# Operaciones, extración y otras funcionalidades entre tipos de estructuras de datos

Santiago Lozano

21 de febrero de 2020

# Evaluación de combinaciones de TRUE o FALSE

Es importante saber cómo se operan los conectores lógicos vistos anteriormente, referente a los distintos valores logical que pueden resultar, además de cómo se operan los Missing Values con los otros valores lógicos, veamos todas las combinaciones mediante la siguiente codificación

```
x <- c(NA, FALSE, TRUE)
names(x) <- as.character(x)
x
```

```
##  <NA> FALSE  TRUE
##    NA FALSE  TRUE
```

# Evaluación de combinaciones de TRUE o FALSE

Veamos las distintas combinaciones mediante el conector lógico y (&)

```
outer(x, x, "&")
```

```
##         <NA> FALSE  TRUE
## <NA>      NA FALSE    NA
## FALSE FALSE FALSE FALSE
## TRUE      NA FALSE  TRUE
```

# Evaluación de combinaciones de TRUE o FALSE

Ahora mediante el conector ó (|)

```
outer(x, x, "|")
```

```
##       <NA> FALSE TRUE
## <NA>    NA    NA TRUE
## FALSE   NA FALSE TRUE
## TRUE  TRUE  TRUE TRUE
```

# Lógica Aritmética

La aritmética que envuelven las expresiones lógica las diferentes estrucutras de datos juega un papel muy importante a la hora de programar nos indicará de una u otra forma que elementos escoger y cuales no, la clave para entende esta parte es que las expresiones lógicas evaluan si es cierto o falso alguna proposición, y qué R puede convertir en valores numéricos 1 para TRUE, 0 para FALSE

```
x<- 0:6
```

```
x < 4
```

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

# Lógica Aritmética

Para poder chequear si todos los valores de un vector cumplen con la condición o para verificar si alguno cumple la condición usamos all() y any()

```
all(x>0)
```

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

```
any(x<0)
```

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

# Lógica Aritmética

Podemos usar las respuestas de las funciones lógicas en aritmética. Podemos contar los valores TRUE de (x<4), usando sum

```
sum(x<4)
```

```
## [1] 4
```

Podemos multiplicar el vector (x<4) por otros vectores

```
(x<4)*runif(7)
```

```
## [1] 0.2541504 0.3750152 0.9489598 0.6974481 0.0000000
## [6] 0.0000000 0.0000000
```

# Lógica Aritmética

```
y <- c(4,NA,7)
y == NA # no funciona
```

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

U

```
y == "NA" # no funciona
```

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

```
is.na(y)
```

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

# Lógica Aritmética

La lógica aritmética es útil para generar niveles simplificados de factores durante el modelamiento estadístico. Suponga que queremos reducir un factor de 5 niveles (a, b, c, d, e) llamado tratamiento a un factor de 3 niveles llamado t2 juntados los niveles a y e (nuevo factor nivel 1) y c y d (nuevo factor nivel 3) mientras que dejamos b solo (nuevo factor nivel 2)

```
tratamiento <- letters[1:5]
tratamiento
```

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

```
1 + (tratamiento == "b")
```

```
## [1] 1 2 1 1 1
```

# Lógica Aritmética

```
t2 <- factor(1+(tratamiento=="b")+2*(tratamiento=="c")
             +2*(tratamiento=="d"))
t2
```

```
## [1] 1 2 3 3 1
## Levels: 1 2 3
```

# Distinciones entre las igualdades

`x <- y` a x le asigno el valor de y

`x = y` en una función o una lista x se establece en y, a menos que especifique lo contrario

`x == y` produce TRUE si x es exactamente igual a y y falso en otro caso

# Resumén de diferencias entre objetos usando all.equal

La función all.equal es muy útil in programación para chequear que los objetos son realmente como tu esperabas. Cuando ocurren diferencias, all.equal hace un trabajo útil en describir todas las diferencias encontradas

```
a <- c("cat","dog","goldfish")
b <- factor(a)
```

```
all.equal(a,b)
```

```
## [1] "Modes: character, numeric"
## [2] "Attributes: < target is NULL, current is list >"
## [3] "target is character, current is factor"
```

# Resumen de diferencias entre objetos usando all.equal

en este caso el objeto de la izquierda (a) es llamado “target” y el objeto de la derecha (b) es “current”

```
mode(b)
```

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

```
mode(a)
```

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

# Resumen de diferencias entre objetos usando all.equal

```
attributes(b)
```

```
## $levels
## [1] "cat"       "dog"       "goldfish"
##
## $class
## [1] "factor"
```

```
attributes(a)
```

```
## NULL
```

# Resumén de diferencias entre objetos usando all.equal

```
n1 <- c(1,2,3)
n2 <- c(1,2,3,4)
all.equal(n1,n2)
```

```
## [1] "Numeric: lengths (3, 4) differ"
```

```
n2 <- as.character(n2)
all.equal(n1,n2)
```

```
## [1] "Modes: numeric, character"
## [2] "Lengths: 3, 4"
## [3] "target is numeric, current is character"
```

# Funciones básicas para vectores

Una de las fortalezas de R es la habilidad de evaluar funciones sobre vectores enteros, con gran utilidad a la hora de operar ciclos, veamos algunas de las funciones más importantes, la mayoría de las funciones aplican para vectores de tipo numeric

```
y <- c(8,3,5,7,6,6,8,9,2,3,9,4,10,4,11)
```

función media

```
mean(y)
```

```
## [1] 6.333333
```

# Funciones básicas para vectores

Largo de un vector

```
length(y)
```

```
## [1] 15
```

rango de los números en un vector

```
range(y)
```

```
## [1]  2 11
```

# Funciones básicas para vectores

ordenar los elementos del vector de menor a mayor

```
sort(y)
```

```
##  [1]  2  3  3  4  4  5  6  6  7  8  8  9  9 10 11
```

de forma decreciente

```
sort(y,decreasing = TRUE)
```

```
##  [1] 11 10  9  9  8  8  7  6  6  5  4  4  3  3  2
```

# Funciones básicas para vectores

reodernar los elementos del vector en order reversivo

```
rev(y)
```

```
##  [1] 11  4 10  4  9  3  2  9  8  6  6  7  5  3  8
```

remover los duplicados en un vector

```
unique(y)
```

```
##  [1]  8  3  5  7  6  9  2  4 10 11
```

# Funciones básicas para vectores

valor lógico en cada elemento que afirma si está duplicado o no

```
duplicated(y)
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
##  [8] FALSE FALSE  TRUE  TRUE  FALSE FALSE TRUE
## [15] FALSE
```

which indica que índice lleva el valor TRUE

```
mask <- c(TRUE,FALSE,TRUE,NA,FALSE,FALSE,TRUE)
which(mask)
```

```
## [1] 1 3 7
```

# Funciones básicas para vectores

Para buscar el índice con el menor valor

```
which.min(y)
```

```
## [1] 9
```

ahora con el mayor valor

```
which.max(y)
```

```
## [1] 15
```

# Funciones básicas para vectores

para nombrar los elementos de un vector

```
names(y) <- 1:15
y
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
##  8  3  5  7  6  6  8  9  2  3  9  4 10  4 11
```

La suma acumulada

```
cumsum(y)
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
##  8 11 16 23 29 35 43 52 54 57 66 70 80 84 95
```

# Funciones básicas para vectores

El producto acumulado

```
cumprod(y)
```

```
##         1         2         3         4         5
##         8        24       120       840      5040
##         6         7         8         9        10
##     30240    241920   2177280   4354560  13063680
##        11        12        13        14        15
##117573120 470292480 4702924800 18811699200 206928691200
```

# Funciones básicas para vectores

mínimo

```
min(y)
```

```
## [1] 2
```

```
max(y)
```

```
## [1] 11
```

```
quantile(y)
```

```
##   0%  25%  50%  75% 100%
##  2.0  4.0  6.0  8.5 11.0
```

# Funciones básicas para vectores

con pmin y pmax tomamos tres vectores con el mismo largor y hallamos el valor mínimo de cada componente

```
x<-c(0.99,0.98,0.20,0.65,0.93,0.18)
x
```

```
## [1] 0.99 0.98 0.20 0.65 0.93 0.18
```

```
y<-c(0.51,0.30,0.41,0.53,0.07,0.49)
y
```

```
## [1] 0.51 0.30 0.41 0.53 0.07 0.49
```

# Funciones básicas para vectores

```
z<-c(0.26,0.132,0.44,0.65,0.031,0.36)
z
```

```
## [1] 0.260 0.132 0.440 0.650 0.031 0.360
```

```
pmin(x,y,z)
```

```
## [1] 0.260 0.132 0.200 0.530 0.031 0.180
```

# Funciones básicas para vectores

para contar la cantidad el número de elementos de cada componente del vector

```
table(y)
```

```
## y
## 0.07  0.3 0.41 0.49 0.51 0.53
##    1    1    1    1    1    1
```

ordena los elementos pero despliega los índices

# Funciones básicas para vectores

```
y <- c(8,3,5,7,6,6,8,9,2,3,9,4,10,4,11)
```

```
order(y)
```

```
##  [1]  9  2 10 12 14  3  5  6  4  1  7  8 11 13 15
```

# Funciones básicas sobre vectores

```
rank(y)
```

```
##  [1] 10.5  2.5  6.0  9.0  7.5  7.5 10.5 12.5  1.0
## [10] 2.5 12.5  4.5 14.0  4.5 15.0
```

# Funciones básicas sobre matrices y dataframes

Tomemos la siguiente matriz

```
vector <- c(1,2,3,4,4,3,2,1)
V <- matrix(vector,byrow=T,nrow=2)
V
```

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

```
dim(V)
```

```
## [1] 2 4
```

# Funciones básicas sobre matrices y dataframes

```
nrow(V)
```

```
## [1] 2
```

```
ncol(V)
```

```
## [1] 4
```

# Funciones básicas sobre matrices y dataframes

```
dimnames(V) <- list(c("ad","sd"),c("aa","bb","cc"," d"))
V
```

```
##    aa bb cc  d
## ad  1  2  3  4
## sd  4  3  2  1
```

names(), colnames(): nombres de columnas

rownames(): nombres de filas

# Funciones básicas

| Operation | Meaning |
|---|---|
| `max(x)` | maximum value in $x$ |
| `min(x)` | minimum value in $x$ |
| `sum(x)` | total of all the values in $x$ |
| `mean(x)` | arithmetic average of the values in $x$ |
| `median(x)` | median value in $x$ |
| `range(x)` | vector of min($x$) and max($x$) |
| `var(x)` | sample variance of $x$ |
| `cor(x,y)` | correlation between vectors $x$ and $y$ |
| `sort(x)` | a sorted version of $x$ |
| `rank(x)` | vector of the ranks of the values in $x$ |
| `order(x)` | an integer vector containing the permutation to sort $x$ into ascending order |
| `quantile(x)` | vector containing the minimum, lower quartile, median, upper quartile, and maximum of $x$ |
| `cumsum(x)` | vector containing the sum of all of the elements up to that point |
| `cumprod(x)` | vector containing the product of all of the elements up to that point |
| `cummax(x)` | vector of non-decreasing numbers which are the cumulative maxima of the values in $x$ up to that point |
| `cummin(x)` | vector of non-increasing numbers which are the cumulative minima of the values in $x$ up to that point |
| `pmax(x,y,z)` | vector, of length equal to the longest of $x$, $y$ or $z$, containing the maximum of $x$, $y$ or $z$ for the $i$th position in each |
| `pmin(x,y,z)` | vector, of length equal to the longest of $x$, $y$ or $z$, containing the minimum of $x$, $y$ or $z$ for the $i$th position in each |
| `colMeans(x)` | column means of dataframe or matrix $x$ |
| `colSums(x)` | column totals of dataframe or matrix $x$ |
| `rowMeans(x)` | row means of dataframe or matrix $x$ |
| `rowSums(x)` | row totals of dataframe or matrix $x$ |

# Aplicación de funciones en masa para matrices, dataframes y listas (apply())

La función apply(), aplica una función dada (con el argumento FUN) a todas la filas (MARGIN=1) o todas las columnas (MARGIN=2)

```
V <- matrix(vector,byrow=T,nrow=2)
V
```

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

```
apply(V,MARGIN = 1,FUN = mean)
```

```
## [1] 2.5 2.5
```

# Aplicación de funciones en masa para matrices, dataframes y listas (apply())

```
apply(V,MARGIN = 1,FUN = sum)
```

```
## [1] 10 10
```

# Hágalo usted mismo

Vamos a calcular la suma cuadrada de todas las filas de una matriz M de tamaño 5x2 con los número a bondad. Usando la función apply() sobre la filas de la matriz, se usará el argumento asociado FUN=function(x) {sum(x^2)}

# Función sweep()

Esta función es muy útil para realizar un barrido (en el sentido de una función por FUN) a cierto estadístico (dado por el argumento STATS), para todas las filas (MARGIN=1) o todas las columnas (MARGIN=2)

```
V
```

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

# Función sweep()

vamos a restar 3 de la fila 1 y 5 de la fila 2

```
sweep(V,MARGIN=1,STATS=c(3,5),FUN="-")
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   -2   -1    0    1
## [2,]   -1   -2   -3   -4
```

vamos a dividir las primeras dos columnas por 2 y las dos útilmas por 3

```
sweep(V,MARGIN=2,STATS=c(2,2,3,3),FUN="/")
```

```
##      [,1] [,2]      [,3]      [,4]
## [1,]  0.5  1.0 1.0000000 1.3333333
## [2,]  2.0  1.5 0.6666667 0.3333333
```