Manejo de datos

David García Callejas

En esta sección repasaremos el manejo de datos de diferentes tipos. La selección, modificación, eliminación, o agregación de datos, principalmente usando dataframes.

Selección

```
eq <- read.csv2(file = "../data/Earthquake_data.csv",
                header = TRUE,
                dec = ".",
                stringsAsFactors = FALSE)

# selección de columnas
eq.clean <- eq[,c("En","Year","Mo","Da","Ho",
                  "Mi","Se","Area","Lat","Lon",
                  "Dep","M")]

# selección de observaciones. P.ej terremotos de más de 7 grados de magnitud
eq.7 <- subset(eq.clean,M > 7)
summary(eq.7$M)

# si en vez de recuperar los datos completos sólo me interesa
# la posición de las observaciones con magnitud >7, uso which
pos.7 <- which(eq.clean$M > 7)

# y si quiero recuperar el dataframe sólo con estas posiciones:
eq.7.2 <- eq.clean[pos.7,]

# la manera de seleccionar con subset y con which
# debería darnos dataframes equivalentes
identical(eq.7,eq.7.2)

# si tenemos filas duplicadas, podemos eliminarlas con la función unique
# en este caso, no existen duplicados,
# así que la función devuelve un dataframe igual al original
eq.7.3 <- unique(eq.7)
```

Hemos visto muy por encima que hay situaciones en las que una variable puede tener valor nulo (NULL). De manera similar, una observación en un dataframe o matriz (es decir, una fila) puede no tener un valor asociado en alguna de las columnas. Por ejemplo, puede haber terremotos de los que no sepamos con seguridad la hora a la que sucedieron.

```
# columna Ho indica la hora, Mi los minutos, Se los segundos
head(eq.clean)
```

Estos valores de los que no tenemos datos se indican con la palabra reservada "NA". Algunas de las operaciones

más comunes en dataframes implican limpiar los NAs de nuestros conjuntos de datos, por ejemplo para realizar análisis estadísticos. Las funciones que hemos visto para seleccionar datos también nos sirven para esto.

```
# dejar sólo observaciones en las que no haya ni un solo NA en ninguna columna
eq.sin.na.vec <- complete.cases(eq.clean)

# fijaos que la función devuelve un vector de valores lógicos,
# es decir, un valor TRUE o FALSE por cada fila de nuestros datos.
# Para seleccionar estos datos, lo hacemos como antes con "which"
eq.sin.na <- eq.clean[eq.sin.na.vec,]
eq.sin.na

# estas dos operaciones se pueden realizar en una sola línea
eq.sin.na <- eq.clean[complete.cases(eq.clean),]

# podemos seleccionar también los valores de una sola columna
# por ejemplo, aquellos terremotos de los que sepamos la profundidad
# estas dos maneras son equivalentes:
# en subset, usamos el nombre de la columna directamente
eq.profundidad <- subset(eq.clean, !is.na(Dep))
# en which, necesitamos especificar dataframe$columna
eq.profundidad.2 <- eq.clean[which(!is.na(eq.clean$Dep)),]

# hemos usado la función is.na, cuyo objetivo es tan simple
# como decirnos qué elementos de un vector son NA.
ejemplo.vector <- c("este", "vector", "tiene", "algún", "NA", NA)

is.na(ejemplo.vector)
which(is.na(ejemplo.vector))
# recordad que la exclamación delante de una condición lógica es
# la negación de esa condición
which(!is.na(ejemplo.vector))
# ¿cuántos NA hay en un vector?
# la función "sum" también funciona con vectores lógicos
sum(is.na(ejemplo.vector))
```

Modificación

Todas las funciones anteriores, en las que seleccionamos filas determinadas de un dataframe, nos pueden servir para modificar los valores que queramos

```
# no es recomendable en general, pero podemos
# sustituir los NAs por otros valores.
# primero copiamos el dataframe original, para evitar pisar datos
eq.sust <- eq.clean

# asignamos 0 a aquellas horas que sean NA
eq.sust$Ho[which(is.na(eq.sust$Ho))] <- 0

# ¿cómo son ahora los datos de hora?
summary(eq.clean$Ho) # originales
summary(eq.sust$Ho) # con 0 en los NAs
```

```r
# también podemos sustituir valores de cualquier otro tipo, no sólo NA
# por ejemplo, el campo "Area" es bastante confuso. Nos puede interesar
# aislar los terremotos en dos regiones. Por ejemplo, Chile por un lado y
# el resto del mundo por otro

head(eq.sust$Area)

eq.sust$Area[which(eq.sust$Area != "[Chile]")] <- "other"
head(eq.sust$Area)
table(eq.sust$Area)
```

Aquí no las trataremos en detalle, pero existen maneras de seleccionar caracteres más sofisticadas, por medio de las denominadas "expresiones regulares", o "regular expressions" (regex) en inglés. Son una serie de reglas que podemos aplicar para seleccionar qué elementos de un vector de caracteres cumplen un determinado patrón, o sustituir unos caracteres por otros.

```r
# podemos sustituir un carácter dentro de los elementos de un vector
# por cualquier otro carácter
# por ejemplo, espacios por guiones
area.orig <- eq.clean$Area

# la función gsub reemplaza un patrón de caracteres por otro
area.guiones <- gsub(pattern = " ", replacement = "_",x = area.orig)
```

Agregación

Es muy común tener conjuntos de datos dispersos en varias tablas. Por ejemplo, podríamos tener los datos de terremotos en diferentes tablas, una de las cuales fueran las características físicas de la zona, otra la descripción del terremoto, otra tabla para daños causados, etc. R nos permite unir información de diferentes tablas, y hay varias formas para ello.

```r
# match es una función muy útil
# nos indica qué valores de un vector se corresponden
# con los valores de un segundo vector
v1 <- c(1,2,3,4,5)
v2 <- c(8,4,9,7,0,1)

pos <- match(v1,v2)
pos

# el primer elemento de v1 está en la posición 6 de v2.
# El segundo y tercero no están (NA), el cuarto
# está en la posición 2 de v2, y el quinto no está.

pos2 <- match(v2,v1)
pos2

# match nos puede servir para unir dataframes en base a una columna común
d1 <- data.frame(caracter = c("a","b","c"), num = c(1,2,3))
d1
d2 <- data.frame(caracter = c("c","d","e"), mas.info = c(6.6,3.4,8.7))
d2

# añadimos la columna mas.info de d2 en d1, en base a la columna caracter que comparten.
```

```r
d1$mas.info <- d2$mas.info[match(d1$caracter,d2$caracter)]
d1

# como d1 y d2 sólo comparten el caracter c,
# el resto de valores de la columna mas.info se asignan NA.

# otra manera muy importante y más general de unir dataframes
# en base a columnas comunes es la familia de funciones "join".
# Para usarlas, necesitamos cargar los paquetes "tidyverse",
# que trabajaremos bastante en el resto de sesiones.
library(tidyverse)
?join

# existen varias funciones para unir tablas,
# dependiendo de qué tipo de unión necesitamos.
# Por ahora veremos un ejemplo de left_join
# usaremos dos tablas con información sobre los
# personajes de la saga de Star Wars.
# En una tabla tenemos datos físicos de cada personaje,
# y en otra tabla tenemos información sobre las naves espaciales
# que cada personaje pilota.

personajes_SW <- read.csv2("../data/starwars_info_personajes.csv",
                           header = TRUE,
                           stringsAsFactors = FALSE)
naves_SW <- read.csv2("../data/starwars_personajes_naves.csv",
                      header = TRUE,
                      stringsAsFactors = FALSE)

head(personajes_SW)
head(naves_SW)

# podemos unir ambas tablas usando como columna común
# el nombre de los personajes.
SW_conjunta <- left_join(x = personajes_SW,y = naves_SW, by = "name")

# left_join añade las columnas de y (el segundo dataframe) a x (el primero).
head(SW_conjunta)
# en este caso todas las filas tienen su correspondencia en x e y
# (todos los nombres de x están en y, y viceversa).
# cuando esto no sucede, left_join devuelve valores NA donde corresponda.
```

Una cuestión importante en cuanto a la estructura de los dataframes es entender cómo están organizados los datos y qué implicaciones tiene esa organización para los análisis posteriores. En particular, hay dos maneras generales de organizar observaciones en dataframes (ver, por ejemplo, https://sejdemyr.github.io/r-tutorials/basics/wide-and-long/). Una es el formato "wide" (ancho), en el que cada variable o factor es una columna del dataframe.

```r
# en estos datos, cada nave es una columna
head(naves_SW)

# y en otro ejemplo con un subconjunto de terremotos,
# podemos tener la magnitud de cada terremoto en cada Área ordenada por años
eq.wide.data <- read.csv2("../data/Earthquake_wide_example.csv",header = TRUE)
```

```r
head(eq.wide.data)
# fijaos que R añade una "X" al principio de cada año/columna.
# Esto es porque nombres de columnas que empiecen por un número
# son considerados incorrectos.
# Si queréis evitar esto, usad el argumento "check.names = FALSE"
# al leer los datos.
```

El formato "long" (largo) agrupa las variables o factores en una columna y los valores en otra. Aunque pueda parecer contraintuitivo, es mucho más eficiente en general trabajar con datos en formato long. En nuestro ejemplo de Star Wars, es mucho más sencillo comparar visualmente los diferentes vehículos para un mismo personaje. Además, varias funciones avanzadas de R y otros lenguajes de programación trabajan con datos long.

```r
# podemos transformar datos de wide a long con las funciones "pivot"
# especificando
# qué columnas queremos incluir (cols), por su número,
# qué nombre tendrá la columna que diferencia los factores (names_to)
# qué nombre tendrá la columna de los datos en sí (values_to)
naves_SW_long <- pivot_longer(data = naves_SW,
                              cols = 2:17,
                              names_to = "spaceship",
                              values_to = "is.pilot")
head(naves_SW_long)

# otro ejemplo, con los datos simplificados de terremotos
eq_long <- pivot_longer(data = eq.wide.data,
                        cols = 2:47,
                        names_to = "year",
                        values_to = "magnitude")
head(eq_long)
# en este caso hay muchas combinaciones de área y año
# que no tuvieron terremotos, por lo que tenemos muchos NAs.
# podemos eliminarlos como hicimos más arriba
eq_long_clean <- eq_long[which(!is.na(eq_long$magnitude)),]
head(eq_long_clean)
```