

Explorando Data Frames

Luciano Selzer

28 June, 2018

¿Cómo manipular `data.frames`?

Añadir una nueva columna

```
edad <- c(2,3,5,12)
gatos
```

```
      pelaje peso gusta_ovillo
1 atigrado  2.1          TRUE
2   negro  5.0          FALSE
3 bicolor  3.2          TRUE
```

Podemos añadir la columna nueva con `cbind`:

```
gatos <- cbind(gatos, edad)
```

Error in `data.frame(..., check.names = FALSE)`: arguments imply differing number of rows: 3, 4

```
gatos
```

```
      pelaje peso gusta_ovillo
1 atigrado  2.1          TRUE
2   negro  5.0          FALSE
3 bicolor  3.2          TRUE
```

```
edad <- c(4,5,8)
gatos <- cbind(gatos, edad)
gatos
```

```
      pelaje peso gusta_ovillo edad
1 atigrado  2.1          TRUE    4
2   negro  5.0          FALSE    5
3 bicolor  3.2          TRUE    8
```

Añadir una nueva fila

```
nuevaFila <- list("carey", 3.3, TRUE, 9)
gatos <- rbind(gatos, nuevaFila)
```

Warning in ``[<-factor`(`*tmp*`, ri, value = "carey")`: invalid factor level, NA generated

No es posible añadir nuevo nivel al factor dinámicamente.

Hay que añadir el nuevo nivel de forma explícita y después añadir la nueva fila.

```
levels(gatos$pelaje)
```

```
[1] "atigrado" "bicolor" "negro"
```

```
levels(gatos$pelaje) <- c(levels(gatos$pelaje), 'carey')
gatos <- rbind(gatos, list("tortoiseshell", 3.3, TRUE, 9))
```

Warning in `[<-.factor`(`*tmp*`, ri, value = "tortoiseshell"): invalid factor level, NA generated

Otra forma de trabajar con esto es cambiar la columna a `character`.

```
str(gatos)
```

```
'data.frame':  5 obs. of  4 variables:
 $ pelaje      : Factor w/ 4 levels "atigrado","bicolor",...: 1 3 2 NA NA
 $ peso        : num  2.1 5 3.2 3.3 3.3
 $ gusta_ovillo: logi  TRUE FALSE TRUE TRUE TRUE
 $ edad        : num  4 5 8 9 9
```

```
gatos$pelaje <- as.character(gatos$pelaje)
str(gatos)
```

```
'data.frame':  5 obs. of  4 variables:
 $ pelaje      : chr  "atigrado" "negro" "bicolor" NA ...
 $ peso        : num  2.1 5 3.2 3.3 3.3
 $ gusta_ovillo: logi  TRUE FALSE TRUE TRUE TRUE
 $ edad        : num  4 5 8 9 9
```

Quitando cosas

Mientras estuvimos trabajando con la `data.frame`, añadimos una fila que no corresponde:

```
gatos
```

	pelaje	peso	gusta_ovillo	edad
1	atigrado	2.1	TRUE	4
2	negro	5.0	FALSE	5
3	bicolor	3.2	TRUE	8
4	<NA>	3.3	TRUE	9
5	<NA>	3.3	TRUE	9

Podemos eliminarla usando el signo menos

```
gatos[-4,]
```

	pelaje	peso	gusta_ovillo	edad
1	atigrado	2.1	TRUE	4
2	negro	5.0	FALSE	5
3	bicolor	3.2	TRUE	8
5	<NA>	3.3	TRUE	9

¿Cómo eliminar varias filas a la vez?

También podemos eliminar las filas que tienen NA

```
na.omit(gatos)
```

```
      pelaje peso gusta_ovillo edad
1 atigrado  2.1      TRUE      4
2   negro  5.0     FALSE      5
3 bicolor   3.2      TRUE      8
```

Reasignemos la salida de `na.omit` así hacemos permanentes los cambios.

```
gatos <- na.omit(gatos)
```

La clave para añadir datos a data.frames es recordar que:

- Las columnas son vectores
- Las filas son listas

También podemos pegar dos data.frames con `rbind`

```
gatos <- rbind(gatos, gatos)
gatos
```

```
      pelaje peso gusta_ovillo edad
1 atigrado  2.1      TRUE      4
2   negro  5.0     FALSE      5
3 bicolor   3.2      TRUE      8
4 atigrado  2.1      TRUE      4
5   negro  5.0     FALSE      5
6 bicolor   3.2      TRUE      8
```

Pero ahora los `rownames` (nombres de las filas) son innecesariamente complicados. Podemos quitar los `rownames` y R va a renombrarlos automáticamente.

```
rownames(gatos) <- NULL
gatos
```

```
      pelaje peso gusta_ovillo edad
1 atigrado  2.1      TRUE      4
2   negro  5.0     FALSE      5
3 bicolor   3.2      TRUE      8
4 atigrado  2.1      TRUE      4
5   negro  5.0     FALSE      5
6 bicolor   3.2      TRUE      8
```

Ejercicio 1

Ahora puedes crear una nueva data.frame directamente en R con la siguiente sintaxis:

```
df <- data.frame(id = c('a', 'b', 'c'),
                 x = 1:3, y = c(TRUE, TRUE, FALSE),
                 stringsAsFactors = FALSE)
```

Haz un data.frame que tenga la siguiente información sobre vos:

- nombre
- apellido
- número de la suerte

Ejercicio 1

Luego usa `rbind` para añadir una entrada para la gente que tenés al lado.

Y finalmente, usa `cbind` para añadir una columna para la respuesta de cada persona a la pregunta ¿Es momento para un recreo?

Hasta ahora hemos visto lo más básico para manipular data.frames.

Ahora vamos a ver un dataset más realista. Vamos a leer los datos de gapminder que hemos descargado previamente.

```
gapminder <- read.csv("data/gapminder-FiveYearData.csv")
```

Tip

- Se pueden leer archivos separados por tabulaciones con `read.delim`
- También se pueden leer archivos directamente desde Internet

```
gapminder <- read.csv("https://swcarpentry.github.io/r-novice-gapminder/data/gapminder-FiveYearData.csv")
```

- O directamente desde Excel sin convertirlo a csv con el paquete `readxl`

Investiguemos el nuevo dataset. Lo primero que hay que hacer es usar `str`:

```
str(gapminder)
```

```
'data.frame':  1704 obs. of  6 variables:
 $ country   : Factor w/ 142 levels "Afghanistan",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ year      : int   1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
 $ pop       : num   8425333 9240934 10267083 11537966 13079460 ...
 $ continent : Factor w/ 5 levels "Africa","Americas",...: 3 3 3 3 3 3 3 3 3 3 ...
 $ lifeExp   : num   28.8 30.3 32 34 36.1 ...
 $ gdpPercap: num   779 821 853 836 740 ...
```

También podemos investigar columnas individuales de nuestro dataset con la función `typeof`:

```
typeof(gapminder$year)
```

```
[1] "integer"
```

```
typeof(gapminder$lifeExp)
```

```
[1] "double"
```

```
typeof(gapminder$country)
```

```
[1] "integer"
```

```
str(gapminder$country)
```

```
Factor w/ 142 levels "Afghanistan",...: 1 1 1 1 1 1 1 1 1 1 ...
```

Además podemos ver sus dimensiones. `str` dice que tiene 1706 observaciones y 6 variables. ¿Qué creen que devolverá `length`?

```
length(gapminder)
```

```
[1] 6
```

```
typeof(gapminder)
```

```
[1] "list"
```

Para obtener el número de filas y columnas de nuestro dataset, intenta

```
nrow(gapminder)
```

```
[1] 1704
```

```
ncol(gapminder)
```

```
[1] 6
```

O ambos a la vez

```
dim(gapminder)
```

```
[1] 1704    6
```

También podríamos querer saber cuales son los nombres de las columnas

```
colnames(gapminder)
```

```
[1] "country" "year" "pop" "continent" "lifeExp" "gdpPercap"
```

En este punto es importante preguntarse si la estructura que nos está dando R es razonable y si coincide con nuestras expectativas.

¿Los tipos básicos de cada columna tienen sentido?

Una vez que estamos conformes con nuestros tipos de datos y estructuras es tiempo de empezar a investigar en nuestros datos.

```
head(gapminder)
```

	country	year	pop	continent	lifeExp	gdpPercap
1	Afghanistan	1952	8425333	Asia	28.801	779.4453
2	Afghanistan	1957	9240934	Asia	30.332	820.8530
3	Afghanistan	1962	10267083	Asia	31.997	853.1007
4	Afghanistan	1967	11537966	Asia	34.020	836.1971
5	Afghanistan	1972	13079460	Asia	36.088	739.9811
6	Afghanistan	1977	14880372	Asia	38.438	786.1134

Para asegurarnos que nuestro análisis es reproducible, deberíamos poner todo el código en un archivo de script para que podamos volver a él más tarde.

Ejercicio 2

Ve a File -> New File -> R Script, y escribe un script para cargar los datos de gapminder. Guardalo en la carpeta **scripts/** y añadelo al control de versión.

Ejecuta el archivo usando la función **source**, usando la ruta al archivo como argumento (o presionando el botón “source” en RStudio).

Ejercicio 3

Lee la salida de **str(gapminder)** de nuevo; esta vez, usa lo aprendido sobre factores, listas y vectores así como la salida de funciones como **colnames** y **dim** para explicar todo lo que imprime **str** sobre gapminder. Si hay partes que no comprendes discútelo con tus compañeros.