Análisis y Tratamiento de Datos con R

Con ejemplos e ilustraciones

Primera Edición

Diego Paul Huaraca S.
MS-PLUS, Inc.

MS+
Soluciones
Matemáticas

Un aporte de Source Stat Lab Ecuador a la sociedad.

Índice general

1

Manipulación de datos

1.1. Operadores de encadenamiento

El desarrollo de los operadores de encadenamiento inicia el 17 de Enero de 2012 a partir de la inquietud colocada por el usuario anónimo *user4* en el sitio web `Stack Overflow`[1], la misma trataba de averiguar la posible implementación de los operadores del lenguaje *F#*[2] (F Sharp) en R.

Ben Bolker respondío el mismo día dando lo que podríamos considerar el primer operador en R:

```
"%>%" <- function(x,f) do.call(f,list(x))
16 %>% sqrt

## [1] 4
```

Para Octubre de 2013 aparece el primer operador de encadenamiento como parte del paquete `dplyr` desarrollado por Hadley Wickham. Este operador fue denominado `chain` (%.%), la idea detrás de la introducción del operador fue simplificar la notación con el fin de aplicar varias funciones al mismo tiempo a un `data.frame`.

Instalamos y cargamos el paquete `dplyr` para trabajar en los ejericios siguientes:

```
install.packages('dplyr', dependencies = TRUE)
library(dplyr)
```

A continuación, mostramos un ejemplo de rutina común dentro del análisis de datos con y sin el uso del operador de encadenamiento:

Rutina con encadenamiento

```
mtcars %.% group_by(carb, cyl) %.%
   select(mpg, disp, hp) %.%
   summarise(
      mean_mpg = mean(mpg, na.rm = TRUE),
```

[1] Sitio web desarrollado por Jeff Attwood muy utilizado por una comunidad de desarrolladores informáticos, en la cual se pueden encontrar soluciones a problemas de programación en diferentes lenguajes.

[2] Lenguaje de programación multiparadigma de código abierto para la plataforma .NET

```
      mean_disp = mean(disp, na.rm = TRUE),
      mean_hp = mean(hp, na.rm = TRUE)
   )

## Source: local data frame [9 x 5]
## Groups: carb
##
##   carb cyl mean_mpg mean_disp mean_hp
## 1    1   4    27.58     91.38    77.4
## 2    1   6    19.75    241.50   107.5
## 3    2   4    25.90    116.60    87.0
## 4    2   8    17.15    345.50   162.5
## 5    3   8    16.30    275.80   180.0
## 6    4   6    19.75    163.80   116.5
## 7    4   8    13.15    405.50   234.0
## 8    6   6    19.70    145.00   175.0
## 9    8   8    15.00    301.00   335.0
```

Rutina sin encadenamiento

```
summarise(select(group_by(mtcars, carb, cyl), mpg, disp, hp),
          mean_mpg = mean(mpg, na.rm = TRUE),
          mean_disp = mean(disp, na.rm = TRUE),
          mean_hp = mean(hp, na.rm = TRUE))

## Source: local data frame [9 x 5]
## Groups: carb
##
##   carb cyl mean_mpg mean_disp mean_hp
## 1    1   4    27.58     91.38    77.4
## 2    1   6    19.75    241.50   107.5
## 3    2   4    25.90    116.60    87.0
## 4    2   8    17.15    345.50   162.5
## 5    3   8    16.30    275.80   180.0
## 6    4   6    19.75    163.80   116.5
## 7    4   8    13.15    405.50   234.0
## 8    6   6    19.70    145.00   175.0
## 9    8   8    15.00    301.00   335.0
```

En Diciembre de 2013, Stefan Bache propone una respuesta alternativa original para la inquietud colocada en `Stack Overflow`.

```
`%>%` <- function(e1, e2){
   cl <- match.call()
   e  <- do.call(substitute, list(cl[[3]], list(. = cl[[2]])))
   eval(e)
}
```

Un ejemplo rápido del uso para el operador propuesto por Bache es el siguiente:

```
mtcars %>%
  subset(., cyl == 8, select = -vs) %>%
  colMeans(.)
```

```
##         mpg         cyl        disp          hp        drat          wt
##  15.1000000   8.0000000 353.1000000 209.2142857   3.2292857   3.9992143
##        qsec          am        gear        carb
##  16.7721429   0.1428571   3.2857143   3.5000000
```

Stefan continuó trabajando con la finalidad de mejorar el funcionamiento de su operador de encadenamiento al punto que implementó el paquete `magrittr` que incluída al operador $\%>\%$.

El paquete `dplyr` estuvo siendo desarrollado en paralelo al trabajo de Bache y ambos contenian un operador de asignación. Finalmente, en Abril de 2014 el paquete `dplyr` incorporó el operador de `magrittr` sustituyendo al operador original de Hadley $\%.\%$ debido que el primero es más fácil de escribir y posee un desarrollo más minucioso.

Desde Agosto de 2014, RStudio incorporó un acceso directo para el operador $\%>\%$ por medio de la combinación de teclas: Ctrl+Shift+M.

1.2. Exploración de datos

En esta sección analizaremos las funciones implementadas dentro del paquete `dplyr`, las cuales conjuntamente con el operador de asignación simplifican la interpretación del código. El siguiente ejemplo inicia creando un data.frame y lo asociamos al objeto `data`, para posteriormente convertirlo en una estructura tabular más versatil.

```
library(dplyr)
# Fijamos una semilla para que los datos sean replicables
set.seed(123)
# Creamos una data frame
datos <- data.frame(num=seq(1:8),
                    edad=sample(25:70, size=8),
                    segmento=sample(LETTERS[1:3], size=8, replace=TRUE),
                    deudas=round(1000*runif(8),2),
                    sueldo=sample(354:2000, size=8),
                    tarjetas=sample(1:4, size=8, replace=TRUE),
                    hijos=sample(0:5, size=8, replace=TRUE))

# Verificamos la clase del objeto creado
class(datos)
```

```
## [1] "data.frame"
```

```
# Ahora convertimos el data frame inicial en un data frame tabular
datos <- tbl_df(datos)
# Volvemos a verificar la clase del objeto
class(datos)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
# Vizualizamos los datos
datos
```

```
## Source: local data frame [8 x 7]
##
##   num edad segmento deudas sueldo tarjetas hijos
## 1   1   38        B 246.09   1433        3     0
## 2   2   60        B  42.06   1520        4     2
## 3   3   42        C 327.92   1248        1     2
## 4   4   62        B 954.50   1330        2     2
## 5   5   64        C 889.54    829        4     0
## 6   6   26        B 692.80    595        1     0
## 7   7   46        A 640.51   1934        2     1
## 8   8   59        C 994.27   1833        1     2
```

Se recomienda al usuario hacer uso de la función `tbl_df` con la finalidad de convertir un data.frame nativo en una estructura tabular sobre la cual se pueden realizar operaciones de manera más rápida sin perder la esencia principal del objeto orientado al almacenamiento de diversos tipos de vectores.

1.2.1. Selección de variables

El paquete `dplyr` provee la función `select`, la cual se encarga de la selección de variables y mantiene la siguiente estructura:

```
select(datos, variables a seleccionar)
# o su equivalente
datos %>% select(variables a seleccionar)
```

a continuación algunos ejemplos:

```
# Seleccionamos las variables edad y sueldo
select(datos, edad, sueldo)

## Source: local data frame [8 x 2]
##
##   edad sueldo
## 1   38   1433
## 2   60   1520
## 3   42   1248
## 4   62   1330
## 5   64    829
## 6   26    595
## 7   46   1934
## 8   59   1833

# datos %>% select(edad, sueldo)
# Una forma alternativa es colocar las posiciones de las variables
select(datos, 1, 2)

## Source: local data frame [8 x 2]
##
##   num edad
## 1   1   38
## 2   2   60
```

```
## 3   3   42
## 4   4   62
## 5   5   64
## 6   6   26
## 7   7   46
## 8   8   59

# datos %>% select(2, 5)
# Seleccionamos las primeras 4 variables
select(datos, num:deudas)

## Source: local data frame [8 x 4]
##
##   num edad segmento deudas
## 1   1   38        B 246.09
## 2   2   60        B  42.06
## 3   3   42        C 327.92
## 4   4   62        B 954.50
## 5   5   64        C 889.54
## 6   6   26        B 692.80
## 7   7   46        A 640.51
## 8   8   59        C 994.27

# datos %>% select(num:deudas)
# Una forma alternativa es colocar las posiciones de las variables
select(datos, 1:4)

## Source: local data frame [8 x 4]
##
##   num edad segmento deudas
## 1   1   38        B 246.09
## 2   2   60        B  42.06
## 3   3   42        C 327.92
## 4   4   62        B 954.50
## 5   5   64        C 889.54
## 6   6   26        B 692.80
## 7   7   46        A 640.51
## 8   8   59        C 994.27

# datos %>% select(1:4)
# Seleccionamos las variables de posicion par
select(datos, seq(2, ncol(data), by=2))

## Error in seq.default(2, ncol(data), by = 2): 'to'must be of length 1

# datos %>% select(seq(2, ncol(data), by=2))
```

En ocasiones se vuelve un dolor de cabeza seleccionar las variables que se encuentran en diferentes posiciones pero presentan algún tipo de patrón en los nombres de las variables. En este caso el paquete `dplyr` provee funciones que facilitan esta tarea.

```
# Selecionamos las variables que inician con "s"
select(datos, starts_width("s"))

## Error in eval(expr, envir, enclos): could not find function "starts_width"

# Selecionamos las variables que terminan en "s"
select(datos, ends_width("s"))

## Error in eval(expr, envir, enclos): could not find function "ends_width"

# Seleccionamos las variables que contienen en sus nombres "e"
select(datos, contains("s"))

## Source: local data frame [8 x 5]
##
##   segmento deudas sueldo tarjetas hijos
## 1        B 246.09   1433        3     0
## 2        B  42.06   1520        4     2
## 3        C 327.92   1248        1     2
## 4        B 954.50   1330        2     2
## 5        C 889.54    829        4     0
## 6        B 692.80    595        1     0
## 7        A 640.51   1934        2     1
## 8        C 994.27   1833        1     2
```

1.2.2. Creación de nuevas variables

Para la creación de nuevas variables, el paquete `dplyr` provee la función `mutate`, la misma que tiene la siguiente estructura:

```
# Creamos la variable "index" como la suma de las variables "var1" y "var2"
mutate(datos, index=var1+var2)
# o su equivalente
datos %>% mutate(index=var1+var2)
```

Al momento que crear la variable es indispensable colocar el nombre de la nueva variable. Esta nueva variable puede ser creada mediante diversas operaciones aplicadas a las variables de la base inicial.

```
# Creamos una variable que identifique los sujetos con hijos
mutate(datos, dummy=ifelse(hijos==0, "SIN HIJOS", "CON HIJOS"))

## Source: local data frame [8 x 8]
##
##   num edad segmento deudas sueldo tarjetas hijos     dummy
## 1   1   38        B 246.09   1433        3     0 SIN HIJOS
## 2   2   60        B  42.06   1520        4     2 CON HIJOS
## 3   3   42        C 327.92   1248        1     2 CON HIJOS
## 4   4   62        B 954.50   1330        2     2 CON HIJOS
## 5   5   64        C 889.54    829        4     0 SIN HIJOS
## 6   6   26        B 692.80    595        1     0 SIN HIJOS
## 7   7   46        A 640.51   1934        2     1 CON HIJOS
## 8   8   59        C 994.27   1833        1     2 CON HIJOS
```

1.2.3. Filtrado

La función `filter` del paquete `dplyr` nos permite filtrar los registros de una base que cumplan una o más condiciones que se requieran. La estructura de uso es la siguiente:

```
filter(datos, condición)
# o su equivalente
datos %>% filter(condición)
```

A continuación, algunos ejemplos:

```
# Filtramos los registros de sujetos con mas de 2 tarjetas
filter(datos, tarjetas > 2)

## Source: local data frame [3 x 7]
##
##   num edad segmento deudas sueldo tarjetas hijos
## 1   1   38        B 246.09   1433        3     0
## 2   2   60        B  42.06   1520        4     2
## 3   5   64        C 889.54    829        4     0
```

1.2.4. Ordenando las observaciones

1.2.5. Calculando estadísticos