Introducción a R

Arturo Pérez

29/1/2021

1. Conociendo la interfaz

1. Editor de código fuente
2. Consola
3. Historial y ambiente de trabajo
4. Ventana de archivos, gráficas, paquetes, ayuda, etc.

Diferencias con la interfaz de R: Es más ordenada y es más cómodo trabajar desde RStudio.

Iniciando un proyecto en RStudio

1. File/New Project.
2. Símbolo de "create project"
3. Click en nombre del proyecto y click en new project

2. Instalando paquetes

CRAN

```
install.packages("ggplot2") library(ggplot2)
```

GitHub

```
install.packages("devtools") library(devtools) install_github("author/package")
```
#github username/ nombre del paquete

Bioconductor

```
install.packages("biocmanager") library(biocmanager) install("nombre del paquete")
```

3. Clases y tipos de objetos

Todo lo que se manipula en R son objetos con diferentes tipos (o clases) de datos.

Console input

```r
x <- 2+2
print(x)
```

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

```r
x
```

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

```r
msg <- "Intersemestral de R"
msg
```

```
## [1] "Intersemestral de R"
```

Las 5 clases atómicas de los objetos en R son las siguientes:

1. Caracter

```r
x <- "hola" ##cualquier cosa
class(x)
```

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

2. Numérico

```r
y <- 3.14
class(y)
```

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

3. Integrer (números enteros)

```r
y <- 1:3
t <-  4L
class(y)
```

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

```r
class(t)
```

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

4. Complejo

```r
x <- 1+3i
class(x)
```

```
## [1] "complex"
```

5. Logico(TRUE/FALSE)

```r
z <- 0>1
z
```

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

```r
class(z)
```

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

Atributos de los objetos

Pueden formar parte de los objetos en R 1. Nombres, nombres de dimensiones 2. dimensiones (p.e matrices o tablas de datos) 3. Class (todos los objetos pertenecen a una clase) 4. Lenght (largo del objeto o comunmente vectores) 5. otros

`attributes()` : Permite acceder a los atributos de un objeto en R.

```r
x <- 1
names(x) <- "juan"
attributes(x)
```

```
## $names
## [1] "juan"
```

```r
x
```

```
## juan
##    1
```

4. Vectores y listas

Vectores

Son el objeto más básico en R. Se crean utilizando las siguientes funciones

`c()`: Concatena una serie de objetos

```r
x <- c(1,2,3,4,5,6,7,8,9,10)
class(x)
```

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

`vector()`: Crea un vector de cierto tipo y cierto largo

```r
y <- vector("numeric", 10)
z <- vector("logical", 10)
y
```

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

En un vector únicamente se pueden almacenar objetos de una misma clase. Cuando intentamos combinar objetos de diferentes clases, todos los objetos dentro del vector se vuelven de una misma clase según el denominador menos común

```
class(c(1.7, "a"))
```

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

```
class(c(TRUE, 2))
```

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

```
class(c("a", TRUE))
```

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

Coerción específica

Los objetos pueden cambiar de clase usando al función `as.`

```
x <- c(0,1,2,3,4,5,6)
class(x)
```

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

```
x <- as.logical(x)
x
```

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

```
class(x)
```

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

```
x <- as.character(x)
x
```

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

```
class(x)
```

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

```
##NO SIEMPRE FUNCIONA
x <- c("a", "b", "c")
as.numeric(x)
```

```
## Warning: NAs introducidos por coerción
```

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

listas

Son un tipo de vector con elementos de distintas clases

```
y <- list(1:3, "a", 3)
y
```

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

varios elementos en diferentes listas

```
c <- list(c(1:4), c("a", "d", "v"))
c
```

```
## [[1]]
## [1] 1 2 3 4
##
## [[2]]
## [1] "a" "d" "v"
```

```
class(c)
```

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

5. Matrices y tablas de datos

Matrices

Las matrices son otros de los tantos tipos de objetos que existen en R. Se crean utilizando la función `matrix()`

```
m <- matrix(c(1,4,5,7,4,0), nrow = 2, ncol= 3)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    4
## [2,]    4    7    0
```

Creando matrices utilizando un vector con la función `dim()`

```
m <- 1:10
dim(m) <- c(2,5)
m
```

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

Creando matrices uniendo vectores con las funciones `cbind()` `rbind()`

```
x <- 1:3
y <- 10:12
matriz1 <- cbind(x,y) #datos en diferentes columnas
matriz2 <- rbind(x,y) #datos en diferentes filas
```

Tablas de datos ("DATA.FRAMES")

Las tablas de datos son el objeto clave utilizado en R donde se almacena información tabulares. Se representan como un tipo especial de lista donde *cada elemento de la lista tiene el mismo largo*:

```
-Columnas: Es el elemento
-Filas: Es el largo de la lista.
```

Las columnas no precisamente tienen que ser de la misma clase (principal diferencia con una matriz). Las columnas guardan diferentes clases de objetos en cada columna.

Atributos de las tablas de datos

Nombre de las filas y las columnas. `rownames()` y `colnames()`

```
## FILAS
data <- mtcars
head(rownames(data))
```

```
## [1] "Mazda RX4"         "Mazda RX4 Wag"     "Datsun 710"
## [4] "Hornet 4 Drive"    "Hornet Sportabout" "Valiant"
```

```
head(data)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
data2 <- airquality
head(rownames(data2))
```

```
## [1] "1" "2" "3" "4" "5" "6"
```

```
head(data2)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

```
## COLUMNAS
names(data)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

```
colnames(data)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

Se puede crear una matriz de una tabla de datos con la función `data.matrix()` o con la función `as.matrix()`. Sin embargo, los elementos dentro de las tablas se verán forzados a convertirse a elementos de una misma clase.

```
x <- data.matrix(data)
head(x)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
class(x)
```

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

```
y <- airquality
y <- as.matrix(y)
head(y)
```

```
##      Ozone Solar.R Wind Temp Month Day
## [1,]    41     190  7.4   67     5   1
## [2,]    36     118  8.0   72     5   2
## [3,]    12     149 12.6   74     5   3
## [4,]    18     313 11.5   62     5   4
## [5,]    NA      NA 14.3   56     5   5
## [6,]    28      NA 14.9   66     5   6
```

```r
class(y)
```

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

crear una tabla de datos

Se crean utilizando la función `data.frame()` y cada columna representaría un vector con objetos de una misma clase.

```r
z <- data.frame(columna1=1:10,
                variable2=c(T,T,F,T,F,T,T,T,T,F))
z
```

```
##    columna1 variable2
## 1         1      TRUE
## 2         2      TRUE
## 3         3     FALSE
## 4         4      TRUE
## 5         5     FALSE
## 6         6      TRUE
## 7         7      TRUE
## 8         8      TRUE
## 9         9      TRUE
## 10       10     FALSE
```

```r
nrow(z)
```

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

```r
ncol(z)
```

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

Otros objetos dentro de una tabla de datos

Como ya vimos, dentro de una tabla de datos podemos almacenar diferentes clases de objetos, sin embargo, existen objetos particulares dentro de las tablas, los factores y los valores perdidos.

factores Se trata de un tipo especial de vector para expresar datos categóricos. Muy útiles para crear grupos. Es mucho mejor que utilizar vectores **integrers** porque los factores se autodescriben. P.E, es mejor tener *macho* y *hembra* que *1* y *2*.

Se crean con la función `factor()`.

```r
x <- factor(c("macho", "hembra", "hembra", "hembra", "macho"))
table(x)
```

```
## x
## hembra  macho
##      3      2
```

```r
class(x)
```

```
## [1] "factor"
```

Quitar la clase del vector.

```r
unclass(x)
```

```
## [1] 2 1 1 1 2
## attr(,"levels")
## [1] "hembra" "macho"
```

El orden de los niveles de los factores se puede establecer utilizando el argumento `levels` dentro de la función `factor()`. P.E en modelos lineales el primer nivel puede ser la línea base.

```r
x <- factor(x, levels = c("macho", "hembra"))
x
```

```
## [1] macho  hembra hembra hembra macho
## Levels: macho hembra
```

Sin embargo, supongamos que en el data frame tenemos dos grupos indicados como 1 y 2 para hombres y mujeres, entonces hacemos uso del argumento `labels` para ponerle nombre a cada uno de los grupos.

```r
x <- c(rep(c(1,2), each= 10))
x
```

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

```r
x <- factor(x, labels = c("mujeres", "hombres"))
x
```

```
##  [1] mujeres mujeres mujeres mujeres mujeres mujeres mujeres mujeres mujeres
## [10] mujeres hombres hombres hombres hombres hombres hombres hombres hombres
## [19] hombres hombres
## Levels: mujeres hombres
```

valores perdidos En R son denominados como **NA** o **NaN**.

`is.na()` se utiliza para ver si hay valores perdidos o faltantes en un objeto

`is.nan()` evalua NaN, es decir cuando tenemos una operación aritmética indefinida o sin sentido

los valores **NA** también tienen una clase: Integrer NA, character NA, numeric NA, etc.

Un valor NaN es un valor NA, es decir, puede haber un valor NaN perdido y te sale NA, sin embargo al inverso no es posible

```r
x <- c(1,2,NA,NA,NA,NA,3,4,5,6)
is.na(x)
```

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

```r
is.nan(x)
```

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

```r
x <- c(1,2,NaN,NaN,NA,NA,3,4,5,6)
is.na(x)
```

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

```r
is.nan(x)
```

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

Atributo "NOMBRES"

Como vimos anteriormente, podemos nombrar objetos en R. Esto facilita la lectura del código porque de esta forma, los objetos se autodescriben. Se utiliza la función `names()` para asignarle nombre a los objetos

```r
x <- 1:3
names(x)
```

```
## NULL
```

```r
names(x) <- c("objeto", "cosa", "item")
x
```

```
## objeto   cosa   item
##      1      2      3
```

```r
names(x)
```

```
## [1] "objeto" "cosa"   "item"
```

Las listas también pueden tener nombres

```r
x <- list(a=1:5, b=c(3,4,5,6,7), c=c("a", "b", "c", "d", "f"))
x
```

```
## $a
## [1] 1 2 3 4 5
##
## $b
## [1] 3 4 5 6 7
##
## $c
## [1] "a" "b" "c" "d" "f"
```

También se pueden asignar nombres a las matrices y tablas de datos con las funciones `dimnames()`,`rownames()`,`colnames()`.

```r
m <- matrix(sample(1:5),nrow = 6, ncol = 5)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    4    5    2    3    1
## [2,]    5    2    3    1    4
## [3,]    2    3    1    4    5
## [4,]    3    1    4    5    2
## [5,]    1    4    5    2    3
## [6,]    4    5    2    3    1
```

```r
dimnames(m) <- list(c("a","b","c", "d", "e", "f"), c("col1", "col2", "col3", "col4", "col5"))
m
```

```
##   col1 col2 col3 col4 col5
## a    4    5    2    3    1
## b    5    2    3    1    4
## c    2    3    1    4    5
## d    3    1    4    5    2
## e    1    4    5    2    3
## f    4    5    2    3    1
```

```r
## añadir nombres utilizando las otras dos funciones
m <- matrix(rnorm(30), nrow = 6, ncol = 5)
m
```

```
##            [,1]       [,2]       [,3]       [,4]        [,5]
## [1,]  0.6979846  0.5060788  1.7877092 -0.3492734 -0.50819389
## [2,]  2.1130557 -1.1417384 -1.2114209  0.9414483  0.70277541
## [3,] -0.1591949  1.4796209 -1.1032868 -1.8258291  1.84210970
## [4,]  0.6857158  1.0128991 -0.0889729  2.7984764  2.50704205
## [5,] -0.6059189  0.1463889  2.7370227  1.5480494  0.06815174
## [6,]  0.5777231  0.3970970  0.9907240 -0.1031780  2.10785147
```

```r
rownames(m) <- c("a","b","c", "d", "e", "f")
m
```

```
##         [,1]       [,2]       [,3]       [,4]        [,5]
## a  0.6979846  0.5060788  1.7877092 -0.3492734 -0.50819389
## b  2.1130557 -1.1417384 -1.2114209  0.9414483  0.70277541
## c -0.1591949  1.4796209 -1.1032868 -1.8258291  1.84210970
## d  0.6857158  1.0128991 -0.0889729  2.7984764  2.50704205
## e -0.6059189  0.1463889  2.7370227  1.5480494  0.06815174
## f  0.5777231  0.3970970  0.9907240 -0.1031780  2.10785147
```

```r
colnames(m) <- c("col1", "col2", "col3", "col4", "col5")
m
```

```
##         col1       col2       col3       col4        col5
## a  0.6979846  0.5060788  1.7877092 -0.3492734 -0.50819389
## b  2.1130557 -1.1417384 -1.2114209  0.9414483  0.70277541
```

```
## c -0.1591949  1.4796209 -1.1032868 -1.8258291  1.84210970
## d  0.6857158  1.0128991 -0.0889729  2.7984764  2.50704205
## e -0.6059189  0.1463889  2.7370227  1.5480494  0.06815174
## f  0.5777231  0.3970970  0.9907240 -0.1031780  2.10785147
```

6. Leyendo y descargando tablas

Descargando archivos

Si queremos analizar alguna tabla de datos que esté dentro de una base de datos pública, podemos utilizar código en R para descargar el archivo sin necesidad de buscar la tabla y moverla manualmente a nuestro directorio. Para ello, estableceremos una serie de pasos a seguir para que todo te resulte mucho más sencillo.

1. verifica que estás en el directorio correcto de tu proyecto Para esto haremos uso de las funciones `getwd()` y `setwd()`

```
getwd() ##nos dice donde estamos
setwd("./data") ##nos movemos a otro lugar
setwd("../") ##nos movemos una carpeta arriba.
```

2. Crear un directorio en donde colocaremos nuestras tablas de datos descargadas Esto es más que nada para mantener limpieza dentro de nuestro directorio del proyecto. Se hace uso de la función `dir.create()`

```
if(!file.exists("covid")){
     dir.create("covid")
}
```

3. Obtener datos en línea Para esto haremos uso de la función `download.file()`. Parametros importantes: URL, destfile, method. Cuando tenemos un archivo *ZIP*

```
url <- "http://datosabiertos.salud.gob.mx/gobmx/salud/datos_abiertos/historicos/2021/01/datos_abiertos_
download.file(url, destfile = "./covid/mexicocovid19.zip")
unzip("./covid/mexicocovid19.zip")
```

Si Se tratara de un archivo *csv* el código sería de la siguiente forma

```
url2 <- "https://datos.cdmx.gob.mx/dataset/9c45ead6-9016-469a-b6ba-41e3660590cb/resource/c97a1898-5343-
download.file(url2, destfile = "./covid/capacidadcovid19CDMX.csv")
```

De igual forma, si se trata de un archivo tipo excel, se guarda como *./directorio/nombredelarchivo.xlsx*

Leyendo tablas de datos en R

Existen varias formas de leer archivos en R. En este caso nos enfocaremos en 3. La primera implica el uso de las funciones `read.csv()` y `read.table()` que se encuentran en el R base, mientras que la otra implica el uso del paquete *xlsx* para leer tablas en documentos de excel.

read.table() y read.csv()

`read.table()` nos permite leer archivos en formatos planos como el *csv* o *txt*. Esta función para leer tablas grandes (big data) causa problemas.

parametros importantes: *na.strings*, *skip*, *stringsAsFactors*, *quote*

```
data1 <- read.table("./covid/capacidadcovid19CDMX.csv", sep = ",", header = TRUE)
head(data1)
```

```
##        Fecha
## 1 2020-04-14
## 2 2020-04-14
## 3 2020-04-14
## 4 2020-04-14
## 5 2020-04-14
## 6 2020-04-14
##                                                                Nombre_hospital
## 1                                          INSTITUTO NACIONAL DE NUTRICIÃ"N
## 2                              INSTITUTO NACIONAL DE ENFERMEDADES RESPIRATORIAS
## 3                                           HOSPITAL GENERAL DE MÃ%XICO
## 4                                                HOSPITAL JUÃ\201REZ
## 5 HOSPITAL DE ESPECIALIDADES DE LA CIUDAD DE MÃ%XICO DR BELISARIO DOMÃ\215NGUEZ
## 6                                        HOSPITAL GENERAL DR ENRIQUE CABRERA
##   Institucion Estatus_capacidad_hospitalaria Estatus_capacidad_UCI
## 1         SSA                          MEDIA                BUENA
## 2         SSA                          MEDIA                BUENA
## 3         SSA                          MEDIA                BUENA
## 4         SSA                          BUENA                BUENA
## 5      SEDESA                          MEDIA                MEDIA
## 6      SEDESA                          MEDIA                MEDIA
##               Coordenadas
## 1   19.289599, -99.155441
## 2  19.291375,Â -99.158653
## 3 19.4129258, -99.1520225
## 4   19.481829, -99.135576
## 5   19.307226, -99.065356
## 6   19.361675, -99.224432
```

`read.csv()` nos permite leer únicamente archivos en formato csv.

```
data2 <- read.csv("./covid/capacidadcovid19CDMX.csv", stringsAsFactors = TRUE)
```

Para leer archivos de excel, hacemos uso del paquete *xlsx* y utilizamos la función `read.xlsx()`. Parametros importantes: *sheetIndex*,*header*,*colindex*,*rowindex*.

```
library(xlsx)
read.xlsx("nombre del documento.xlsx", sheetIndex = 2 , header = TRUE, rowIndex = 1:50, colIndex = 1:5)
```

Convirtiendo tablas en archivos

Una vez que tengamos una tabla que hayamos creado nosotros podemos guardarla en diferentes formatos utilizando los comandos `write.table()`,`write.xlsx()`.

```
##csv
write.table(x, file = "nombre del archivo.csv", sep = ",")
##txt
write.table(x, file = "nombre del archivo.txt", row.names = FALSE)
```

```r
##xlsx
library(xlsx)
xlsx::write.xlsx()
```

7. Extracción de información

Operadores básicos para extraer trozos de información de objetos en R

[] El corchete lo que hace es regresarnos un objeto de la misma clase que el objeto original, es decir, si extraes información de un vector, tendrás de regreso otro vector, si es una lista, obtendrás de regreso otra lista. Puede ser utilizado para seleccionar más de un elemento dentro de un objeto.

[[]] Doble corchete se utiliza para extraer generalmente elementos de una lista o tabla de datos. Extrae un elemento y la clase no necesariamente tiene que ser de la misma del objeto original.

$ Extrae elementos de una lista o dataframe utilizando el nombre del elemento a extraer, puede o no ser de la misma clase que el objeto original

extrayendo de un vector

```r
x <- rnorm(100, 2,4)
x[3]
```

```
## [1] 1.058754
```

```r
x[1:10] ## una serie de elementos
```

```
##  [1]  4.995180  6.514876  1.058754  8.413419 -7.080194 -1.562183  4.992550
##  [8]  4.851064 -1.997865  6.062118
```

```r
x[c(3,5,8,9,15)] ## elementos específicos
```

```
## [1]  1.058754 -7.080194  4.851064 -1.997865  5.849712
```

```r
y <- c("a","r","t","g","e","d","j","ñ")
y[5]
```

```
## [1] "e"
```

```r
y[y>"a"]
```

```
## [1] "r" "t" "g" "e" "d" "j" "ñ"
```

```r
z <- y>"a" ## ojo, no son lo mismo
y[z]
```

```
## [1] "r" "t" "g" "e" "d" "j" "ñ"
```

Extrayendo de una lista Podemos utilizar los 3 operadores anteriormente mencionados para extraer elementos de una lista. Podemos extraer por nombre o por número de elemento de la lista.

```r
lista <- list(data=matrix(1:30, 6, 5), matriz=head(as.matrix(airquality), 20), vector1=rnorm(10))
head(lista,)
```

```
## $data
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    7   13   19   25
## [2,]    2    8   14   20   26
## [3,]    3    9   15   21   27
## [4,]    4   10   16   22   28
## [5,]    5   11   17   23   29
## [6,]    6   12   18   24   30
##
## $matriz
##       Ozone Solar.R Wind Temp Month Day
##  [1,]    41     190  7.4   67     5   1
##  [2,]    36     118  8.0   72     5   2
##  [3,]    12     149 12.6   74     5   3
##  [4,]    18     313 11.5   62     5   4
##  [5,]    NA      NA 14.3   56     5   5
##  [6,]    28      NA 14.9   66     5   6
##  [7,]    23     299  8.6   65     5   7
##  [8,]    19      99 13.8   59     5   8
##  [9,]     8      19 20.1   61     5   9
## [10,]    NA     194  8.6   69     5  10
## [11,]     7      NA  6.9   74     5  11
## [12,]    16     256  9.7   69     5  12
## [13,]    11     290  9.2   66     5  13
## [14,]    14     274 10.9   68     5  14
## [15,]    18      65 13.2   58     5  15
## [16,]    14     334 11.5   64     5  16
## [17,]    34     307 12.0   66     5  17
## [18,]     6      78 18.4   57     5  18
## [19,]    30     322 11.5   68     5  19
## [20,]    11      44  9.7   62     5  20
##
## $vector1
##  [1]  0.8466821 -0.3607518  1.0498500 -0.6973733 -0.4751658  1.4466801
##  [7] -0.6565717  0.2909487  0.4903145 -1.3790765
```

```r
lista["data"]
```

```
## $data
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    7   13   19   25
## [2,]    2    8   14   20   26
## [3,]    3    9   15   21   27
## [4,]    4   10   16   22   28
## [5,]    5   11   17   23   29
## [6,]    6   12   18   24   30
```

```r
lista[["matriz"]]
```

```
##       Ozone Solar.R Wind Temp Month Day
##  [1,]    41     190  7.4   67     5   1
##  [2,]    36     118  8.0   72     5   2
##  [3,]    12     149 12.6   74     5   3
##  [4,]    18     313 11.5   62     5   4
##  [5,]    NA      NA 14.3   56     5   5
##  [6,]    28      NA 14.9   66     5   6
##  [7,]    23     299  8.6   65     5   7
##  [8,]    19      99 13.8   59     5   8
##  [9,]     8      19 20.1   61     5   9
## [10,]    NA     194  8.6   69     5  10
## [11,]     7      NA  6.9   74     5  11
## [12,]    16     256  9.7   69     5  12
## [13,]    11     290  9.2   66     5  13
## [14,]    14     274 10.9   68     5  14
## [15,]    18      65 13.2   58     5  15
## [16,]    14     334 11.5   64     5  16
## [17,]    34     307 12.0   66     5  17
## [18,]     6      78 18.4   57     5  18
## [19,]    30     322 11.5   68     5  19
## [20,]    11      44  9.7   62     5  20
```

```
lista$vector1
```

```
##  [1]  0.8466821 -0.3607518  1.0498500 -0.6973733 -0.4751658  1.4466801
##  [7] -0.6565717  0.2909487  0.4903145 -1.3790765
```

```
lista[c(1,3)]
```

```
## $data
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    7   13   19   25
## [2,]    2    8   14   20   26
## [3,]    3    9   15   21   27
## [4,]    4   10   16   22   28
## [5,]    5   11   17   23   29
## [6,]    6   12   18   24   30
##
## $vector1
##  [1]  0.8466821 -0.3607518  1.0498500 -0.6973733 -0.4751658  1.4466801
##  [7] -0.6565717  0.2909487  0.4903145 -1.3790765
```

```
##sinónimos
lista[[c(3,5)]]
```

```
## [1] -0.4751658
```

```
lista[[3]][[5]]
```

```
## [1] -0.4751658
```

```
lista[[3]][5]
```

```
## [1] -0.4751658
```

```
lista$vector1[5]
```

```
## [1] -0.4751658
```

```
##concordancia parcial
lista$d
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    7   13   19   25
## [2,]    2    8   14   20   26
## [3,]    3    9   15   21   27
## [4,]    4   10   16   22   28
## [5,]    5   11   17   23   29
## [6,]    6   12   18   24   30
```

```
lista[["d", exact=FALSE]]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    7   13   19   25
## [2,]    2    8   14   20   26
## [3,]    3    9   15   21   27
## [4,]    4   10   16   22   28
## [5,]    5   11   17   23   29
## [6,]    6   12   18   24   30
```

extrayendo de una matriz o tabla de datos Podemos extraer información de las tablas de datos utilizando los siguientes subíndices [i,j] donde i = filas y j = columnas

```
data1[1,3]
```

```
## [1] "SSA"
```

```
data1[1,]
```

```
##        Fecha                    Nombre_hospital Institucion
## 1 2020-04-14 INSTITUTO NACIONAL DE NUTRICIÃ"N          SSA
##   Estatus_capacidad_hospitalaria Estatus_capacidad_UCI           Coordenadas
## 1                          MEDIA                 BUENA 19.289599, -99.155441
```

```
data[c(3,5,7), 1:3]
```

```
##                   mpg cyl disp
## Datsun 710       22.8   4  108
## Hornet Sportabout 18.7   8  360
## Duster 360       14.3   8  360
```

```r
## Columnas
head(data1[,3])
```

```
## [1] "SSA"    "SSA"    "SSA"    "SSA"    "SEDESA" "SEDESA"
```

```r
head(data1$Estatus_capacidad_UCI)
```

```
## [1] "BUENA" "BUENA" "BUENA" "BUENA" "MEDIA" "MEDIA"
```

```r
head(data1[data1$Estatus_capacidad_UCI == "BUENA",])
```

```
##         Fecha                                  Nombre_hospital Institucion
## 1  2020-04-14                INSTITUTO NACIONAL DE NUTRICIÃ"N          SSA
## 2  2020-04-14 INSTITUTO NACIONAL DE ENFERMEDADES RESPIRATORIAS         SSA
## 3  2020-04-14                      HOSPITAL GENERAL DE MÃ%XICO         SSA
## 4  2020-04-14                                HOSPITAL JUÃ\201REZ          SSA
## 36 2020-04-14                             HG IGNACIO ZARAGOZA      ISSSTE
## 37 2020-04-14                           HR PRIMERO DE OCTUBRE      ISSSTE
##    Estatus_capacidad_hospitalaria Estatus_capacidad_UCI             Coordenadas
## 1                           MEDIA                 BUENA   19.289599, -99.155441
## 2                           MEDIA                 BUENA  19.291375,Â -99.158653
## 3                           MEDIA                 BUENA 19.4129258, -99.1520225
## 4                           BUENA                 BUENA   19.481829, -99.135576
## 36                                                BUENA   19.389472, -99.045019
## 37                          BUENA                 BUENA   19.486751, -99.133249
```

Extrayendo y eliminando valores perdidos

Por lo general queremos eliminar aquellos valores NA que en realidad no aportan información a nuestra tabla. Por lo que existen muchas formas de identificar y eliminar valores perdidos en una tabla de datos, matriz o un vector.

1. Primero identificamos si existen valores perdidos, utilizando las funciones `is.na()`, `complete.cases()`
2. Eliminarlos directamente utilizando la función `na.omit()`

Paso 1

```r
## Identificando los valores NA.
head(complete.cases(airquality))
```

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

```r
head(is.na(airquality))
```

```
##      Ozone Solar.R  Wind  Temp Month   Day
## [1,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [2,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [3,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [4,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [5,]  TRUE    TRUE FALSE FALSE FALSE FALSE
## [6,] FALSE    TRUE FALSE FALSE FALSE FALSE
```

```r
sum(complete.cases(airquality))
```

```
## [1] 111
```

```r
nrow(airquality)
```

```
## [1] 153
```

```r
sum(is.na(airquality))
```

```
## [1] 44
```

```r
sum(!is.na(airquality))
```

```
## [1] 874
```

```r
## eliminando los valores NA de una tabla o matriz
head(airquality[complete.cases(airquality),],10)
```

```
##    Ozone Solar.R Wind Temp Month Day
## 1     41     190  7.4   67     5   1
## 2     36     118  8.0   72     5   2
## 3     12     149 12.6   74     5   3
## 4     18     313 11.5   62     5   4
## 7     23     299  8.6   65     5   7
## 8     19      99 13.8   59     5   8
## 9      8      19 20.1   61     5   9
## 12    16     256  9.7   69     5  12
## 13    11     290  9.2   66     5  13
## 14    14     274 10.9   68     5  14
```

```r
### !is.na sólo funciona con vectores
head(airquality$Solar.R[is.na(!airquality$Solar.R)])
```

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

```r
## O bien con la función na.omit
head(na.omit(airquality))
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 7    23     299  8.6   65     5   7
## 8    19      99 13.8   59     5   8
```

8. Estructuras básicas de Control

Cuando queremos establecer un programa dentro de R (escribir un código para que haga algo), existen ciertas estructuras que nos ayudan a controlar este flujo. Estas expresiones no se escriben dentro de un comando o función, al contrario, las funciones se escriben dentro de estas estructuras o cuando nosotros creamos una función.

if else

Nos ayuda a establecer y evaluar condiciones de tipo lógico, si la condición es verdadera, entonces el programa hace algo. *if* evalua una condición inicial, si esta condición es falsa, entonces *else* hace otra cosa. Es decir, si(*if*) tengo una condición A que es verdadera, ocurre algo, de otro modo (*else*) si A es falsa, ocurre otra cosa.

```
x <- 11

if (x>=10){
        y <- 1+1
}else{
        y <- 1+2
                }
y
```

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

```
if (x<10){
     y <- 1+1
}
```

Bucle for

Es el operador para bucles más común. La idea es tener un indice de búcle **i** (podemos tener más de un índice de blucle j,l,k,etc.). que cubre una secuencia de números enteros (integrer), números específicos, etc.

```
x <- vector("numeric", 20)
for (i in 1:20) {
     x[i] <- NA
}
x
```

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

Se pueden anidar los bucles for. Por ejemplo entre filas y columnasde una tabla.

```
x <- matrix(1:15, 5, 3)
x
```

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

```
for (i in seq_len(nrow(x))) {
     for (j in seq_len(ncol(x))) {
          print(x[i,j])
     }
}
```

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

```
x
```

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

bucle while

Toma una expresión lógica y ejecuta un loop (búcle) basado en el valor de esa expresión lógica. Si la condición es verdadera se ejecuta el loop, una ve ejecutada, la condición es evaluada de nuevo, si es cierta, se vuelve a repetir el loop. podemos tener más de una condición dentro del bucle while

```
x <- 5
while (x>3 & x<10) {
     x <- x+.5
}
x
```

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

En esta condición el loop se detiene cuando alguna de las dos condiciones establecidas dejan de ser verdaderas, es decir, cuando X llega a 3 o llega a 10 el loop se detiene.

Hay que tener cuidado porque podemos crear un loop infinito.

next break y return

Interrumpen el flujo de un programa. Por ejemplo si tenemos un código y dentro de ese código hay un loop y dentro de ese loop tenemos una condición que dé como resultado break, Next o Return, lo que harán será romper el loop y pasar a la siguiente parte del código.