Clase 5 (Beta)

Alexis Zúñiga & Ernesto Barrios

Capítulo 13: Manipulación de datos

En el capítulos anteriores ya habíamos visto lo que son las estructuras de datos y su exploración, y los data frames y sus operadores lógicos. Ahora lo que vamos a hacer es entrar en la parte de Manipulación de datos con todos los conceptos que ya habíamos visto relacionados a esta parte.

Para ello utilizaremos el arreglo mtcars. Recordemos un poco cómo es que se ve:

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Para obtener columnas individuales se utiliza el operador $ seguido del nombre de la columna. Si queremos que la consola nos regrese la columna 'mpg' escribimos:

```
mtcars$mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

Esto nos devuelve el vector correspondiente a la columna 'mpg'. Al tratarse de un vector, podemos aplicar todas las operaciones y técnicas conocidas. Si quisiéramos acceder al segundo elemento, escribiríamos:

```
mtcars$mpg[2]
```

```
## [1] 21
```

A continuación, crearemos una nueva columna. Utilizamos la notación $ con un nombre de columna inexistente y el operador de asignación <- para asignar valores a esta nueva columna:

```
mtcars$like <- rep(0, nrow(mtcars))
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb like
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4    0
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4    0
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1    0
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1    0
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2    0
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1    0
```

En este ejemplo, usamos la función rep() para repetir el valor 0 tantas veces como filas tenga mtcars. Sin embargo, en R, también podemos asignar un único valor que se reciclará a lo largo de toda la columna:

```
mtcars$like <- 0
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb like
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4    0
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4    0
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1    0
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1    0
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2    0
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1    0
```

Para modificar valores específicos de esta columna, seguimos el mismo procedimiento utilizado para vectores. Seleccionamos el elemento deseado y le asignamos un nuevo valor:

```
mtcars$like[18] <- 1
mtcars$like[12] <- 1
mtcars$like[2] <- 1
mtcars$like[28] <- 1
mtcars$like[20] <- 1
mtcars$like[21] <- 1
mtcars$like
```

```
##  [1] 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0
```

Dado que la columna es un vector, podemos aplicar las mismas operaciones que conocemos para vectores:

```
sum(mtcars$like)
```

```
## [1] 6
```

```
max(mtcars$cyl)
```

```
## [1] 8
```

La primera línea suma los valores de la columna 'like', mientras que la segunda encuentra el valor máximo en la columna 'cyl'.

Para aplicar condiciones lógicas y filtrar datos, podemos ejecutar:

```
mtcars$cyl >=8
```

```
##  [1] FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE
## [13]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
## [25]  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
```

Esto devuelve un vector booleano indicando si cada elemento de mtcars$cyl cumple con la condición. Podemos usar este vector booleano para filtrar filas:

```
mtcars[mtcars$cyl >= 8, ]
```

```
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb like
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2    0
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4    0
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3    1
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3    0
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3    0
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4    0
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4    0
```

```
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4    0
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2    0
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2    0
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4    0
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2    0
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4    0
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8    0
```

Para seleccionar columnas específicas junto con esta condición, usamos un vector de índices o nombres de columnas:

```
#Un vector de índices columnas
mtcars[mtcars$cyl >=8 & mtcars$disp > 400, c(1,4,5)]
```

```
##                      mpg  hp drat
## Cadillac Fleetwood  10.4 205 2.93
## Lincoln Continental 10.4 215 3.00
## Chrysler Imperial   14.7 230 3.23
```

```
#Un rango de índices columnas
mtcars[mtcars$cyl >=8 & mtcars$disp > 400, 2:5]
```

```
##                     cyl disp  hp drat
## Cadillac Fleetwood    8  472 205 2.93
## Lincoln Continental   8  460 215 3.00
## Chrysler Imperial     8  440 230 3.23
```

```
#Un vector con nombres de columnas
mtcars[mtcars$cyl >=8 & mtcars$disp > 400, c('mpg','cyl', 'disp')]
```

```
##                      mpg cyl disp
## Cadillac Fleetwood  10.4   8  472
## Lincoln Continental 10.4   8  460
## Chrysler Imperial   14.7   8  440
```

Para seleccionar una sola columna, utilizamos el operador $ después de los corchetes:

```
mtcars[mtcars$cyl >=8 & mtcars$disp > 400,]$mpg
```

```
## [1] 10.4 10.4 14.7
```

Si queremos asignar estos resultados a una nueva variable, lo hacemos de la siguiente manera:

```
cars_4_cyl <- mtcars[mtcars$cyl == 4, ]
head(cars_4_cyl)
```

```
##                 mpg cyl  disp hp drat    wt  qsec vs am gear carb like
## Datsun 710     22.8   4 108.0 93 3.85 2.320 18.61  1  1    4    1    0
## Merc 240D      24.4   4 146.7 62 3.69 3.190 20.00  1  0    4    2    0
## Merc 230       22.8   4 140.8 95 3.92 3.150 22.90  1  0    4    2    0
## Fiat 128       32.4   4  78.7 66 4.08 2.200 19.47  1  1    4    1    1
## Honda Civic    30.4   4  75.7 52 4.93 1.615 18.52  1  1    4    2    0
## Toyota Corolla 33.9   4  71.1 65 4.22 1.835 19.90  1  1    4    1    1
```

Hasta ahora, estas operaciones no han modificado el data frame original mtcars. Las modificaciones ocurren sólo cuando hacemos asignaciones.

Ahora, usaremos nuestro conocimiento sobre la creación de columnas y números pseudoaleatorios para agregar una columna tank, que indicará el tamaño del tanque de gasolina de los coches:

```r
set.seed(13)
cars_4_cyl$tank <- round(rnorm(nrow(cars_4_cyl), 18, 5))
cars_4_cyl
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb like tank
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1    0   21
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2    0   17
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2    0   27
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1    1   19
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2    0   24
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1    1   20
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1    1   24
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1    0   19
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2    0   16
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2    1   24
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2    0   13
```

Aquí, rnorm() genera números con una media de 18 y una desviación estándar de 5, y round() redondea estos números. La cantidad de números generados corresponde al número de filas en cars_4_cyl.

Podemos ver un resumen de esta nueva columna:

```r
summary(cars_4_cyl$tank)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   13.00   18.00   20.00   20.36   24.00   27.00
```

Como las columnas de un data frame son vectores del mismo tamaño, podemos realizar operaciones entre ellas. Para calcular la distancia máxima de cada coche, multiplicamos la columna mpg por la columna tank:

```r
cars_4_cyl$distancia_maxima <- cars_4_cyl$mpg * cars_4_cyl$tank
summary(cars_4_cyl$distancia_maxima)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   278.2   447.4   518.7   544.6   646.8   729.6
```

Finalmente, podemos filtrar el data frame con múltiples condiciones. Por ejemplo, para encontrar filas donde mpg sea mayor a 30 y la distancia_maxima sea menor a 400:

```r
cars_4_cyl[cars_4_cyl$mpg > 30 & cars_4_cyl$distancia_maxima < 400, ]
```

```
##  [1] mpg              cyl              disp             hp
##  [5] drat             wt               qsec             vs
##  [9] am               gear             carb             like
## [13] tank             distancia_maxima
## <0 rows> (or 0-length row.names)
```

Esto devolverá un data frame sin filas, ya que ninguna cumple ambas condiciones.

Para asignar los nombres de marca y modelo a una nueva columna:

```r
cars_4_cyl$marca_modelo <- rownames(cars_4_cyl)
head(cars_4_cyl)
```

```
##                 mpg cyl  disp hp drat    wt  qsec vs am gear carb like tank
## Datsun 710     22.8   4 108.0 93 3.85 2.320 18.61  1  1    4    1    0   21
## Merc 240D      24.4   4 146.7 62 3.69 3.190 20.00  1  0    4    2    0   17
## Merc 230       22.8   4 140.8 95 3.92 3.150 22.90  1  0    4    2    0   27
## Fiat 128       32.4   4  78.7 66 4.08 2.200 19.47  1  1    4    1    1   19
## Honda Civic    30.4   4  75.7 52 4.93 1.615 18.52  1  1    4    2    0   24
```

```
## Toyota Corolla 33.9   4  71.1 65 4.22 1.835 19.90  1  1    4    1    1   20
##                distancia_maxima   marca_modelo
## Datsun 710                478.8     Datsun 710
## Merc 240D                 414.8      Merc 240D
## Merc 230                  615.6       Merc 230
## Fiat 128                  615.6       Fiat 128
## Honda Civic               729.6    Honda Civic
## Toyota Corolla            678.0 Toyota Corolla
```

Otra forma de ver el data frame completo es con la función `View()`, que abre una ventana separada para explorar los datos:

```
View(cars_4_cyl)
```

Ahora podemos ver que nuestra nueva columna con los nombres de marca y modelo está incluida en el data frame.

Capítulo 14: Manipulación de texto

La manipulación de texto en R es una tarea común y se puede realizar utilizando diversas funciones y paquetes. Vamos a ver algunos ejemplos de cómo manipular texto utilizando el conjunto de datos mtcars.

Convertir nombre de filas a columnas

Primero, podemos convertir los nombres de las filas de mtcars a una columna para facilitar la manipulación del texto:

```
mtcars$car_name <- rownames(mtcars)
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb like
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4    0
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4    1
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1    0
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1    0
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2    0
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1    0
##                             car_name
## Mazda RX4                   Mazda RX4
## Mazda RX4 Wag           Mazda RX4 Wag
## Datsun 710                 Datsun 710
## Hornet 4 Drive         Hornet 4 Drive
## Hornet Sportabout Hornet Sportabout
## Valiant                       Valiant
```

Manipulaciones básicas de texto

Convertir mayúsculas a minúsculas

Podemos convertir los nombres de coches a mayúsculas o minúsculas usando `toupper()`o `tolower()`

```
mtcars$car_name_upper <- toupper(mtcars$car_name)
mtcars$car_name_lower <- tolower(mtcars$car_name)
head(mtcars[, c("car_name", "car_name_upper", "car_name_lower")])
```

```
##                             car_name    car_name_upper    car_name_lower
## Mazda RX4                   Mazda RX4         MAZDA RX4         mazda rx4
## Mazda RX4 Wag           Mazda RX4 Wag     MAZDA RX4 WAG     mazda rx4 wag
## Datsun 710                 Datsun 710        DATSUN 710        datsun 710
## Hornet 4 Drive         Hornet 4 Drive    HORNET 4 DRIVE    hornet 4 drive
## Hornet Sportabout Hornet Sportabout HORNET SPORTABOUT hornet sportabout
## Valiant                       Valiant           VALIANT           valiant
```

Susitituit texto

Usamos `gsub()` para reemplazar partes de los nombres de los coches. Por ejemplo, podemos reemplazar "Mazda" por "Toyota":

```
mtcars$car_name_replaced <- gsub("Mazda", "Toyota", mtcars$car_name)
head(mtcars[, c("car_name", "car_name_replaced")])
```

```
##                             car_name car_name_replaced
## Mazda RX4                   Mazda RX4        Toyota RX4
## Mazda RX4 Wag           Mazda RX4 Wag    Toyota RX4 Wag
```

```
## Datsun 710               Datsun 710        Datsun 710
## Hornet 4 Drive       Hornet 4 Drive    Hornet 4 Drive
## Hornet Sportabout Hornet Sportabout Hornet Sportabout
## Valiant                     Valiant           Valiant
```

Dvidir y unir texto

Podemos dividir los nomnbres de los coches en palabras individuales utilizando `strsplit()` y unirls d nuevo con `paste()`.

```
# Dividir nombres en palabras
split_names <- strsplit(mtcars$car_name, " ")
split_names[1:5]
```

```
## [[1]]
## [1] "Mazda" "RX4"
##
## [[2]]
## [1] "Mazda" "RX4"   "Wag"
##
## [[3]]
## [1] "Datsun" "710"
##
## [[4]]
## [1] "Hornet" "4"      "Drive"
##
## [[5]]
## [1] "Hornet"     "Sportabout"
```

```
# Unir palabras con un guion
mtcars$car_name_hyphen <- sapply(split_names, function(x) paste(x, collapse = "-"))
head(mtcars[, c("car_name", "car_name_hyphen")])
```

```
##                           car_name   car_name_hyphen
## Mazda RX4                 Mazda RX4         Mazda-RX4
## Mazda RX4 Wag         Mazda RX4 Wag     Mazda-RX4-Wag
## Datsun 710               Datsun 710        Datsun-710
## Hornet 4 Drive       Hornet 4 Drive    Hornet-4-Drive
## Hornet Sportabout Hornet Sportabout Hornet-Sportabout
## Valiant                     Valiant           Valiant
```

Extraer subcadenas

Podemos extraer ciertas partes de los nombres usando substr(). Por ejemplo, extraer los primeros 5 caracteres de cada nombre de coche:

```
mtcars$car_name_substr <- substr(mtcars$car_name, 1, 5)
head(mtcars[, c("car_name", "car_name_substr")])
```

```
##                           car_name car_name_substr
## Mazda RX4                 Mazda RX4           Mazda
## Mazda RX4 Wag         Mazda RX4 Wag           Mazda
## Datsun 710               Datsun 710           Datsu
## Hornet 4 Drive       Hornet 4 Drive           Horne
## Hornet Sportabout Hornet Sportabout           Horne
## Valiant                     Valiant           Valia
```

Veremos ahora una serie de funciones para manipular texto (o vectores de texto). Estas son especialmente útiles para la limpieza de columnas de datos. La primera que analizaremos es `grepl()`.

Esta sirve para buscar un patrón de caracteres en un vector. Usemos la columna de marca y modelo como vector ejemplo. La sintaxis es `gsub(patrón, vector)`

```
grepl('Fiat', cars_4_cyl$marca_modelo)
```

```
##  [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

Obtenemos un vector booleano que por sí solo no nos es muy útil. Sin embargo, este se puede escribir dentro de los corchetes de indexación de un arreglo para obtener un resultado más útil.

```
cars_4_cyl[grepl('Fiat', cars_4_cyl$marca_modelo), ]
```

```
##            mpg cyl disp hp drat    wt  qsec vs am gear carb like tank
## Fiat 128  32.4   4 78.7 66 4.08 2.200 19.47  1  1    4    1    1   19
## Fiat X1-9 27.3   4 79.0 66 4.08 1.935 18.90  1  1    4    1    0   19
##           distancia_maxima marca_modelo
## Fiat 128             615.6     Fiat 128
## Fiat X1-9            518.7    Fiat X1-9
```

Ahora vemos que la función nos es útil para buscar datos específicos dentro de una cadena en un arreglo, no solo el dato completo de la columna (Esto se lograría con arreglo[dato == buscado]).

Vectorización de funciones

Para resolver el problema anterior primero tenemos que definir una función que haga lo que queremos. Ya analizamos el problema en el paso anterior entonces lo podemos aplicar de manera casi idéntica. Con la única excepción de que ahora lo pensamos como si la función recibiera un solo elemento a la vez.

```
extrae_apellido <- function(nombre_completo){
  strsplit(nombre_completo, " ")[[1]][2]
}
```

Verificamos que funcione como queremos:

```
extrae_apellido("Juan Pérez")
```

```
## [1] "Pérez"
```

Ahora utilizaremos la familia de funciones `apply()` para aplicar esta función a todos los elementos de un vector. Primero analizaremos `sapply()` y `lapply()`. La sintaxis para estas es: `apply(vector, función)`

```
nombre_apellidos <- c("Alexis Zúñiga", "Juan Pérez", "José Hernández")

sapply(nombre_apellidos, extrae_apellido) #La función se escribe sin paréntesis
```

```
##  Alexis Zúñiga     Juan Pérez José Hernández
##       "Zúñiga"        "Pérez"    "Hernández"
```

```
lapply(nombre_apellidos, extrae_apellido)
```

```
## [[1]]
## [1] "Zúñiga"
##
## [[2]]
## [1] "Pérez"
##
## [[3]]
## [1] "Hernández"
```

La función `sapply()` nos regresa un vector con los resultados que además tiene nombres y estos son los elementos del vector original. Por otro lado, `lapply()` regresa una lista donde cada resultado está en su propio vector.

Declaremos ahora una función más compleja y por lo tanto más útil que la que tenemos actualmente. Esta nos va a permitir indicarle qué caracter usar para el separador y también qué parte del nombre completo queremos extraer. Nótese que vamos a establecer valores predeterminados para estos parámetros.

```
extrae_de_nombre <- function(nombre_completo, separador = " ", posicion = 2){ strsplit(nombre_completo,
}
# Usando los valores default
extrae_de_nombre("Elba Laso")
```

```
## [1] "Laso"
```

```
# Si los especificamos
extrae_de_nombre("Pepe Roni", " ", 2)
```

```
## [1] "Roni"
```

```
# Extraer el nombre
extrae_de_nombre("Alan Brito ", posicion =1)
```

```
## [1] "Alan"
```

```
# Con diferente separador
extrae_de_nombre("Aitor-Tilla", separador = "-")
```

```
## [1] "Tilla"
```

Para usar parámetros con las funciones `lapply()` o `sapply()` se usa la siguiente sintaxis: `apply(vector, función, parámetro = valor)`.

```
sapply(nombre_apellidos, extrae_de_nombre, posicion = 1)
```

```
##  Alexis Zúñiga     Juan Pérez José Hernández
##       "Alexis"         "Juan"         "José"
```

Sin embargo estas funciones solo reciben un parámetro por función. En este caso funciona porque nuestra función tiene valores default para los argumentos que no especificamos. Sin embargho, si queremos declarar todos individualmente, es necesario utilizar la función `mapply()`. Funciona igual que las demás funciones de la familia apply pero en este caso recibe argumentos infinitos y por lo tanto cambia la sintaxis: `mapply(función, arg1,arg2,... ,vector)`

```
nombres2 <- c("Ernesto-J-Barrios", "Alexis-X-Zúñiga", "Ana-L-Melo")
mapply(extrae_de_nombre, separador = "-", posicion = 3, nombres2)
```

```
##         -      <NA>      <NA>
## "Barrios"  "Zúñiga"    "Melo"
```