Introducción al software estadístico

Módulo III

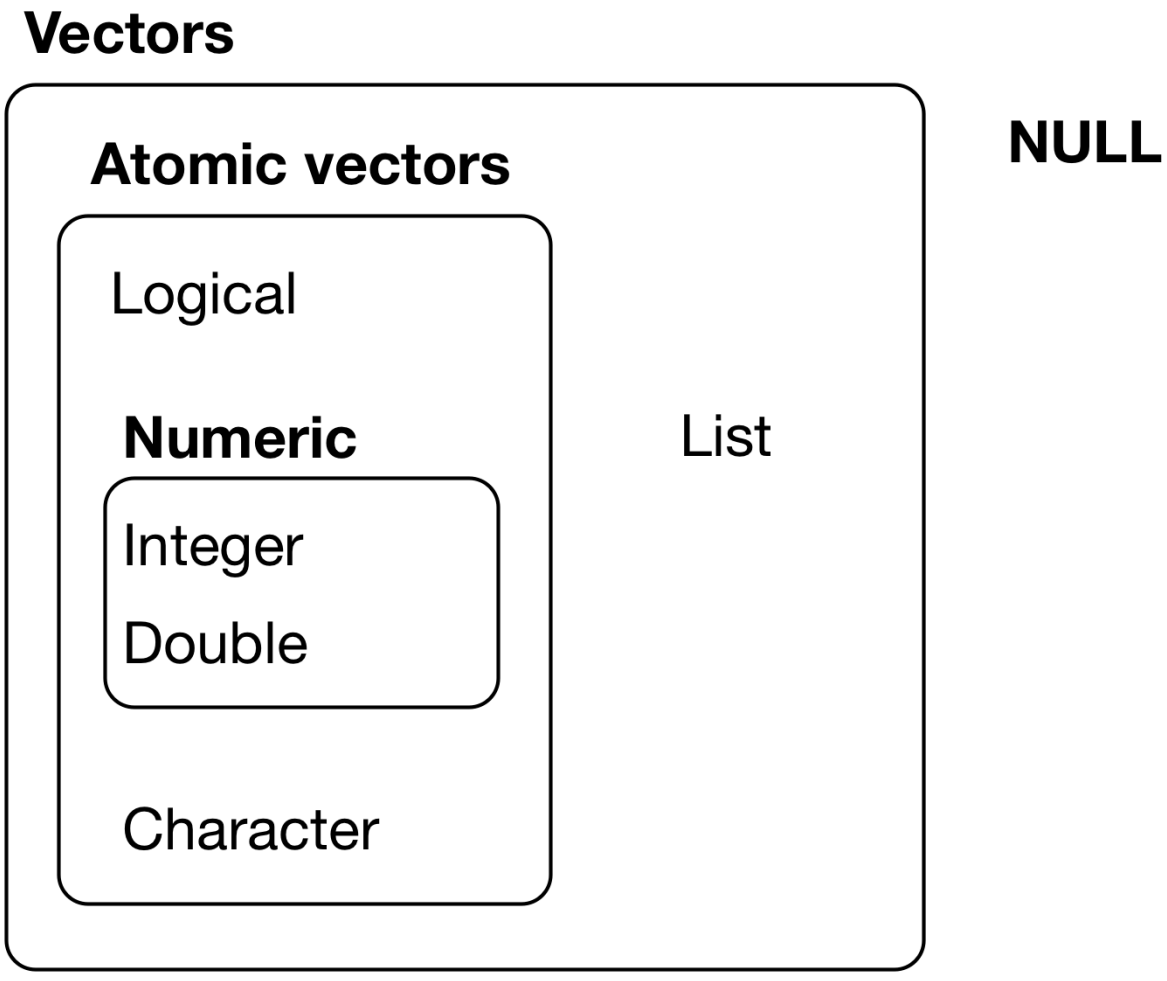
Nicolás Schmidt

nschmidt@cienciassociales.edu.uy

Departamento de Ciencia Política
Facultad de Ciencias Sociales
Universidad de la República

Estructura de la presentación

Estructura de la presentación



Fuente: **http://r4ds.had.co.nz**

Vectores

Hay dos tipos de vectores en R.

1. Atómicos
   - Los datos que componen el vector deben ser del mismo tipo.
2. Generales (listas)
   - Los datos que componen la lista pueden ser de distinto tipo.

Vectores atómicos

- La propiedad principal de un vector es el largo (`length()`)
- Los vectores comúnmente pueden ser de 4 tipos:
  1. `logical`
  2. `numeric-integer`
  3. `numeric-double`
  4. `character`
- La principal manera de crear un vector atómico de cualquier tipo es con la función '`c()`'. La 'c' es de concatenar. IMPORTANTE: es una 'c' minúscula.
- Es posible crear vectores vacíos de cualquier tipo (y veremos que son muy útiles!)
- Tipo especial de vector atómico: `factor()`

Creación de vectores

```
# opción 1: usando c()
mi.vector <- c(1, 3, 5)

# opción 2: usando un solo dato
mi.vector <- 1

# opción 3: en el caso de vectores numéricos usando funciones
mi.vector <- 1:10
mi.vector <- seq(1, 10, 1)
```

El operador ':' es un tipo especial de secuencia de valores. Esta secuencia puede hacer referencia a datos de distinto tipo, pero la secuencia debe ser una expresión numérica ya sea por los valores en sí o por la indexación. Cada elemento de la secuencia es igual al anterior más 1.

```
char_vector <- letters      # en R están cargadas las letras del abecedario
char_vector

##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"

char_vector[1:5]            # imprimo en pantalla los valores del 1 al 5 del objeto

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

Creación de vectores

```
x <- c(1, 3, 8, 10:15)
x
```

```
## [1]  1  3  8 10 11 12 13 14 15
```

```
vec1 <- 1:5
vec2 <- 5:1
vec3 <- c(20, 30)
vec4 <- c(8)
vec5 <- 8
vec6 <- c(vec1, vec2, vec3, vec4, vec5)
vec6
```

```
##  [1]  1  2  3  4  5  5  4  3  2  1 20 30  8  8
```

```
y <- c(1, c(2, c(3)))
y
```

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

Pregunta: ¿al correr estas dos lineas qué se obtiene?

```
c <- c("c")
c <- c(c)
```

Vectores Lógicos

En R hay tres posible valores lógicos:

1. `TRUE`, se puede escribir en modo abreviado como `T` (No es recomendable!)
2. `FALSE`, se puede escribir en modo abreviado como `F` (No lo recomendable!)
3. `NA`, dato faltante, not available.

```
vec_logical <- c(TRUE, FALSE, TRUE)
vec_logical <- c(T, F, T)
vec_logical <- c(TRUE, FALSE, TRUE, F, F, T, T)
vec_logical

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

Importante: el valor lógico `TRUE` y `FALSE` tienen una expresión numérica que asume el valor 1 y 0 respectivamente. Esto permite realizar operaciones matemáticas con datos de tipo `logical`.

```
sum(vec_logical)

## [1] 4
```

NA

```
vec_logical <- c(TRUE, FALSE, TRUE, NA)
sum(vec_logical)                          # NA no tiene expresión numérica

## [1] NA

is.na(vec_logical)                        # devuelve una sentencia lógica de T o F

## [1] FALSE FALSE FALSE  TRUE

sum(is.na(vec_logical))                   # cantidad de NA's en el vector

## [1] 1
```

Vectores Lógicos: por qué no usar `T` o `F`

Ejemplo:

```
T <- F

rep(T, 5)

## [1] FALSE FALSE FALSE FALSE FALSE

TRUE <- FALSE # son palabras reservadas!

## Error in TRUE <- FALSE: invalid (do_set) left-hand side to assignment

rep(TRUE, 5)

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

Vectores numéricos

En R hay dos tipos de vectores numéricos:

1. `integer`, número entero que se puede forzar su tipo con la letra 'L'.
2. `double`, número ni entero ni complejo.

```
vec_entero <- c(1L, 3L, 5L)
is.integer(vec_entero)

## [1] TRUE

is.numeric(vec_entero)

## [1] TRUE
```

```
vec_double <- c(1.2, 1.5)
is.double(vec_double)

## [1] TRUE

is.numeric(vec_double)

## [1] TRUE
```

Vectores de caracteres

En R una cadena de caracteres debe ir entre comillas. Todo lo que sea una letra pero no esté entre comillas R lo va a interpretar como una llamada a un objeto.

```
vec_cha1 <- c("a", "b")
typeof(vec_cha1)

## [1] "character"

LETTERS

##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
## [18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"

letters

##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"

is.character(letters)

## [1] TRUE
```

Coerciones implícitas

Cuando se intenta combinar en un vector dos tipos de datos R resuelve sobre el tipo final que tendrá el vector de acuerdo a una jerarquía basada en la flexibilidad del dato.

Ejemplo

```
typeof(vec.1 <- c(5.2, 3.8))

## [1] "double"

typeof(vec.2 <- c(vec.1, "Hola"))

## [1] "character"
```

typeof() resultante de concatenar datos de distinto tipo

	logical	integer	double	character
logical	logical			
integer	ineger	integer		
double	double	double	double	
character	character	character	character	character

NA

`NA` es un tipo de dato `logical`. Pero como las coerciones implícitas se hacen según la flexibilidad del tipo de dato el NA se adapta el tipo real del vector siempre.

La coerción siempre se fuerza hacia el tipo más flexible (de menor a mayor este es el orden: `logical`, `integer`, `double`, `character`).

Ejemplo

```
typeof(c(5.2, NA))

## [1] "double"

typeof(c(NA, "Hola"))

## [1] "character"
```

length()

La longitud de un vector es su principal propiedad. No solo reporta información sobre el tamaño del vector sino que éste dato va a permitir realizar operaciones con el.

```
vec <- c(1:50, 70:91, 8, 83, 79)
vec
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
## [47] 47 48 49 50 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88
## [70] 89 90 91  8 83 79
```

```
length(vec)
```

```
## [1] 75
```

```
sum(vec)/length(vec)        # promedio
```

```
## [1] 42.88
```

‘ [’

La forma de acceder a uno o más datos de un vector es usando el operador ‘ [ ] ’. Se puede hacer de múltiples formas:

```
vec <- c(5, 10, 15, 20, 25)
vec[1]                          # el valor que esta en el primer lugar del vector

## [1] 5

vec[1:3]                        # los valores desde la primera a la tercera posición

## [1]  5 10 15

vec[c(1, 2, 3)]                 # idem anterior

## [1]  5 10 15

vec[-2]                         # todos los valores menos el de la segunda posición

## [1]  5 15 20 25

vec[]                           # devuelve el vector completo

## [1]  5 10 15 20 25
```

'['

Error:

```
vec[c(-2, 3)]                 # no se puede excluir e incluir de manera conjunta

## Error in vec[c(-2, 3)]: only 0's may be mixed with negative subscripts
```

Selección por fuera del largo del vector:

```
vec[8]                        # Si la indexación excede el largo devuelve un NA

## [1] NA

v <- vec[3:8]                 # cuidado!, esto modifica el vector! (usar funciones!)
v

## [1] 15 20 25 NA NA NA
```

Indexación lógica

```
vec[c(TRUE, FALSE, TRUE, FALSE, TRUE)]

## [1]  5 15 25
```

'['

Modificar valores

```
vec <- c(10, 20, 20, 40, 50, 60)
vec

## [1] 10 20 20 40 50 60

vec[3] <- 30; vec

## [1] 10 20 30 40 50 60

vec[1:2] <- NA; vec

## [1] NA NA 30 40 50 60
```

Modificar valores indexando con funciones y operaciones

```
vec[1:(length(vec)/2)] <- 1
vec

## [1]  1  1  1 40 50 60
```

Vectores vacíos

En ocasiones es probable que se necesite crear un vector vacío pero de un *tipo* específico.

El procedimiento consiste en crear un vector con las características deseadas e ir ingresando datos.

Ejemplo:

```
vector.cha <- integer() ; vector.cha2 <- vector("integer")
vector.num <- numeric() ; vector.num2 <- vector("numeric")
vector.log <- logical() ; vector.log2 <- vector("logical")

vector.num ; vector.num2

## numeric(0)
## numeric(0)

for(i in 1:5){
    vector.num[i] <- i
    }
vector.num

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

Estructura de la presentación

Reciclaje

Si se realizan operaciones con dos vectores (con `length()` mayor o igual a 1) que no tienen la misma longitud se generará un *reciclaje* hacia el vector de mayor longitud.

Ejemplo:

```
uno <- rep(1, 10)
dos <- rep(2, 5)
(tres <- uno + dos)

##  [1] 3 3 3 3 3 3 3 3 3 3

length(uno)

## [1] 10

length(dos)

## [1] 5

length(tres)

## [1] 10
```

Operadores numéricos

Operador	Descripción
-	Resta, puede ser unitario o binario
+	Suma, puede ser unitario o binario
*	Multiplicación, binaria
/	División, binaria
^	Potencia, binario
%%	Módulo, binario
%/%	División entera, binario

Operadores numéricos

Ejemplos:

```
j <- c(10, 20, 20, 30, 20, 10)
j/2
```

```
## [1]  5 10 10 15 10  5
```

```
12.3 %% 3
```

```
## [1] 0.3
```

```
12.3 %/% 3
```

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

```
10*5^2
```

```
## [1] 250
```

```
(10*5)^2
```

```
## [1] 2500
```

Funciones de redondeo

`ceiling()`	Redondea hacia arriba siempre
`floor()`	Redondea hacia abajo siempre
`round()`	Redondeo en .5

Ejemplos:

```
ceiling(30/22)

## [1] 2

floor(30/22)

## [1] 1

round(30/22, digits = 3)

## [1] 1.364
```

Operadores lógicos

Operador	Descripción
`!x`	Negación de x
`x & y`	x AND y, devuelve un vector
`x && y`	x AND y, devuelve un solo valor
`x | y`	x OR y, devuelve un vector
`x || y`	x OR y, devuelve un solo valor
`xor(x, y)`	OR exclusivo
`x%in%y`	x IN y (es distinto que y IN x)
`x < y`	x menor que y
`x > y`	x mayor que y
`x <= y`	x menor o igual que y
`x >= y`	x mayor o igual que y
`x == y`	x igual que y
`x != y`	x distinto de y

Ejemplos de operadores lógicos

```
k <- c(6, 8, 10, 12, 14, 16, 18, 20)
j <- c(10, 20, 20, 30, 20, 10)
k %in% j
```

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

```
k[k %in% j == TRUE]
```

```
## [1] 10 20
```

```
k[k == 8]
```

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

```
k[k %% 3 != 0]
```

```
## [1]  8 10 14 16 20
```

```
p <- seq(-2, 2, .3)
abs(p) < 1.5
```

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

Funciones lógicas relevantes

Función	Descripción
`isTRUE(x)`	Devuelve TRUE si todos los valores de x son TRUE
`all()`	Devuelve TRUE si todos los argumentos son TRUE
`any()`	Devuelve TRUE si al menos un argumento es TRUE
`identical(x, y)`	Compara dos objetos y devuelve TRUE si son iguales
`is.na()`	Devuelve TRUE donde hay un NA
`is.null()`	Devuelve TRUE si es NULL
`is.nan()`	Devuelve TRUE si es NaN 'Not a Number'

Funciones lógicas relevantes

Ejemplos:

```
identical(1, 1.)
```

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

```
identical(1, 1L)
```

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

```
w <- c(1, 2, 3, NA, 5, NA, 7)
is.na(w)
```

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

```
is.nan(0/0)
```

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

which()

La función 'which()' sirve para localizar la ubicación de los elementos del vector que cumplen una determinada condición:

```
x <- c(1, 5, 7, 8, 10, 12, 3, 23, 50)
y <- which(x > 10)
x[y]

## [1] 12 23 50

x[x > 10]                    # para casos simples no siempre es mejor usar which()

## [1] 12 23 50

z <- c("a", "b", "c", "a", "a")
which(z == "a")

## [1] 1 4 5
```

Asimismo, hay dos funciones de la familia 'which' que son de utilidad. which.min() y which.max() que localizan el valor mínimo y máximo, respectivamente dentro de un vector.

Operadores con cadena de caracteres

En el caso de vectores de tipo `character()` se utilizan expresiones regulares.

Expresión	Descripción
`^`	Inicio de cadena
`$`	Fin de cadena
`.`	Cualquier carácter
`.{n}`	Cualquier cadena de caracteres de longitud n
`[ch1 - ch2]`	Rango de caracteres desde ch1 hasta ch2
`[ch1, ch2, ch3]`	Conjunto de caracteres ch1, ch2 y ch3

Funciones con vectores de caracteres

Función	Descripción
`nchar(x)`	Número de caracteres
`paste()`	Concatena los objetos y devuelve una cadena de caracteres
`substr()`	Extrae una cadena de un vector de caracteres
`strtrim()`	Elimina los espacios de un vector de caracteres
`strsplit()`	Divide los objetos de un vector de caracteres utilizando un carácter como delimitador
`grep()`	Busca coincidencias con un patrón dentro de un vector de caracteres
`grepl()`	Busca coincidencias con un patrón dentro de un vector de caracteres y devuelve un vector lógico
`agrep()`	Similar a grep(), busca coincidencias aproximadas
`gsub(p, r, v)`	Reemplaza todas las ocurrencias de *p* (patrón) por *r* (remplazo) en *v* (vector de caracteres)
`sub(p, r, v)`	Reemplaza la primer ocurrencia de *p* (patrón) por *r* (remplazo) en *v* (vector de caracteres)
`tolower(x)`	Convierte x a minúsculas
`toupper(x)`	Convierte x a mayúsculas
`noquote(x)`	Imprime un vector de caracteres sin comillas

Funciones con vectores de caracteres

Ejemplos:

```
nchar("Hola mundo")

## [1] 10

(letras <- letters[1:10])

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

gsub("a", "Hola", letras)

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

gsub("[a-d]", "Hola", letras)

##  [1] "Hola" "Hola" "Hola" "Hola" "e"    "f"    "g"    "h"    "i"    "j"

gsub("[a, c, e]", "Hola", letras)

##  [1] "Hola" "b"    "Hola" "d"    "Hola" "f"    "g"    "h"    "i"    "j"
```

Ejemplos:

```
ch <- c("Partido A", "Partido B", "Partido C")
gsub("^Partido", "Party" , ch)

## [1] "Party A" "Party B" "Party C"

grep("ido.", ch)

## [1] 1 2 3

colors()[grep("^blue", colors())]

## [1] "blue"       "blue1"      "blue2"      "blue3"      "blue4"
## [6] "blueviolet"

colors()[grep("red$", colors())]

## [1] "darkred"         "indianred"       "mediumvioletred" "orangered"
## [5] "palevioletred"   "red"             "violetred"

colors()[grep("^[y,z]", colors())]

## [1] "yellow"      "yellow1"     "yellow2"     "yellow3"     "yellow4"
## [6] "yellowgreen"
```

Funciones básicas

Función	Descripción
`sum(x)`	Suma los elementos de x
`prod(x)`	Producto de los elementos de x
`cumsum(x)`	Suma acumulativa de los elementos de x
`cumprod(x)`	Producto acumulativo de los elementos de x
`min(x)`	Valor mínimo de x
`max(x)`	Valor máximo de x
`mean(x)`	Media de x
`median(x)`	Mediana de x
`var(x)`	Varianza de x
`sd(x)`	Desviación estándar de x
`cov(x,y)`	Covarianza de x, y
`cor(x,y)`	Correlación de x, y
`range(x)`	Rango de x. Devuelve un vector con el min() y max().
`quantile(x)`	Cuantiles de x
`fivenum(x)`	Resumen de cinco números de x
`unique(x)`	Elementos únicos de x
`rev(x)`	Orden inverso de los elementos de x
`sort(x)`	Ordena los elementos de x
`match(x,v)`	Primera posición de un elemento de x con valor v
`union(x,y)`	Unión de x, y
`intersect(x,y)`	Intersección de x, y
`setdiff(x,y)`	Elementos de x que no están en y
`setequal(x,y)`	Indica si ambos vectores tienen los mismos elementos

Funciones básicas

Ejemplos:

```
x <- 1:8
y <- 4:12

prod(x)

## [1] 40320

cumsum(y)

## [1]  4  9 15 22 30 39 49 60 72

unique(c(x, y))

##  [1]  1  2  3  4  5  6  7  8  9 10 11 12

intersect(x, y)

## [1] 4 5 6 7 8

rev(y)

## [1] 12 11 10  9  8  7  6  5  4
```

rep()

```
rep(x, times = 1, length.out = NA, each = 1)
```

`x`	valor/es a replicar (repetir)
`times`	entero positivo que determina la cantidad de veces que se va a repetir `x`
`length.out`	repetir `x` hasta un largo (`length.out()`) determinado.
`each`	repetir `x` o cada elemento de `x` `each` veces.

```
rep(x = 1:4, times = 2)

## [1] 1 2 3 4 1 2 3 4

rep(x = c("A", "B", "C"), times = c(3,2,1))

## [1] "A" "A" "A" "B" "B" "C"

rep(x = 1:4, each = 2, length.out = 4)

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

seq()

```
seq(from = 1, to = 1, by = ((to-from)/(length.out-1)),
length.out = NULL, ...)
```

`from`	valor de inicio de la secuencia
`to`	valor final de la secuencia
`by`	incremento de la secuencia
`length.out`	realizar la secuencia hasta un largo (`length.out()`) determinado.

```
seq(from = 0, to = 1, length.out = 5)
```

```
## [1] 0.00 0.25 0.50 0.75 1.00
```

```
seq(1, 18, by = 3)
```

```
## [1]  1  4  7 10 13 16
```

```
seq(20)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

paste()

```
paste (..., sep = " ", collapse = NULL)
```

`...`	uno o mas objetos para concatenar en modo ‘`characeter`’
`sep`	separador entre valores de los objetos
`collapse`	colapsa los valores de los objetos

```
a <- 1:5
b <- letters[1:5]
paste(a, b)

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

paste(a, b, sep = "_")

## [1] "1_a" "2_b" "3_c" "4_d" "5_e"

paste(a, "Hola", sep = "")

## [1] "1Hola" "2Hola" "3Hola" "4Hola" "5Hola"

paste(b, collapse = "-")

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

ifelse()

`ifelse(test, yes, no)`

`test`	una sentencia lógica
`yes`	valor a asignar si la sentencia (`test`) es `TRUE`
`no`	valor a asignar si la sentencia (`test`) es `FALSE`

```
vec <- 1:10
ifelse(vec == 2, 100, 0)

##  [1]   0 100   0   0   0   0   0   0   0   0

ifelse(vec != 5, NA, vec)

##  [1] NA NA NA NA  5 NA NA NA NA NA

ifelse(vec %% 2 == 0, vec^2, sqrt(vec))

##  [1]   1.000000   4.000000   1.732051  16.000000   2.236068  36.000000
##  [7]   2.645751  64.000000   3.000000 100.000000
```

Vectores aleatorios

En R hay dos maneras muy simples de generar vectores aleatorios.

1. usando la función `sample()`
2. generado números aleatorios siguiendo determinada distribución.

Cada vez que se ejecute una funciones que genera números aleatorios se inicia un nuevo proceso aleatorio. Si se desea obtener siempre el mismo vector aleatorio (fundamentalmente para replicar una tarea) es necesario usar la función `set.seed()`. Esta función tiene el objetivo de establecer un punto de inicio del proceso aleatorios que siempre será igual si la semilla (`seed`) es la misma.

sample()

```
sample(x, size, replace = FALSE, prob = NULL)
```

x	vector sobre el que se va a sacar una muestra
replace	valor lógico, seleccionar con remplazamiento (sample(TRUE)) o sin remplazamiento (sample(FALSE)).
prob	un vector de pesos de probabilidad

```
sample(x = 1:10)
```

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

```
sample(x = 1:10, size = 10, replace = TRUE)
```

```
##  [1]  8  1  4  1  1  7  4 10  6  9
```

```
sample(x = c("cara", "seca"), size = 10, replace = TRUE)
```

```
##  [1] "seca" "seca" "seca" "cara" "seca" "cara" "cara" "seca" "cara" "cara"
```

```
sample(x = c(1, 0), size = 10, replace = TRUE, prob = c(0.9, 0.1))
```

```
##  [1] 1 1 1 1 1 1 1 0 1 1
```

set.seed()

Ejemplo:

```
sample(x = 1:10)

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

sample(x = 1:10)

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

set.seed(2018)              # puede ser cualquier valor, yo puse el año corriente
sample(x = 1:10)

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

set.seed(2018)
sample(x = 1:10)

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

Distribuciones en el paquete base

Distribución	Función
Normal	`rnorm(n, mean = 0, sd = 1)`
Uniforme	`runif(n, min = 0, max = 1)`
Exponencial	`rexp(n, rate = 1)`
Gamma	`rgamma(n, shape, scale = 1)`
Poisson	`rpois(n, lambda)`
Weibull	`rweibull(n, shape, scale = 1)`
Cauchy	`rcauchy(n, location = 0, scale = 1)`
Beta	`rbeta(n, shape1, shape2)`
t de Student	`rt(n, df)`
F (Snedecor)	`rf(n, df1, df2)`
Pearson χ^2	`rchisq(n, df)`
Binomial	`rbinom(n, size, prob)`
Geométrica	`rgeom(n, prob)`
Hipergeométrica	`rhyper(nn, m, n, k)`
Logística	`rlogis(n, location = 0, scale = 1)`
Lognormal	`rlnorm(n, meanlog = 0, sdlog = 1)`
Binomial negativa	`rnbinom(n, size, prob)`

$\Rightarrow$ la letra '`r`' al inicio de cada función hace referencia a random.

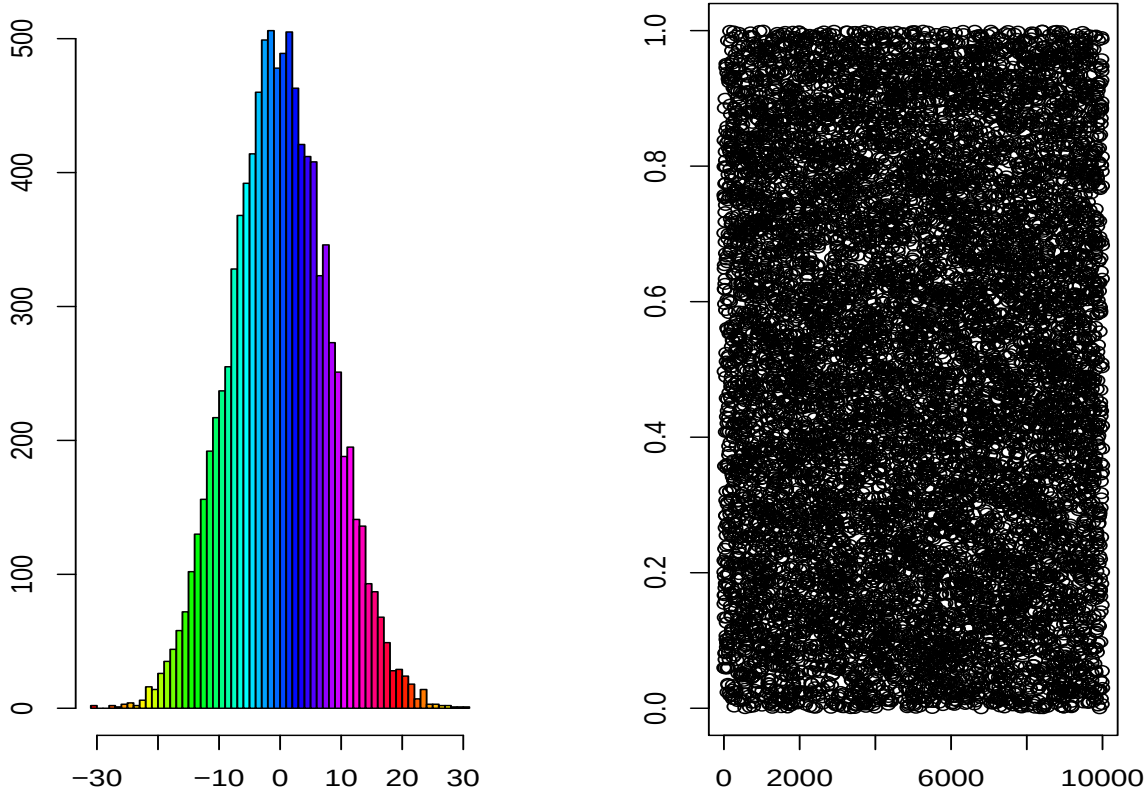
Distribuciones de probabilidad en el paquete base

Ejemplo:

```
rnorm(5)

## [1] -0.2647112  2.0994707  0.8633512 -0.6105871  0.6370556

rnorm(5, mean = 5, sd = 2)

## [1] 3.713931 2.939943 6.424963 4.108456 5.497959

runif(5)

## [1] 0.14136788 0.86259901 0.03383022 0.49020482 0.50618012

rexp(5)

## [1] 0.5240688 3.2463786 3.4421388 0.1620119 1.7873534
```

Distribuciones de probabilidad en el paquete base

```
par(mfrow = c(1,2))
hist(rnorm(10000, sd=8), col=rainbow(50), main="", breaks = 50, ylab="", xlab="")
plot(runif(10000), main="", ylab="", xlab="")
```



Estructura de la presentación

Factores

- Un factor es un vector que contienen un número fijo de categorías (datos cualitativos).
- La diferencia con los vectores de tipo `character` es la manera en la que almacenan la información.
- Los vectores de caracteres guardan todas la información (letra por letra), mientras que los factores asignan un número a cada categoría. Cada uno de estos números establece los niveles del factor.
- El atributo (`attributes()`) principal de un factor son sus niveles.
- Los factores son especialmente útiles para utilizar en modelos estadísticos.

Creación de un factor

- Se puede crear un factor usando la función `factor()`
- Se puede convertir un vector con `as.factor()`
- Se puede generar un factor ordenado con `ordered()`
- Usando la función `cut()`, transforma un vector numérico a uno de tipo factor
- con la función `gl()` (*Generate Factor Levels*)

Creación de un factor

Ejemplo:

```
(sexo <- sample(c("F", "M"), 10, replace = TRUE))

##  [1] "M" "M" "F" "M" "F" "F" "F" "F" "M" "F"

class(sexo)

## [1] "character"

(sexo <- as.factor(sexo))

##  [1] M M F M F F F F M F
## Levels: F M

levels(sexo)

## [1] "F" "M"
```

Creación de un factor

Ejemplo:

```
(fac_letras <- factor(sample(letters[1:5], 20, replace = TRUE)))

##  [1] a a a c e e e b b e c d a b d a c e c a
## Levels: a b c d e

(fac_letras <- factor(fac_letras, levels = rev(levels(fac_letras))))

##  [1] a a a c e e e b b e c d a b d a c e c a
## Levels: e d c b a

(fac_cut  <- cut(rep(1, 10), breaks = 5))

##  [1] (0.9998,1.0002] (0.9998,1.0002] (0.9998,1.0002] (0.9998,1.0002]
##  [5] (0.9998,1.0002] (0.9998,1.0002] (0.9998,1.0002] (0.9998,1.0002]
##  [9] (0.9998,1.0002] (0.9998,1.0002]
## 5 Levels: (0.999,0.9994] (0.9994,0.9998] ... (1.0006,1.001]

as.numeric(fac_letras)      # convertir a numeric un factor se imprimen los niveles

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

Creación de un factor

Ejemplo:

```
paises <- c("Argentina", "Uruguay", "Brasil", "Chile", "Uruguay", "Brasil",
            "Chile","Argentina", "Uruguay", "Brasil", "Chile")

(paises2 <- factor(paises))

##  [1] Argentina Uruguay   Brasil    Chile     Uruguay   Brasil    Chile
##  [8] Argentina Uruguay   Brasil    Chile
## Levels: Argentina Brasil Chile Uruguay

levels(paises2) <- c("Uruguay", "Chile", "Brasil", "Argentina")
paises2

##  [1] Uruguay   Argentina Chile     Brasil    Argentina Chile     Brasil
##  [8] Uruguay   Argentina Chile     Brasil
## Levels: Uruguay Chile Brasil Argentina

gl(4, 3, 12, labels = c("Uruguay", "Chile", "Brasil", "Argentina"))

##  [1] Uruguay   Uruguay   Uruguay   Chile     Chile     Chile     Brasil
##  [8] Brasil    Brasil    Argentina Argentina Argentina
## Levels: Uruguay Chile Brasil Argentina
```

Función drop() aplicada a factores

Ejemplo:

```
f <- factor(c("A", "B", "C"))
f

## [1] A B C
## Levels: A B C

f[1]

## [1] A
## Levels: A B C

f[1, drop = TRUE]

## [1] A
## Levels: A

f2 <- as.factor(as.character(f[1]))
f2

## [1] A
## Levels: A
```