Manipulación de Datos

Santiago Casanova y Ernesto Barrios

Manipulación de datos

En notas anteriores vimos una introducción a los arreglos o `data.frames`, comparadores lógicos y operaciones con vectores. Todos estos conceptos ahora nos serán útiles para aprender a manipular los datos que tenemos almacenados.

Recordemos cómo se ve el arreglo `mtcars`

```
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
```

Una forma de obtener columnas individuales es utilizando 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
```

Y el resultado es el vector que forma la columna `mpg`. Al ser un vector le podemos aplicar todas las tecnicas y operaciones que ta conocemos para los vectores. Por ejemplo, si quisiéramos obtener el dato en la pocisión dos escribimos:

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

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

Ahora vamos a crear nuestra propia columna. Para hacer esto, usamos la notación del operador `$` pero ahora con un nombre de columna que no exista. Después usamos el operador de asignación `<-` para asignarle algo a dicha 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 caso utilizamos la función `rep()` para repetir el cero n veces donde n es el número de filas que tiene el arreglo `mtcars`. Sin embargo, `R` es un lenguaje con muchas comodidades y podemos asignar solo un cero y automáticamente lo recicla a lo largo de 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
```

Ahora nos gustaría cambiar algunos valores de esta columna. Para hacer esto seguimos exáctamente el mismo proceso que usamos para modificar vectores. Seleccionamos el elemento que queremos y le asignamos un valor nuevo.

```
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 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0
```

De la misma forma, al ser un vector, podemos usar todas las técnicas y operaciones que conocemos que fincionan para vectores. Por ejemplo:

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

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

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

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

La primera nos regresa la suma de la columna `like`. Es facil ver que nos regresará 6 ya que en la sección anterior le asignamos 6 `1` en diferentes pocisiones. La segunda nos regresa el valor máximo de la columna `cyl`.

Ahora vamos a analizar cómo podemos utilizar pruebas lógicas para obtener valores de un arreglo. Si corremos la expresión:

```
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
```

vemos que nos regresa un vector booleano con valores `TRUE` o `FALSE` dependiendo si los elementos del vector escogido `mtcars$cyl` cumplen la condición. Ahora loq eu podemos hacer es pasar este vector lógico como argumento dentro de los corchetes del arreglo. Esto nos dará las filas que tengan `TRUE` en nuestra prueba lógica.

```
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
```

Ponemos la prueba lógica seguida de una coma porque queremos obtener los renglones que cumplan esta condición, como lo vimos en la sección de arreglos de la nota anterior.

Si quisieramos que nos regrese estas filas pero sólamente una selección de columnas, podemos usar un vector con los índices (o los nombres) de las columnas deseadas después de la coma.

En estos próximos ejemplos agregaremos otra condición para limitar los reultados. Ahora buscamos todas las filas que cumplan que `cyl` sea mayor o igual a 8 y que `disp` sea mayor a 400.

```
#Un vector de indices 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 indices 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 vecotr 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
```

Si sólo bucamos una sola columna, también se puede utilizar el operador `$` después de los corchetes para indicar que queremos que nos regrese esa columna. Nótese que aún es necesario escribir la coma.

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

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

De igual manera podemos notar que cuando seleccionamos más de una columna la consola nos regresa un arreglo, mientras que cuando sólo seleccionamos una columna (ya sea con índice, nombre o el operador `$`) la consola regresa un vector.

Esto es crucial ya que nos permite aplicar todas las operaciones y manipulaciones de vectores que ya conocemos.

Esta sintáxis no solo sirve para obtener los datos a través de la consola. Naturalmente también podemos asignar estos resultados a una nueva variable. Vamos a crear un *subset* de `mtcars` que sólo incluya las filas con `cyl` igual a 4.

```
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
```

Nótese que si no le asignaramos nuestro *subset* a la variable `cars_4_cyl`, el arreglo original no se vería modificado.

Ahora usemos lo que sabemos sobre crear columnas y números pseudo-aleatorios para crear una columna `tank` que indique el tamaño del tanque de gasolina de los coches.

```
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
```

Estamos creando la columna `tank` con números enteros (gracias a `round()`) con media 20 y desviación estandar 8 (con la función `rnorm()`). Para el cantidad de números aleatorios a generar utilizamos `nrow()` para que la función nos regrese los suficientes para todas las filas de nuestro arreglo.

Veamosel resumen nuestra nueva columna.

```
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
```

Las columnas de un arreglo son vectores del mismo tamaño por lo que podemos hacer operaciones entrada-a-entrada entre ellos. Si queremos calcular la distancia total de cada coche nos absta con multiplicar la columna `mpg` (*miles per gallon*) por nuestra nueva columna `tank` en galones.

```
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
```

Ahora filtremos nuestro arreglo con varias condiciones. Queremos todas las columnas de las filas que cumplan que `mpg` sea mayor a 30 y la distancia máxima sea menor a 400.

```
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 nos regresa un arreglo de 11 columnas sin embargo tiene cero filas. Ninguna cumple las condiciones que le pedimos.

A lo largo de esta sección hemos visto las marcas y modelos de los coches a un lado del arreglo, sin embargo no son parte de una columna. Si queremos asignarlas a una columna propia podemos hacer:

```
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 un arreglo competo es con la función `View()`. Esta en vez de regresar algo a la consola, abre el arreglo en otra pestaña donde lo podemos filtrar y buscar datos individuales a manera de interfaz gráfica.

```
View(cars_4_cyl)
```

Podemos ver que nuestra nueva columna de marca y modelo está ya incluida.

Maipulación de texto

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 sintáxis 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]).

También es necesario en muchas ocasiones remplazar un patrón por otro en un vector de cadenas. Para esto usaos la función `gsub()`. Funciona de manera similar a `grepl()` pero ahora también recibe el parámetro `remplazo`. Entonces la sintáxis queda `gsub(patrón, remplazo, vector)`.

```
gsub('Fiat', 'Mazda',  cars_4_cyl$marca_modelo)
```

```
##  [1] "Datsun 710"     "Merc 240D"      "Merc 230"       "Mazda 128"
##  [5] "Honda Civic"    "Toyota Corolla" "Toyota Corona"  "Mazda X1-9"
##  [9] "Porsche 914-2"  "Lotus Europa"   "Volvo 142E"
```

En este caso la salida es un vector de cadenas en lugar de uno booleano. Este vector es la versión modificada de nuestro vector original.

En muchos casos es necesario separar una cadena (más adelante lo veremos para columnas) en dos o mas secciones. Para eso podemos usar la función `strsplit()`. Como lo indica su nombre, separa cadenas. Se usa la siguiente sintáxis: `strsplit(cadena, separador)`, donde el separador es un caracter que indique una separación.

```
strsplit("palabra1 palabra2", " ")
```

```
## [[1]]
## [1] "palabra1" "palabra2"
```

En este caso queremos separar nuestra cadena en el espacio por lo que escribimos un espacio como parámetro de separador. Nótese que en este caso la salida no es un vector sino una lista. Esto es porque el resultado de una sola separación es inherentemente un vector. Si quisiéramos aplicar la función a todo un vector, necesitamos que la función regrese una "colección de vectores". En este caso es una lista de vectores.

```
strsplit(c("palabra1 palabra2", "palabra3 palabra4"), " ")
```

```
## [[1]]
## [1] "palabra1" "palabra2"
##
## [[2]]
## [1] "palabra3" "palabra4"
```

Este ejemplo ilustra por qué es necesario que el resultado sea una lista. Sin embargo, esto presenta otros problemas. Supongamos que tenemos una columna que incluye nombres y apellidos y la queremos separar en una columna sólo de nombres y otra solo de apellidos. ¿Cómo logramos esto?

Probemos varias opciones para obtener solo el apellido del siguiente vector.

```
nombre_apellidos <- c("Ana Perez", "Ernesto Barrios", "Santiago Casanova")

# strsplit original
strsplit(nombre_apellidos, " ")
```

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

```
## [1] "Ana"   "Perez"
##
## [[2]]
## [1] "Ernesto" "Barrios"
##
## [[3]]
## [1] "Santiago" "Casanova"
```

```
# tomamos la segunda pocision
strsplit(nombre_apellidos, " ")[2]
```

```
## [[1]]
## [1] "Ernesto" "Barrios"
```

```
# tomamos el segundo elemento de la lista
strsplit(nombre_apellidos, " ")[[2]]
```

```
## [1] "Ernesto" "Barrios"
```

Ninguna de estas opciones cumple lo que queremos. No hay manera de obtener solo el segundo resultado sólo con indices por lo que tendremos que explorar mas opciones. Cómo primer instinto podemos pensar en usar un `for()` para lograr esto.

```
# Declaramos un vector vacío
apellidos <- c()

for(i in 1:length(nombre_apellidos)){
  apellidos[i] <- strsplit(nombre_apellidos[i], ' ')[[1]][2]
}
```

En este ejemplo estamos tomando el elemento **i** del vector (desde 1 hasta el largo del vector), separandolo por el espacio y, una vez separado, tomando el primer elemento de la lista resultante (`[[1]]`). Ya que tenemos acceso al vector tomamos el segundo elemento. Ese elemento es asignado a la pocisión **i** de nuestro vector apellidos.

Mientras que sí es una solución viable no es muy eficiente y se presta a errores. En esta sección vamos a aprender a vectorizar operaciones.

Vectorización de Funciones

Para resolver el problema anterior priemro 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("Enrique Juarez")
```

```
## [1] "Juarez"
```

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

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

```
##         Ana Perez   Ernesto Barrios Santiago Casanova
```

```
##          "Perez"         "Barrios"        "Casanova"
```

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

```
## [[1]]
## [1] "Perez"
##
## [[2]]
## [1] "Barrios"
##
## [[3]]
## [1] "Casanova"
```

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 util 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, separador)[[1]][posicion]
}

# Usando los valores default
extrae_de_nombre("Ana López")
```

```
## [1] "López"
```

```
# Si los especificamos
extrae_de_nombre("Ana López", " ", 2)
```

```
## [1] "López"
```

```
# Extraer el nombre
extrae_de_nombre("Ana López", posicion =2)
```

```
## [1] "López"
```

```
# Con diferente separador
extrae_de_nombre("Ana-López", separador = "-")
```

```
## [1] "López"
```

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

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

```
##         Ana Perez   Ernesto Barrios Santiago Casanova
##             "Ana"         "Ernesto"        "Santiago"
```

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

```r
nombres2 <- c("Ernesto-J-Barrios", "Santiago-J-Casanova", "Ana-P-Lopez")

mapply(extrae_de_nombre, separador = "-", posicion = 3, nombres2)
```

```
##          -       <NA>       <NA>
##  "Barrios" "Casanova"    "Lopez"
```