋配對任 1 個字元. 例如, "c.t" 搜尋配對啟始爲 c + 任何字元 + t.

```
> ## period (.)
> x.chr <- c("cat", "cut", "bar", "cattle", "sat", "cup", "city")
> grep("c.t", x.chr)
[1] 1 2 4 7
> x.chr[grep("c.t", x.chr)]
[1] "cat"    "cut"    "cattle" "city"
```

8.5 搜尋或取代特定型式字串次數

搜尋特定型式字串, 並不受限 1 次, 可以設定在一字串中多次重覆搜尋取代, 或設定重覆搜尋取代的次數, 參見 表 8.2. 例如, "+" 表示搜尋配對同一元素內字串至少 1 次以上, "^[cC].+e$" 搜尋配對啟始爲 c 或 C + 任何字元 + 最後字元爲 e.

```
> ## Repetition
> x.chr <- c("cat", "cute", "bar", "cattle", "sat", "cup", "city", "Centigrade")
> grep("^[cC].+e$", x.chr)
[1] 2 4 8
> x.chr[grep("^[cC].+e$", x.chr)]
[1] "cute"       "cattle"     "Centigrade"
```

設定重覆搜尋取代的次數時, 可以使用大括號 "{", "}" 包含設定的次數. 例如, 搜尋 1 個字元 "m" 至少 2 次, 可以使使用 "m{2,}". 若要搜尋多個字元 "tom" 至少 2 次, 可以使使用 "(tom){2,}".

```
> ## Repetition
> x.chr <- c("Tom", "Thomas", "Tomas", "Tommy", "tomas", "tommas", "Tomtom")
> grep("m{2,}", x.chr)
[1] 4 6
> x.chr[grep("m{2,}", x.chr)]
[1] "Tommy"  "tommas"
> #
```

```
> x.chr <- c("Tom", "Thomas", "Tomas", "Tommy", "tomas", "tommas", "Tomtom")
> grep("[Tt]om{2,}", x.chr)
[1] 4 6
> x.chr[grep("[Tt]om{2,}", x.chr)]
[1] "Tommy"  "tommas"
> #
> x.chr <- c("Tom", "Thomas", "Tomas", "Tommy", "tomas", "tommas", "Tomtom")
> grep("([Tt]om){2,}", x.chr)
[1] 7
> x.chr[grep("([Tt]om){2,}", x.chr)]
[1] "Tomtom"
```

8.5.1 交互輪流搜尋或取代

資料處裡有時須同時搜尋配對 2 種以上特定型式字串, 此使可以使用 "|" (vertical bar) 結合 2 種以上特定型式字串. 例如, B 型肝炎 ICD-10 疾病診斷碼,

```
B16, B16.0, B16.1, B16.2, B16.3, B16.4,
B16.5, B16.7, B16.8, B16.9, B17, B17.0, B17.2, B17.8,
B18, B18.0, B18.1, B18.8, B18.9
```

但是, 保險資料庫內的 B 型肝炎疾病診斷碼並未包含 "." (period), 因此呈現爲

```
B16, B160, B161, B162, B163, B164, B165, B167, B168, B169,
B17, B170, B172, B178,
B18, B180, B181, B188, B189
```

若要將保險資料庫內的疾病診斷碼與正式的 ICD 疾病診斷碼搜尋配對, 可使用

"^B16[0-9]?$|^B17[0,2,8]?$|^B18[0,1,8,9]?$"

上述搜尋配對 "^B16[0-9]?$" 或 "^B17[0,2,8]?$" 或 "^B18[0,1,8,9]?$"".

```
> ## Alternation
> hbv.chr <- c("B16", "B160", "B161", "B162", "B163", "B164",
               "B165", "B167", "B168", "B169",
               "B17", "B170", "B172", "B178",
               "B18", "B180", "B181", "B188", "B189")
> grep("^B16[0-9]?$", hbv.chr)    # B16
 [1]  1  2  3  4  5  6  7  8  9 10
> grep("^B17[0,2,8]?$", hbv.chr)  # B17
[1] 11 12 13 14
> grep("B18[0,1,8,9]?$", hbv.chr) # B18
[1] 15 16 17 18 19
> #
> grep("^B16[0-9]?$^B17[0,2,8]?$^B18[0,1,8,9]?$", hbv.chr) # any
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
> ## misspelling name
> x.chr <- c("Tom", "Thomas", "Tomas", "Tommy", "tomas")
> grep("^[Tt]omm?y?$^[Tt]homas$+tomas$", x.chr)
[1] 1 2 4
> x.chr[grep("^[Tt]omm?y?$^[Tt]homas$+tomas$", x.chr)]
[1] "Tom"    "Thomas" "Tommy"
```

8.5.2 特殊字元與標點符號

R 在搜尋或取代時會用到 metacharacters (特殊符號字元) 或 萬用字元 方便搜尋或取代, 以下爲常用萬用字元的搜尋或取代.

```
"[0-9]" - Digits
"[a-z]" - Lower-case letters
"[A-Z]" - Upper-case letters
"[a-zA-Z]" - Alphabetic characters
"[^a-zA-Z]" - Non-alphabetic characters
"[a-zA-Z0-9]" - Alphanumeric characters
"[ \t\n\r\f\v]" - Space characters
"[]$*+.?[^{(\\#%&~_/< = >'!,:;‘\")}@-]" - Punctuation Characters
```

```
> ## any string containing any of the characters
> x.chr <- c("nose", "letter38", "window9", "apple0")
> grep("[nco]", x.chr, value = TRUE)
[1] "nose"    "window9"
> grep("[01234567]", x.chr, value = TRUE)
[1] "letter38" "apple0"
> grep("[0-7]", x.chr, value = TRUE)
[1] "letter38" "apple0"
```

這些特殊符號字元也是標點符號, 因此在搜尋或取代標點符號必須特殊處理. 通常是使用雙反斜線 \\, 例如, 使用 gsub() 搜尋 "$", 取代 "." (period).

```
> ## metacharacters
> s.chr = "gsub$uses$regular$expressions"
> s.err <- gsub(pattern = "$", replacement = ".", s.chr)
> s.err
[1] "gsub$uses$regular$expressions."
> s.correct <- gsub(pattern = "\\$", replacement = ".", s.chr)
> s.correct
[1] "gsub.uses.regular.expressions"
```

在 character class 搜尋或取代這些字元的處理規則, 比較誤用與正確方式.

```
> ## metacharacters
> ## example: wrong methods
> metaChar = c("$","*","+",".","?","[","^","{","","(","\\")
> grep(pattern = "$", x = metaChar, value = TRUE)
> grep(pattern = "\\", x = metaChar, value = TRUE)
> grep(pattern = "(", x = metaChar, value = TRUE)
> gsub(pattern = "", replacement = ".",
       "gsubusesregularexpressions")
> strsplit(x = "strsplit.aslo.uses.regular.expressions",
           split = ".")
> ## examples: correct methods
> metaChar = c("$","*","+",".","?","[","^","{","","(","\\")
> grep(pattern = "\\$", x = metaChar, value = TRUE)
[1] "$"
> grep(pattern = "$", x = metaChar, value = TRUE, fixed = TRUE)
[1] "$"
> grep(pattern = "\\\\", x = metaChar, value = TRUE)
[1] "\\"
> grep(pattern = "\\", x = metaChar, value = TRUE, fixed = TRUE)
[1] "\\"
> grep(pattern = "\\(", x = metaChar, value = TRUE)
[1] "("
> grep(pattern = "(", x = metaChar, value = TRUE, fixed = TRUE)
[1] "("
> #
> gsub(pattern = "\\", replacement = ".",
       "gsubusesregularexpressions")
[1] "gsub.uses.regular.expressions"
> strsplit(x = "strsplit.aslo.uses.regular.expressions",
           split = "\\.")
[[1]]
[1] "strsplit"    "aslo"        "uses"        "regular"     "expressions"

> strsplit(x = "strsplit.aslo.uses.regular.expressions",
           split = ".", fixed = TRUE)
[[1]]
[1] "strsplit"    "aslo"        "uses"        "regular"     "expressions"
> #
> ## "\\+" at least once
x.chr <-  c("id...of....patient", "patient....age")
gsub(pattern = "\\.+", replacement = ".", x = x.chr)
```

上述在搜尋或取代特殊字元時, 在標點符號前必須使用雙反斜線 \\, 但是以下字元仍須例外處裡.

```
] ^ - \
```

- 搜尋 "]" 須放在 character class 第 1 位置

- "-" 須放在 character class 第 1 位置 或 最後位置
- "^" 不可放 character class 在第 1 位置 (因第 1 位置代表排除 character class 內的字元)
- 除了 "\" 之外, 搜尋其他特殊字元時 可放在 character class 任何位置.

```
> ## metacharacters
> x.str <- "my\\strin\\g\\with\\slashes"
> x.str
[1] "my\\strin\\g\\with\\slashes"
> print(x.str)
[1] "my\\strin\\g\\with\\slashes"
> cat(x.str)
my\strin\g\with\slashes
> gsub("([\\])","", x.str)
[1] "mystringwithslashes"
> gsub("\\", "", x.str, fixed = TRUE)
[1] "mystringwithslashes"
> #
```

表 8.1: 常見萬用字元設定 I

萬用字元	替代簡寫	說明
[[:lower:]]	[a-z]	英文小寫字母 lower-case letters
[[:upper:]]	[A-Z]	英文大寫字母 upper-case letters
[[:alpha:]]	[A-Za-z]	英文大小寫字母 alphabetic characters
[[:digit:]]	[0-9]	數字字元 (Digits)
[[:alnum:]]	[A-Za-z0-9]	英文字母 + 數字 alphanumeric characters
[[:punct:]]	require \	標點符號 punctuation characters
		! " # $ % & ' ( ) * + , -
		. / : ; < = > ? @
		[\ ] ^ _ ‘ { | } ~
[[:space:]]	"[\t\n\r\f\v]"	space, newline vertical tab, ...
		form feed, carriage return,
[[:graph:]]	[[:alnum:][:punct:]]	graphical characters
[[:print:]]	[[:alnum:][:punct:][:space:]]	printable characters
[[:xdigit:]]	[0-9A-Fa-f]	hexadecimal digits

表 8.2: 常見萬用字元設定 II

萬用字元	說明
^	搜尋 1 行字串的第 1 個字元 "^m"
	搜尋 1 行字串的第 1 個字元不是特定字元 "[^m]"
?	搜尋配對同一元素內字串最多 1 次
⋆	搜尋配對同一元素內字串 0 次 或 無限多次
+	搜尋配對同一元素內字串至少 1 次以上
{n}	搜尋配對同一元素內字串最多 n 次
{n,}	搜尋配對同一元素內字串至少 n 次
{n, m}	搜尋配對同一元素內字串至少 n 次, 最多 m 次
[anyone]	character class, 中括號包圍的字串內的任一個單字符合 "[[]"
	[nco] any string containing any of the characters: "n", "c", "o"
(group)	小包圍的字串群組 "([Tt]om){2,}" "[(]" or "[)]"
|	搜尋特定字合併使用 "Tomas|Luis", "[|]"
{	不適用在 R 的 regular expressions "[{]"

8.6 文字分割函式: strsplit()

文字分割函式 strsplit() 可以根據字串內特定字元或文字將長字串分割成小字串.

```
1 strsplit(x, split, fixed = FALSE, perl = FALSE, useBytes = FALSE)
```

其中引數

- x, text: 分割前的字串向量或物件.
- split: 分割依據的特定字元, 可以是特定型式字串 regular expression. split = character(0) 與 split = "" 是相同意義.
- fixed = FALSE: 設定特定型式字串是否須完全相符合.
- perl = FALSE: 使用 perl 程式語言的特定型式字串.

- useBytes = FALSE: 設定搜尋取代爲 character-by-character 而非 byte-by-byte.

若分割依據的特定字元在字串的第 1 位置 或 最後位置, 則分割後會產生第一個 或 最後位置 元素爲 "".

```
> ## strsplit()
> x.chr <- "asfef"
> strsplit(x.chr, "e") # # split x on the letter e
[[1]]
[1] "asf" "f"
> #
> x.chr <- "Display a scentence with spaces"
> strsplit(x.chr, NULL)
[[1]]
 [1] "D" "i" "s" "p" "l" "a" "y" " " "a" " " "s" "c" "e" "n" "t" "e" "n" "c" "e"
[20] " " "w" "i" "t" "h" " " "s" "p" "a" "c" "e" "s"

> noquote(strsplit(x.chr, NULL)[[1]])
 [1] D i s p l a y   a   s c e n t e n c e   w i t h   s p a c e s
> #
> x.chr <- c(as = "asfef", qu = "qwerty", "yuiop[", "b", "stuff.blah.yech")
> strsplit(x.chr, "e") # # split x on the letter e
$as
[1] "asf" "f"

$qu
[1] "qw"  "rty"

[[3]]
[1] "yuiop["

[[4]]
[1] "b"

[[5]]
[1] "stuff.blah.y" "ch"

> #
> x.chr <- "a.b.c"
> strsplit(x.chr, ".")
[[1]]
[1] "" "" "" "" ""

> unlist(strsplit(x.chr, "."))
[1] "" "" "" "" ""
> strsplit(x.chr, "[.]")
[[1]]
[1] "a" "b" "c"

```

```
> unlist(strsplit(x.chr, "[.]"))
[1] "a" "b" "c"
> unlist(strsplit(x.chr, ".", fixed = TRUE))
[1] "a" "b" "c"
> #
> x.chr <- paste(c("", "a", ""), collapse = "#")
> x.chr
[1] "#a#"
> strsplit(x.chr, split = "#")
[[1]]
[1] ""  "a"

> strsplit(x.chr, split = "#")[[1]]
[1] ""  "a"
> #
> strsplit("", " ")[[1]]    # character(0)
character(0)
> strsplit(" ", " ")[[1]]   # [1] ""
[1] ""
```