BSM

Blockchain School
for Management

Tema 5

Módulo II: R. Dataframes

Nicolás Forteza

2022-11-10

Qué son los dataframes

Un dataframe básicamente es una lista, pero cuya clase es Dataframe.

Los componentes deben ser vectores (números, booleanos, caracteres), factores, etc.

Recordatorio sobre listas

Una lista, si lo recordamos, es un objeto con elementos a los que se puede acceder con el operador `$`.

Para crear un dataframe a partir de una lista, simplemente tenemos que invocar la siguiente función: `data.frame`, y pasarle la lista de nuestro interés.

```
a = list(a=c(1, 2, 3), b=c(2, 3, 4))
data.frame(a)
```

Ejercicio

- Prueba a crear una lista con dos vectores, pero que difieren en longitud, y crea un dataframe. ¿Qué es lo que pasa?
- Crea un dataframe a partir de una lista. Sé creativo e invéntate dos o más variables a introducir en el dataframe.
- Crea un dataframe con diferentes tipos de variables (numéricas, caracteres, etc.)

Efectivamente, si creamos un dataframe a partir de una lista de vectores que tienen diferente longitud, no podremos crear dicho dataframe. Entonces:

- Es indispensable que si estamos creando un dataframe a partir de una lista, los componentes de la lista tengan la misma longitud.

Dataframes

Los data frames son estructuras de datos de dos dimensiones (tabulates) que pueden contener datos de diferentes tipos, por lo tanto, son heterogéneas. Esta estructura de datos es la más usada para realizar análisis de datos

Puedes crear dataframes sin especificar una lista.

Ejercicio

- Crea un dataframe a partir de vectores. Pista: pásale como argumentos de la función `data.frame()` varios vectores con diferentes nombres.
- Carga el siguiente set de datos: `data(cars)`.
- Selecciona la primera columna.

Accediendo a los dataframes

Para acceder a una columna de un dataframe por su nombre, usamos el operador $.

```
df = list(x=c(1, 2, 3), y=c(2, 3, 4))
df = data.frame(df)
df$x
```

```
[1] 1 2 3
```

Hemos seleccionado la primera columna.

Accediendo a columnas

¿Qué nos devuelve la selección de una columna?

```
is.vector(df$x)
```

```
[1] TRUE
```

Un dataframe está compuesto por vectores. Estos vectores pueden ser de diferentes tipos, como veíamos al inicio de la lección.

Naturalmente, podemos operar y realizar operaciones matemáticas con las columnas de los dataframe.

```
df$mult = df$x * df$y
```

Para crear variables, simplemente declaro una nueva variable con el operador $ del dataframe, indicando el nombre que se desee.

Se pueden crear columnas también con funciones, no sólo con operaciones de las columnas. Ejemplo:

```
df$unos = rep(1, nrow(df))
```

Ejericio

- Carga los siguientes datos `data(cars)`.
- Crea una nueva columna, siendo esa columna de tipo booleano un indicador de si la velocidad es mayor que 20. Llámala “fast”.
- Selecciona las filas donde se cumple la condición.

Para seleccionar filas, usamos booleanos. O podemos usar también vectores indicando el número de la fila:

```
cars[c(1:5), ]
```

```
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
```

Atributos de los dataframes

Tenemos por un lado, las dimensiones del dataframe, al igual que en las matrices.

```
dim(df)
```

```
[1] 3 4
```

También podemos ver cuantas filas y cuantas columnas tiene el dataframe

```
nrow(df)
```

```
[1] 3
```

```
ncol(df)
```

```
[1] 4
```

Podemos devolver los nombres de las columnas con esta función:

```
colnames(df)
```

```
[1] "x"    "y"    "mult" "unos"
```

Atributos de los dataframes

Podemos cambiar los nombres de las columnas para facilitar el análisis!

```
colnames(df) <- c("x1", "x2", "x3", "unos")
```

E incluso también los nombres de las filas

```
row.names(df) <- c("f1", "f2", "f3")
print(df)
```

```
   x1 x2 x3 unos
f1  1  2  2    1
f2  2  3  6    1
f3  3  4 12    1
```

Atributos de los dataframes

¿Qué pasa si accedemos a las filas de los dataframes ahora?
Podemos usar:

- Vector con los nombres de las filas
- Vector numérico con las posiciones

Es lo mismo!

```
df[c("f1"), ]
```

```
   x1 x2 x3 unos
f1  1  2  2    1
```

```
df[c(1), ]
```

```
   x1 x2 x3 unos
f1  1  2  2    1
```

Por dentro, los dataframes son matrices, que a su vez por dentro, son vectores:

```
m <- as.matrix(df)
class(m)
```

```
[1] "matrix" "array"
```

Y podemos operar de la misma manera que como si fueran matrices. En la práctica, nos da igual realizar operaciones con dataframes vs. con matrices, excepto cuando tenemos un gran volumen de datos.

Ejercicio

- Crea un dataframe de 3x4. Las 3 primeras columnas, números aleatorios. La última, una columna de tipo caracter con los siguientes valores (`c("A", "B", "C")`).
- Nombra las filas de data frame con nombres de plantas, árboles, flores… que se te ocurran (por ejemplo: margarita). Invéntate características de estas plantas y úsalas en las columnas. Recuerda que la última es de tipo caracter
- Carga en memoria el siguiente conjunto de datos: `iris`.
- Ejecuta el siguiente comando: `summary(iris)`.

Summary

La función `summary()` realiza una inspección/resumen de dataframe.

Para las variables numéricas, computa algunos estadísticos. Para las variables de tipo caracter, cuenta cuántos niveles del factor hay.

¿Alguna idea de qué pueden ser los *niveles*? ¿Y el factor?

Las variables de tipo factor son la representación en un dataframe de las variables que contienen caracteres.

```
is.factor(iris$Species)
```

```
[1] TRUE
```

Un nivel es, entonces, una categoría o uno de los valores que la variable recoge entre todos sus valores.

Por defecto, cuando construimos un dataframe, R no transforma las variables de tipo caracter a factor. Imaginemos que tenemos un dataframe con una variable en la que cada fila es, por ejemplo, una review de un negocio de Google Maps… demasiados niveles!

Por eso, nosotros podemos (y debemos) transformar las variables a factor siempre y cuando tengamos la ocasión de hacerlo.

Ejercicio

- Ejecuta la función summary para el dataframe que hemos construido en el ejercicio anterior. ¿qué devuelve?

Podemos transformar el tipo de una variable siempre que R nos lo permita.

```
a = data.frame(letras=c("A", "B", "C"))
is.factor(a)
```

```
[1] FALSE
```

```
a$letras = as.factor(a$letras)
a
```

```
  letras
1      A
2      B
3      C
```

Podemos obviamente hacer la transformación inversa siempre que lo necesitemos.

```
a$letras = as.character(a$letras)
```

Factores y caracteres

Con los factores y los caracteres, no podemos realizar operaciones matemáticas al uso.

Pero sí que son muy útiles para filtrar por el tipo de factor que queremos analizar en concreto.

Incluso podemos transformar variables numéricas a intervalos, deciles, etc… y transformar ese resultado a factor para hacer un análisis más detallado!

Ejercicio

- Con la función `quantile`, calcula el primer cuartil de la variable `speed` del set de datos `cars`, y calcula la media de ese primer cuartil.
- Crea una función que admite como argumenos un vector de entre 0 y 1, donde ese vector indica los percentiles que se quieren analizar, sobre una serie `x`, también admitida como argumento. Recuerda usar variables de tipo factor para indicar el cuartil al que pertenece cada valor de la variable.
- Usa la función para analizar los cuartiles de la variable `speed`.

Hemos visto que podemos cargar datos que R ya dispone en su librería `base`. Sin embargo, existen métodos de lectura de otros tipos de archivos. En concreto, vamos a aprender a cargar:

- Ficheros excel.
- Ficheros csv.
- Ficheros de texto plano.

Otros tipos de ficheros no son tan usados en el análisis de datos (SPSS, Stata, etc.), excepto si trabajas en un lugar en el que se usan estas herramientas. R en realidad puede leer cualquier tipo de fichero.

Rutas

Para cargar datos en excel, primero hay que tener bien claro dónde estamos trabajando, es decir, dónde está nuestro *working directory* fijado. Acordarse del Tema 1 cuando fijábamos y cambiábamos los directorios de trabajo.

Siempre y cuando cargamos ficheros desde nuestro local (es decir, desde nuestro ordenador, equipo, etc.), es muy importante que esto lo tengamos controlado; si no, pasaremos bastante tiempo con problemas de rutas y fallos en la carga de los ficheros

Siempre que cargamos un fichero en memoria, tenemos que especificar el lugar donde se encuentra dicho fichero. Por eso es tan importante.

Vamos a ver las funciones básicas y más comunes para cargar datos.

Carga de datos: Excel

Vamos a cargar datos de un fichero Excel con una librería.

```
library(readxl)
whaledata <- read_excel("datos/whaledata.xls")
head(whaledata)
```

```
# A tibble: 6 x 8
  month time.at.station water.noise number.whales latitude
  <chr>           <dbl> <chr>       <chr>             <dbl>
1 May              1344 low         7                  60.4
2 May              1633 medium      13                 60.4
3 May               743 medium      12                 60.5
4 May              1050 medium      10                 60.3
5 May              1764 medium      12                 60.4
6 May               580 high        10                 60.4
# ... with 1 more variable: gradient <dbl>
```

¿Qué hace la función `head`?

Carga de datos:

```
df <- read.table("datos/mtcars.txt", header = T, row.names
```

Carga de datos: CSV

```
df <- read.csv("datos/Housing (1).csv", header = T, sep=","
dim(df)
```

```
[1] 545  13
```

Ejercicio

- Calcula el precio medio de las casas con menos de 3 habitaciones.
- Calcula el área media que tienen las casas con aire acondicionado y las que no.
- Transforma la columna `furnishingstatus` a factor. Calcula el número de baños medio por cada nivel y la moda de la variable `parking`. Introduce esta información en un dataframe.

Ejercicio

- Crea una función que si le pasas una columna de un dataframe, te devuelve los siguientes estadísticos: media, mediana, percentil 25, percentil 75, desviación estándar. Aplica esta función a las variables numéricas del dataset `Housing`.

Guardado de dataframes

Podemos a su vez, guardar las tablas que queramos en el formato que queramos. Se suele utilizar csv por su facilidad y legibilidad.

```
write.csv(df, "datos/datos_modificados.csv")
```

Tibble

Es una libreria o paquete de dataframes.

Tibbles are a modern take on data frames. They keep the features that have stood the test of time, and drop the features that used to be convenient but are now frustrating (i.e. converting character vectors to factors).

Para instalarla:

```
install.packages("tibble")
```

Como son?

```
library(tibble)
df <- iris
as_tibble(df)
```

Se pueden crear como si fueran dataframes:

```
tibble(
  x = 1:5,
  y = 1,
  z = x ^ 2 + y
)
```

`tibble()` es una buena forma de crear dataframes. La librería encapsula las mejores prácticas para data frames. Nunca cambia el input del dato (no transforma a tipo factor).

```
tibble(x = letters)
```

Esto hace que sea fácil con listas-columnas.

```
tibble(x = 1:3, y = list(1:5, 1:10, 1:20))
```

Es más rápido que los data.frames de R base.

```
library(tibble)
l <- replicate(26, sample(100), simplify = FALSE)
names(l) <- letters

timing <- bench::mark(
  as_tibble(l),
  as.data.frame(l),
  check = FALSE
)

timing
```

```
# A tibble: 2 x 6
  expression             min   median `itr/sec` mem_alloc `g
  <bch:expr>        <bch:tm> <bch:tm>     <dbl> <bch:byt>
1 as_tibble(l)        121us    130us     7350.    1.55KB
```

Tibble vs. dataframe

Hay 3 principales diferencias: print, subset y reglas de reciclaje.

```
tibble(x = -5:100, y = 123.456 * (3 ^ x))
```

```
# A tibble: 106 x 2
       x          y
   <int>      <dbl>
 1    -5      0.508
 2    -4      1.52
 3    -3      4.57
 4    -2     13.7
 5    -1     41.2
 6     0    123.
 7     1    370.
 8     2   1111.
 9     3   3333.
10     4  10000.
# ... with 96 more rows
```

Print en tibble

Puedes controlar las opciones de print() de un dataframe de la siguiente manera:

```
options(pillar.print_max = n, pillar.print_min = m)
options(pillar.width = n)
```

Las diferencias son obvias:

```
df1 <- data.frame(x = 1:3, y = 3:1)
class(df1[, 1:2])
```

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

```
class(df1[, 1])
```

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

```
df2 <- tibble(x = 1:3, y = 3:1)
class(df2[, 1:2])
```

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

Cuando se construye un Tibble, sólo valores de 1 son reciclados. La primera columna diferente de de 1 determina el número de filas del Tibble.

```
tibble(a = 1:3, b = 1)
```

```
# A tibble: 3 x 2
      a     b
  <int> <dbl>
1     1     1
2     2     1
3     3     1
```

```
tbl <- tibble(a = 1:3, b = 4:6)
tbl * 2
```

```
  a  b
1 2  8
2 4 10
3 6 12
```