Introducción a R:

Los primeros pasos

Carlos Iván Espinosa

10 de octubre 2019

Contents

Regresar Introducción R

Puedes descargar esta lección en pdf aquí

Los objetos

Como lo vimos en el apartado anterior, R opera sobre objetos y estos objetos pueden ser diversos. Los diferentes tipos de objetos poseen unas determinadas características de estructura, en el entorno R el tipo de objeto se conoce como modo (mode) o clase (class).

Cuando estoy generado un objeto, es necesario darle una denominación, un nombre, y asignarle a este unos datos. En R la función de asignación es `<-`.

```
profe <- "Carlos Iván"

profe
```

```
## [1] "Carlos Iván"
```

Hemos creado el objeto *profe* asignando el nombre Carlos Iván. Si ejecutamos el nombre del objeto R nos devuelve el contenido. Veamos otro objeto.

```
Notas <- c(rep(10,5), rep(7,3), rep(8.5, 12))

notas

#Error: object 'notas' not found
```

¿Qué es lo que sucedió?

Para R los nombres no son entendidos como palabras sino como una serie de símbolos, por tanto la "N" no es lo mismo que la "n", por lo que es necesario poner exactamente el nombre. En el caso del ejemplo, Notas no es igual que notas. Volvamos a intentar.

```
Notas
```

```
##  [1] 10.0 10.0 10.0 10.0 10.0  7.0  7.0  7.0  8.5  8.5  8.5  8.5  8.5  8.5  8.5
## [16]  8.5  8.5  8.5  8.5  8.5
```

Ahora si podemos ver que tenemos un curso muy aplicado.

Operación de los objetos

A continuación, vamos a realizar un pequeño programa que nos ayude a entender como los objetos pueden ser operados a través de comandos. Haremos un seguimiento de los costos de un producto. Mi hija María Sol tiene un pequeño negocio de chocolates, vamos a utilizar lo que ella hace para ver lo potente que es R.

Para hacer los chocolates ella usa los siguientes ingredientes:

- Chocolate (Choco)
- Nuez
- Dulce de leche (D.leche)
- Empaques (Empa)

Necesitamos saber cuál es el costo por chocolate y poder calcular una ganancia.

```
Choco <- 7 #rinde 60 chocolates
Nuez <- 2.5 #rinde 40 chocolates
D.leche <- 2 #rinde 50 chocolates
Empa <- 0.10 #por cada chocolate

#¿Cuánto cuesta cada chocolate?

costo <- (Choco/60)+(Nuez/40)+(D.leche/50)+Empa
costo
```

```
## [1] 0.3191667
```

```
#¿Cuanto debo sumar si quiero ganar el 30%?

ganancia <- costo*0.3
ganancia
```

```
## [1] 0.09575
```

```
#¿Cuánto cuesta cada chocolate?

pvp<- costo+ganancia
pvp
```

```
## [1] 0.4149167
```

```
#¿Cuantos chocolates debo vender si quiero ganar 100 USD mensuales?

venta<- 100/ganancia
venta
```

```
## [1] 1044.386
```

Como esto puede ser engorroso podríamos desarrollar una función que calcule cada uno de estas mediciones.

```r
negocio <- function(C,N,D,E,G){
  x <- (C/60)+(N/40)+(D/50)+E
  y <- x*G
  z <- x+y

  neg <- c(x,y,z)
  names(neg) <- c("costo", "ganancia", "PVP")
  return(neg)
}

#Veamos cuanto cuesta el chocolate

negocio(C=7,N=2.5,D=2,E=0.10, G=0.3)
```

```
##     costo  ganancia       PVP
## 0.3191667 0.0957500 0.4149167
```

```r
#Ahora puede cambiar el costo de cualquiera de
#los elementos y tendrá automáticamente los parámetros
#de su negocio
```

Como ven R es muy potente y podemos hacer muchas cosas con el, cada uno de los objetos pueden ser operados, en este caso a los objetos los hemos multiplicado, dividido o sumado. Verán más adelante que las operaciones pueden ser mucho más complejas.

Tipos de Objetos

Los objetos pueden tener varios tipos (typeof) y estos se diferencian por el tipo de datos (elementos) por los que están conformados. Los objetos más comunes son los objetos dobles, enteros, lógicos y carácter.

Veamos un ejemplo de este tipo de objetos.

```r
d <- 3.5
e <- 8L
l <- e>d
c <- "a"

d;e;l;c
```

```
## [1] 3.5
```

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

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

```
## [1] "a"
```

Podemos preguntar a R el tipo de objeto con el que estamos trabajando, para esto utilizamos las funciones *is.double, is.integer, is.logical, is.character*

```r
is.double(d); is.double(e); is.double(l)
```

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

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

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

```
is.integer(d); is.integer(e); is.integer(l)
```

```
## [1] FALSE
## [1] TRUE
## [1] FALSE
```

```
is.logical(d); is.logical(l); is.logical(c)
```

```
## [1] FALSE
## [1] TRUE
## [1] FALSE
```

```
is.character(d); is.character(l); is.character(c)
```

```
## [1] FALSE
## [1] FALSE
## [1] TRUE
```

Como vemos cada uno de estos objetos son diferentes. Los objetos dobles (double) están formados por datos continuos, mientras que los enteros (integer) están formados por datos de tipo conteo. Finalmente, los objetos lógicos se dan luego de una operación lógica. Podemos preguntar directamente el tipo de datos que tiene el objeto con la función *typeof*

```
typeof(d); typeof(l); typeof(c)
```

```
## [1] "double"
## [1] "logical"
## [1] "character"
```

Estructura de los objetos

La estructura de los objetos en R puede ser descrita en base de su dimensionalidad y en base a su constitución. Los objetos pueden tener una, dos o n dimensiones, y pueden ser homogéneos o heterogéneos en cuanto al tipo de elementos que lo constituyen. Como vemos en la siguiente tabla, en función de estas dos características podemos tener algunos tipos de estructuras de los objetos.

```
## Warning: package 'knitr' was built under R version 4.0.3
```

Table 1: Estructura de objetos. Fuente: Wickham, 2014

	Homogéneos	Heterogéneos
1d	Atomic vector	List
2d	Matrix	Data frame
nd	Array	

Ahora vamos a ver en detalle cada uno de los objetos.

Vectores (Vectors)

Los vectores son las estructuras más simples de R. Los vectores tienen una sola dimensión, y los elementos que lo constituyen definen el tipo de vector que es, así, si es un vector con números enteros será un vector numérico (integrer), o un vector con letras será un vector de carácter (character). El vector puede ser desde

un solo valor hasta varios miles, pero debe estar constituido por un solo tipo de elemento.

Veamos algunos ejemplos de vectores.

```
a <- 5:12 #vector numérico
b <- a>=6&a<=10 #vector lógico
c <- c(letters[1:10]) #vector de carácter

a;b;c
```

```
## [1]  5  6  7  8  9 10 11 12
```

```
## [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

Cada uno de estos vectores fue generado utilizando diferentes funciones o códigos. El vector numérico se generó utilizando únicamente una secuencia de datos entre 5 y 12, lo hicimos utilizando los dos puntos, esto nos sirve cuando queremos una secuencia ininterrumpida entre dos números, sin embargo, si queremos tener secuencias con diferentes distancias. Para el vector lógico hemos utilizado operadores lógicos como **igual o mayor que (>=)**, **menor o igual que (<=)** , **y (&)**. Finalmente, para generar un vector de carácter hemos utilizado la función **concatenación (*c*)**, esta función permite encadenar varios componentes en un vector.

Veamos el tipo de vector que hemos generado.

```
mode(a);mode(b);mode(c)
```

```
## [1] "numeric"
```

```
## [1] "logical"
```

```
## [1] "character"
```

Como comentamos los vectores lógicos son generados a partir de expresiones lógicas (en la tabla 1 se pueden ver algunos operadores lógicos).

Tabla 1: Operadores Lógicos

Descripción	Operadores
Mayor que	$>$
Menor que	$<$
Mayor o igual que	$>=$
Menor o igual que	$<=$
Igual que	$==$
No es igual que	$!=$

En el caso de los vectores numéricos y categóricos hemos utilizado secuencia y concatenar, pero podríamos utilizar algunas otras funciones.

```
secA<- seq(from=10, to=290, by=20)
secA
```

```
##  [1]  10  30  50  70  90 110 130 150 170 190 210 230 250 270 290
```

Aquí hemos utilizado la secuencia entre 10 y 290, pero le hemos dicho que lo haga cada 20 unidades. R entiende el orden de los datos proporcionados, así que la expresión que acabamos de ejecutar es exactamente igual a: `secA<- seq(10, 290, 20)`

Otra de las funciones que se ocupan mucho para la generación de los vectores es la función `rep`. Esta función permite repetir varias veces un argumento.

```
repA <- rep(1:5, 3) #Repite la secuencia de uno a cinco, tres veces

repB <- rep (1:5, c(3,2,7,2,8)) #Repite para cada número de la secuencia las veces indicada por el vect

repC <- rep(letters[1:3], 3) #Repite las letras de uno a tres, tres veces

repA; repB; repC
```

```
##  [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
```

```
##  [1] 1 1 1 2 2 3 3 3 3 3 3 3 4 4 5 5 5 5 5 5 5
```

```
## [1] "a" "b" "c" "a" "b" "c" "a" "b" "c"
```

Podemos generar un vector de carácter al combinar letras y números en un vector, para esto utilizamos la función `paste`.

```
pasA <- paste(LETTERS[8:16], 1:8, sep="_")

#Algo más complicado

pasB <- paste(letters[1:9], rep(1:3, 3), sep="a" )

pasA; pasB
```

```
## [1] "H_1" "I_2" "J_3" "K_4" "L_5" "M_6" "N_7" "O_8" "P_1"
```

```
## [1] "aa1" "ba2" "ca3" "da1" "ea2" "fa3" "ga1" "ha2" "ia3"
```

La función *paste* requiere tres argumentos; los dos vectores que serán unidos y un símbolo de separación.

Como vemos los vectores deben estar compuestos por un solo tipo de elementos, si un vector tiene más de un tipo de elemento estos son `coaccionados` al tipo más flexible. De esta forma en un vector que tenga números y una letra, estos pueden ser coaccionados (coerced) a caracteres.

```
a <- c(2,5,7,5,3, 7, "a")
typeof(a)
```

```
## [1] "character"
```

```
a
```

```
## [1] "2" "5" "7" "5" "3" "7" "a"
```

Como vemos este vector fue convertido en un vector tipo carácter. Los vectores lógicos son coaccionado a 0 y 1, falso y verdadero respectivamente.

```
al <- a=="a"
al
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
```

```
sum(al)
```

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

Podemos forzar una coerción usando la función `as.numeric` por ejemplo, para cambiar unos factores a números. Usaremos `as.character` para convertir un vector a caracteres, o `as.matrix` para convertir un data frame en una matriz.

Factores (Factors)

Los factores es un modo sencillo en que se guardan las variables categóricas. Si tenemos un vector con 50 hombres y 50 mujeres, si el vector se encuentra como factor, en vez de tener 100 datos lo que tengo es hombres 50 y mujeres 50. Cada categoría se repite una cierta cantidad de veces.

```
cat <- rep(c("alto", "medio", "bajo"), c(10, 20,25))

cat <- factor(cat)

levels(cat)
```

```
## [1] "alto"  "bajo"  "medio"
```

La función `factor` nos permite convertir el vector de caracteres en un vector factor. Ejecute la primera línea de código y luego el nombre de este objeto, ahora ejecute la segunda línea y luego el nombre del objeto. ¿Cuál es la diferencia?

Efectivamente cuando ejecuta la segunda línea al final aparece una observación: `Levels: alto bajo medio`

Como vemos los niveles (levels) son mostrados en base a un orden alfanúmerico, este orden será el utilizado para los gráficos y los análisis por lo que es importante saber si este orden es el que queremos. Si no es así podemos utilizar la función `relevel` para decir cuál es el nivel que queremos que salga primero, o en la función factor informar el orden de los niveles.

```
cat1 <- relevel(cat, ref = "bajo")

cat2 <- factor(cat, levels = c("bajo", "medio", "alto"))

cat1; cat2
```

```
##  [1] alto  alto  alto  alto  alto  alto  alto  alto  alto  alto  medio medio
## [13] medio medio medio medio medio medio medio medio medio medio medio medio
## [25] medio medio medio medio medio medio bajo  bajo  bajo  bajo  bajo  bajo
## [37] bajo  bajo  bajo  bajo  bajo  bajo  bajo  bajo  bajo  bajo  bajo  bajo
## [49] bajo  bajo  bajo  bajo  bajo  bajo  bajo
## Levels: bajo alto medio
```

```
##  [1] alto  alto  alto  alto  alto  alto  alto  alto  alto  alto  medio medio
## [13] medio medio medio medio medio medio medio medio medio medio medio medio
## [25] medio medio medio medio medio medio bajo  bajo  bajo  bajo  bajo  bajo
## [37] bajo  bajo  bajo  bajo  bajo  bajo  bajo  bajo  bajo  bajo  bajo  bajo
## [49] bajo  bajo  bajo  bajo  bajo  bajo  bajo
## Levels: bajo medio alto
```

Podemos también unificar niveles, reduciendo la cantidad de niveles resultantes. Vamos a unificar los niveles alto y medio en un nivel llamado *contaminado* y el bajo lo vamos a llamar *no contaminado*.

```
cat3 <- cat
levels(cat3) <- list(no.contaminado = "bajo", contaminado= c("medio", "alto"))
cat3
```

```
##  [1] contaminado    contaminado    contaminado    contaminado    contaminado
##  [6] contaminado    contaminado    contaminado    contaminado    contaminado
## [11] contaminado    contaminado    contaminado    contaminado    contaminado
## [16] contaminado    contaminado    contaminado    contaminado    contaminado
## [21] contaminado    contaminado    contaminado    contaminado    contaminado
## [26] contaminado    contaminado    contaminado    contaminado    contaminado
## [31] no.contaminado no.contaminado no.contaminado no.contaminado no.contaminado
```

```
## [36] no.contaminado no.contaminado no.contaminado no.contaminado no.contaminado
## [41] no.contaminado no.contaminado no.contaminado no.contaminado no.contaminado
## [46] no.contaminado no.contaminado no.contaminado no.contaminado no.contaminado
## [51] no.contaminado no.contaminado no.contaminado no.contaminado no.contaminado
## Levels: no.contaminado contaminado
```

Muy bien lo que hemos hecho es transformar los niveles iniciales a dos nuevos niveles. Muchas veces cuando trabajamos con datos nos interesa hacer lo contrario, los datos numéricos transformarlos a categorias. Esto lo podemos hacer con la función *cut()*.

```
x <- 1:100 # Porcentaje de contaminación

xcat <- cut(x, breaks = c(0, 30, 70, 100))
xcat1 <- cut(x, breaks = c(0, 30, 70, 100), labels=c("bajo", "medio", "alto"))
```

Lo que hemos hecho es generar tres niveles; el primero entre 0 y 30, el segundo entre 30 y 70 y el tercero entre 70 y 100. En xcat1 hemos asignado unos nombres a cada una de las nuebas categorias con el argumento *labels*.

Listas

Una lista es una colección ordenada de elementos de distinto tipo. Una lista puede contener otra lista, y de este modo puede utilizarse para construir estructuras de datos arbitrarias. Las listas son utilizadas por R como salidas de las funciones estadísticas.

Las listas al igual que los vectores tienen una sola dimensión, pero a diferencia de los vectores estas pueden estar compuestas por diferentes tipos de elementos.

```
listA <- list("a", "b", c(1))
listA
```

```
## [[1]]
## [1] "a"
##
## [[2]]
## [1] "b"
##
## [[3]]
## [1] 1
```

Como vemos las listas pueden tener una estructura lineal, pero puede estar compuesta por diversos elementos. Incluso la lista puede incluir listas, así se puede generar una estructura de datos jerarquizada.

```
listA <- list(1:5, list(rep(1,3)), list("a", "b", "c", list(rep("a",3))))
str(listA)
```

```
## List of 3
##  $ : int [1:5] 1 2 3 4 5
##  $ :List of 1
##   ..$ : num [1:3] 1 1 1
##  $ :List of 4
##   ..$ : chr "a"
##   ..$ : chr "b"
##   ..$ : chr "c"
##   ..$ :List of 1
##   .. ..$ : chr [1:3] "a" "a" "a"
```

Para convertir la lista en un vector podemos usar la función `unlist`, lo que vuelve la lista a un vector. Si esta lista contiene elementos de diferente tipo los elementos serán coaccionados.

```
unlist(listA)
```

```
##  [1] "1" "2" "3" "4" "5" "1" "1" "1" "a" "b" "c" "a" "a" "a"
```

Matrices (Matrix)

Hasta ahora hemos visto vectores y listas, dos objetos unidimensionales, vamos a trabajar con las matrices. Las matrices tienen dos dimensiones; filas y columnas, y tienen una constitución homogénea, es decir los elementos son del mismo tipo.

```
mat <- matrix(1:9, 3, 3, byrow = FALSE)
mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
mode(mat)
```

```
## [1] "numeric"
```

```
class(mat)
```

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

Hemos utilizado la función `matrix` para generar una matriz, los argumentos para ejecutar la función son en orden de aparición; *datos*, los datos que queremos que se escriban en la matriz,el número de *filas* y *columnas* que constituirán la matriz, finalmente puedo incluir si el llenado es por columnas (byrow = FALSE), o por filas (byrow = TRUE).

Otra forma de convertir un vector en matriz es utilizando la función `dim`. Utilizaremos el vector x que lo generamos hace un momento y convertiremos en una matriz de 10x10.

```
dim(x) <- c(10, 10)
x
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1   11   21   31   41   51   61   71   81    91
##  [2,]    2   12   22   32   42   52   62   72   82    92
##  [3,]    3   13   23   33   43   53   63   73   83    93
##  [4,]    4   14   24   34   44   54   64   74   84    94
##  [5,]    5   15   25   35   45   55   65   75   85    95
##  [6,]    6   16   26   36   46   56   66   76   86    96
##  [7,]    7   17   27   37   47   57   67   77   87    97
##  [8,]    8   18   28   38   48   58   68   78   88    98
##  [9,]    9   19   29   39   49   59   69   79   89    99
## [10,]   10   20   30   40   50   60   70   80   90   100
```

En este caso lo que hicimos es decir a R que las dimensiones de x es de 10 filas por 10 columnas.

Marco de Datos (data.frame)

El data frame al igual que la matriz es un objeto bidimensional con filas y columnas. Sin embargo, a diferencia de la matriz el data frame puede estar conformado por datos de diferentes tipos (numéricos y caracteres).

Veamos con un ejemplo:

```
cont <- round(rnorm(20, 60, 20), 0)
catcont <- cut(cont, breaks=c(0,30,70,max(cont)))
```

```
levels(catcont) <- c("bajo", "medio", "alto")

conta <- cbind(cont, catcont)
class(conta)
```

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

```
mode(conta)
```

```
## [1] "numeric"
```

Como vemos, esta función de pegado `cbind`, si bien junta los datos, convierte los datos categóricos en numéricos, y genera una matriz con los mismos datos. Como tenemos datos numéricos y categóricos, los deberíamos unir como marco de datos (data frame), para ello utilizamos la función `data.frame`.

```
conta1 <- data.frame(cont, catcont)
class(conta1)
```

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

```
mode(conta1)
```

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

Arreglos (Arrays)

Los arreglos son objetos tridimensionales, en este caso son la unión de varias matrices. Al igual que las matrices los arreglos están constituidos por datos del mismo tipo.

Para la construcción de arreglos podemos utilizar la función `array`.

```
y <- array(1:9, c(3,3,3))

y
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Los argumentos de la función array son: los datos que queremos se escriban en la matriz (1:9), utilizando la función concatenar (c), ponemos el número de filas, el número de columnas y el número de matrices a generar.

Ejercicios

1. Genere un proyecto con el nombre "Trabajo 1_Nombre grupo".
2. Abra un script y llámele "resolución de ejercicios".
3. Genere una estructura de índices del Script. Esta estructura debería referenciar desde la pregunta 4 hasta el final de las preguntas de este ejercicio.
4. Genere los siguientes vectores:

- Un vector con datos de edad de los estudiantes del aula.
- Un vector con la altura de los estudiantes del aula.
- Un vector con el género de los estudiantes del aula.
- Un vector para conocer aquellos estudiantes mayores de 25

Responda:

¿Qué tipo de vectores hemos generado? ¿Cuantos estudiantes mayores a 25 años hay en el aula?

5. Genere una lista con el vector de edad y género. Vuelva a convertir en vector. ¿El vector resultante es numérico o carácter? Explique la razón del resultado.
6. Genere 3 categorías de edad, entre 18 y 20, entre 20 y 25 y mayores de 25. Nombre a cada categoría; jovenes, adultos y maduros. Para poner el nombre use la función `levels` para asignar los nombres. Modifique el orden de las categorías para que aparezca primero maduro, luego adulto y finalmente joven.
7. Genere una matriz con los datos numéricos de los vectores antes generados.
8. Genere una data frame con los vectores numéricos y caracteres.
9. Convierta el data frame en una matriz y verifique que tipo de elementos constituyen esta matriz. De que tipo son los datos de esta matriz.