Introducción al software R

Notas de clases

Paccioretti Pablo[1] Bruno Cecilia[2]
González Montoro Aldana[3] Nores María Laura[4]

[1]pablopaccioretti@agro.unc.edu.ar
[2]cebruno@agro.unc.edu.ar
[3]aldana.gonzalez.montoro@unc.edu.ar
[4]lalinores@yahoo.com.ar

Índice general

Capítulo 1

Instalación de programas

R puede ser instalado en múltiples plataformas tales como Windows, MacOS y en sistemas basados en Linux. Además hay múltiples entornos de desarrollo integrado (*Integrated Development Environment* IDE) los cuales facilitan la programación. Ejemplos de este tipo de software es RStudio (RStudio Team, 2016) y el intérprete de R que contiene InfoStat (Di Rienzo et al., 2018). Las interfaces gráficas de ambos software son similares.

Links para las descargas:

- R (windows)
- RStudio
- InfoStat

1.1 Interfaz de RStudio

La interfaz de RStudio se divide en cuatro paneles.

El panel superior izquierdo permite al usuario editar scripts (líneas de código), para esto puede cargar scripts previamente escritos o escribir nuevos. En el panel de abajo a la izquierda (*consola*) se muestran las sentencias de código ejecutadas y los resultados. En este panel el software se "comunica" con el usuario. En los paneles derechos se muestran los objetos cargados en el ambiente de trabajo, mientras que en el panel inferior derecho se muestran principalmente los archivos del directorio de trabajo, gráficos generados durante la sesión y ayudas de funciones específicas.

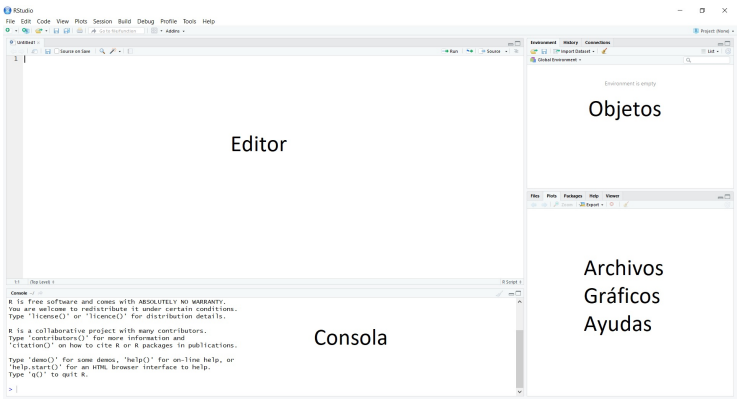


Figura 1.1: Diseño de los paneles de RStudio

1.2 Interfaz del intérprete de InfoStat

La interfaz del intérprete de R en InfoStat se divide en cuatro paneles.

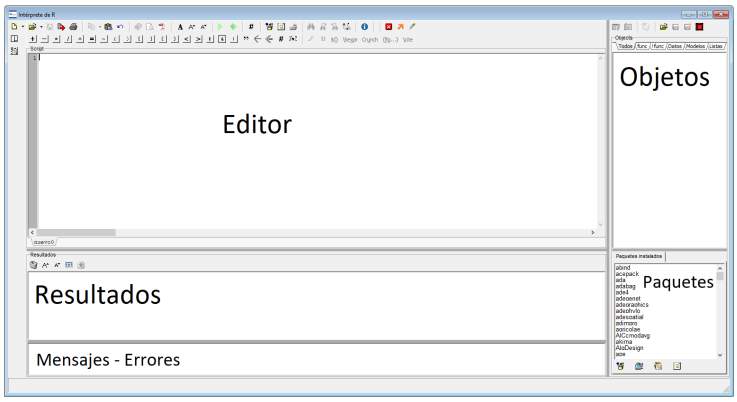


Figura 1.2: Diseño de los paneles del intérprete de R de InfoStat RStudio

El panel superior izquierdo permite al usuario visualizar o editar scripts. En el panel inferior izquierdo se muestran los resultados de funciones. En los paneles derechos se muestran los objetos cargados en el ambiente de trabajo, mientras que en el panel inferior derecho se muestran los paquetes instalados y en rojo los paquetes cargados en memoria.

Capítulo 2

El lenguaje R

R (R Core Team, 2022) es un lenguaje de programación orientado a objetos. Fue creado por Ross Ihaka y Robert Gentleman en 1993 como un dialecto del software S, fue publicado en 1996 (Ihaka and Gentleman, 1996). Es un software libre y de código abierto, lo que significa que se puede usar, compartir y modificar el software libremente. Junto con el instalador de R, se distribuyen ciertos paquetes (*packages*) los cuales incluyen funciones para implementar algunos métodos estadísticos clásicos y modernos. Pot esta razón, muchas personas utilizan R para realizar análisis estadísticos. Muchos algoritmos y metodologías estadísticas están disponibles para ser implementadas en R, pero muchas de ellas se encuentran en paquetes específicos que no se encuentran en la instalación básica de R, por lo que para poder utilizar estas funciones, se debe descargar el paquete.

2.1 Generalidades del ambiente R

R distingue mayúsculas y minúsculas, esto significa que `Producto` y `producto` son dos palabras diferentes. Los comandos elementales consisten en expresiones o asignaciones. Si se ejecuta una expresión el resultado se imprimirá en la consola pero no se guardará dicho valor. Cuando se asigna un valor de una expresión (mediante el comando `<-`), el resultado no se imprimirá en pantalla pero el valor será asignado a un objeto que luego podrá reutilizarse. Diferentes sentencias de código (comandos) pueden ser separados por `;` o por una nueva línea dentro del script, la segunda opción es la más utilizada por los usuarios. Un conjunto de comandos pueden estar encerrados entre llaves (`{` y `}`). Los `#` indican comentarios en el código, todo lo que está a la derecha de este símbolo no será ejecutado por R. Si se desean hacer comentarios de más de una línea, cada una de ellas debe comenzar con `#`.

Si deseamos asignar en el objeto llamado `x` el valor del resultado de aplicar la raíz cuadrada al número 10, debemos utilizar la función `<-`:

```
x <- sqrt(10) #No se muestra el resultado
```

Para ver el valor de cualquier objeto, se puede especificar el nombre y ejecutar la línea, por ejemplo si deseamos ver el valor que está almacenado en `x` debemos escribir y ejecutar:

```
x #Se muestra el resultado
```

```
## [1] 3.162278
```

```
sqrt(10) #Se imprime en la consola el resultado
```

```
## [1] 3.162278
```

Las funciones son segmentos de código escrito para llevar a cabo una tarea específica, en el ejemplo anterior se utilizó la función `sqrt` para calcular la raíz cuadrada de 10. Las funciones pueden necesitar argumentos y devuelven uno o más valores en el resultado, algunas funciones pueden no devolver ningún resultado visible. Los argumentos de la función son los *inputs* para ejecutar la tarea. Argumentos deben ir dentro de paréntesis luego del nombre de la función, cada argumento se separa con `,` (`(arg1, arg2 )`). Nombres de los argumentos pueden especificarse explícitamente o no, en el ejemplo anterior los argumentos no se especificaron con nombre explícito. Si no se detalla el nombre del argumento, R entenderá que están en el mismo orden que se especificaron cuando se creó la función. En el caso de `sqrt` el primer y único argumento de la función es un objeto numérico.

Notar que los nombre de la mayoría de las funciones de R derivan del inglés y que R utiliza `.` como separador decimal.

2.2 Funciones y comandos básicos

En R se puede ejecutar cualquier operación matemática. Comencemos viendo algunas operaciones básicas:

Suma:

```
5 + 2
```

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

Raíz cuadrada:

```
sqrt(15)
```

```
## [1] 3.872983
```

2.2.1 Tablas resumen de operadores y funciones

Cuadro 2.1: Algunas funciones matemáticas en R

Sintaxis	Operación
‘x + y‘	suma de x e y
‘x - y‘	diferencia de x e y
‘x * y‘	multiplicación de x e y
‘x / y‘	división de x por y
‘x %/% y‘	parte entera de la división de x por y
‘x %% y‘	resto de la división de x por y
‘x ^ y‘	x elevado a y-ésima potencia
‘x < y‘	x menor que y
‘x <= y‘	x menor o igual que y
‘x > y‘	x mayor que y
‘x >= y‘	x mayor o igual que y
‘x == y‘	x igual a y
‘x != y‘	x no es igual a y
‘sqrt(x)‘	raíz cuadrada de x
‘exp(x)‘	exponencial de x
‘log(x)‘	logaritmo natural de x
‘log(x, k)‘	logaritmo base k de x
‘sum(x)‘	suma de los elementos de x
‘prod(x)‘	producto de los elementos de x
‘round(x, k)‘	x redondeado a k dígitos

2.2.2 Ayuda

R incluye documentación de ayuda muy detallada. Para acceder a la ayuda de cada función, objeto o datos de prueba se debe ejecutar el comando `help()` o `?`.

Por ejemplo `help(sqrt)`, o `?sqrt`. Otra forma de pedir la ayuda es presionando la tecla F1 luego de seleccionar el nombre de la función en RStudio. La sentencia `??` busca un patrón dentro de la documentación del sistema de ayuda, es útil si no se conoce cuál función ejecuta cierto análisis. Otra herramienta muy útil para buscar ayuda es Google o Stack Overflow.

```
help(sqrt)
??square
```

2.2.3 Asignaciones

Como ya se especificó en la sección 2.1, un comando de asignación es `<-`, donde a la izquierda se especifica el nombre del objeto y a la derecha el valor, ya sean resultados de un cálculo matemático o de un análisis estadístico más complejo. Por ejemplo, si se desea asignar el valor de `5` al objeto `radio` se debe ejecutar `radio <- 5`. Otras formas de hacer asignaciones es mediante la utilización de `=` o `->`, este último no es muy utilizado entre los usuarios de R.

Asignaremos al objeto `x` una secuencia numérica del 1 al 5 y luego imprimiremos el contenido de `x` en la consola:

```
x <- c(1, 2, 3, 4, 5)  #No se muestra el resultado
x                      #Se auto imprime el resultado
## [1] 1 2 3 4 5
print(x)               #Imprime el resultado de manera explícita
## [1] 1 2 3 4 5
                       #mediante el comando print
```

Otras formas de asignar valores es utilizando `->` o `=`

```
c(1, 2, 3) -> x
x
```

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

```
x = c(1, 2, 3, 4)
x
```

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

Al utilizar el comando de asignación con el mismo nombre de objeto (`x`), cada vez que se utilizó ese comando, el valor que contenía previamente se reasignó con el valor nuevo.

2.2.4 R como herramienta estadística

En el paquete `stats` (uno de los paquetes instalados por defecto al momento de instala R) permite entre otras cosas, obtener la densidad, función de distribución (probabilidades), cuantiles y generar números aleatorios de las distribuciones estadísticas más comunes. Por ejemplo, si se desea generar 40 números aleatorios de una distribución normal estándar se deberá ejecutar la sentencia `rnorm(40)`.

Si se desea calcular medidas descriptivas básicas de un vector se puede ejecutar `mean` para calcular la media, `sd` para calcular el desvío estándar y `var` para la varianza. Otra función útil para obtener valores de posición es la función `summary`.

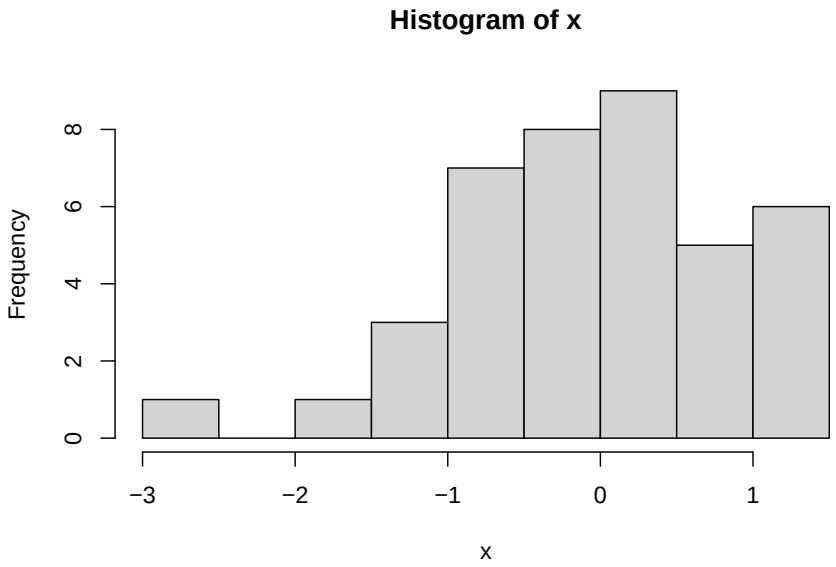
```
x <- rnorm(40)
summary(x)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.90063 -0.69488  0.02006 -0.07096  0.67489  1.21151
```

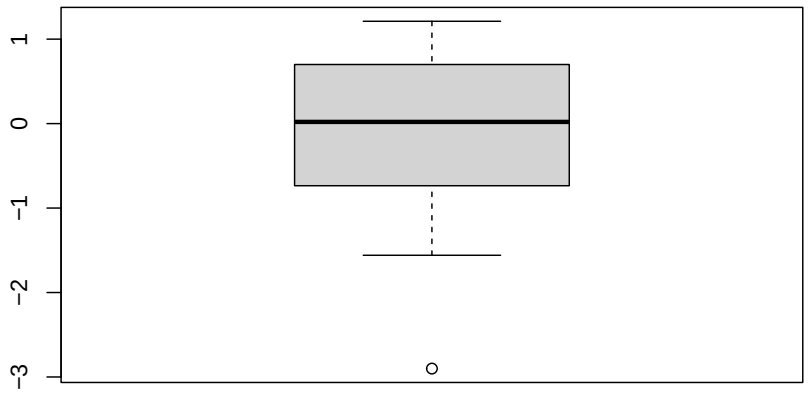
2.2.5 R como herramienta gráfica

Con R se puede realizar gráficos y modificar numerosos parámetros de éste para que pueda ser publicado. A modo de ejemplo se realizará un histograma y un boxplot de la variable `x` generada anteriormente.

```
hist(x)
```



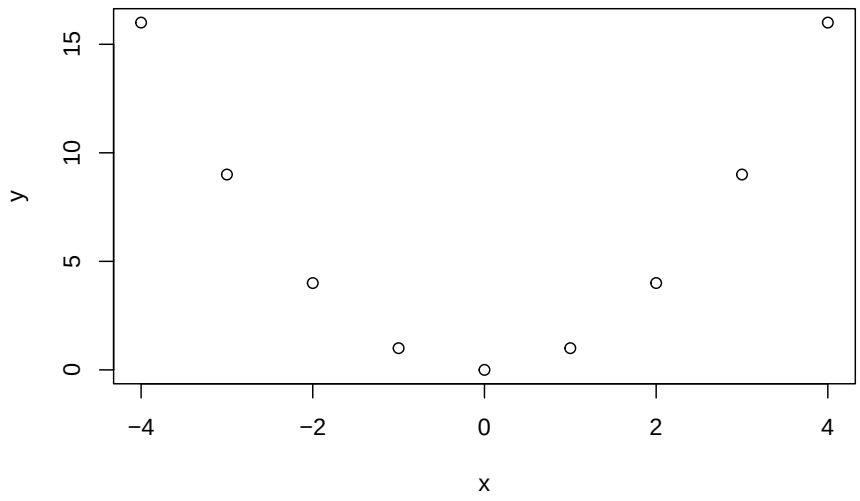

```
boxplot(x)
```



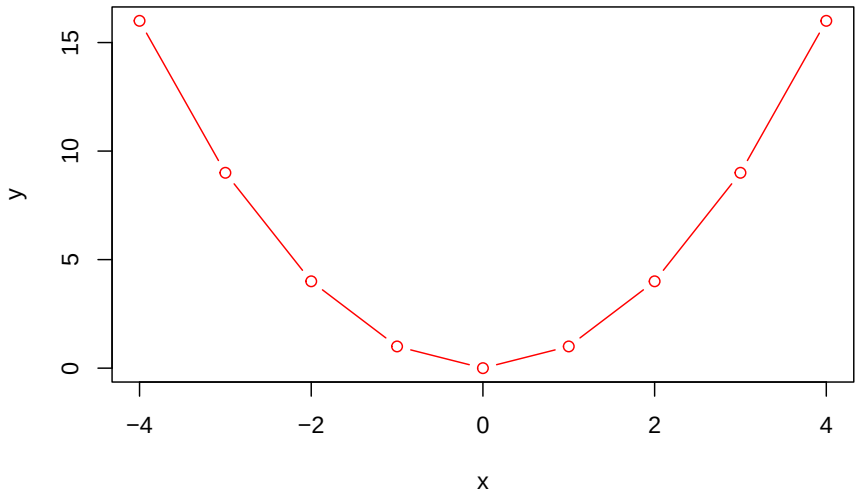
Podría decirse que la función más importante para generar gráficos dentro de los paquetes instalados por defecto es `plot`. Esta función permite permite realizar

diagramas de dispersión y editar algunos elementos del gráfico.

```
x <-
  c(-4, -3, -2, -1, 0, 1, 2, 3, 4)  # Observar que se remplazó el objeto "x" que
                                    # se generó previamente
y <- x ^ 2
plot(x, y)
```




```
plot(x, y, type = "b", col = "red")
```



2.2.5.1 Ejercitación

1. Funciones y comandos básicos

   Calcule la raíz cuadrada de 10

   Calcule el perimetro del círculo de radio 5 ($P = 2\pi \times r$)

   Calcule 270 dividido la suma entre 12 y 78

   Calcule el cuadrado de 8

   Calcule el logaritmo de 10

2. Asignaciones y aritmética vectorial

   Calcule el perímetro del círculo de radio 5 y guárdelo en el objeto `per`.

   Crear el vector de coordenadas 6,7,8,9,10 y llamarlo `z`Suma de dos vectores

   Calcular la suma de `z` y `x`

   Calcular el doble de `x`

   ¿Qué se obtiene haciendo el producto entre los vectores `z` y `x`?

3. R como herramienta estadística

   Generar un vector `y` con 20 realizaciones de una normal con media 5 y desvío estándar 2. Calcular la media y la varianza de `y`. Realizar un histograma.

Capítulo 3

Objetos en R

Los resultados de un cierto procedimiento pueden ser almacenados en diferentes clases de objetos. R tiene cinco clases básicas de objetos, números (`numeric`), números complejos (`complex`), cadenas de caracteres (`character`), factores (`factor`) y valores lógicos (`logical`). Éstos pueden juntarse para formar vectores (`vector`), matrices (`matrix`), hojas de datos (`data.frame`) o listas (`list`). Otras clases de objetos pueden ser funciones, modelos, objetos espaciales, entre otros. En esta sección trabajaremos con algunos de ellos. Para conocer la clase de un objeto se utiliza la función `class`.

3.1 Vectores

Es el objeto más simple de R. Es importante tener en cuenta que los vectores sólo contienen elementos de la **misma** clase básica. La función `c()` puede utilizarse para crear vectores concatenando sus argumentos.

```
x <- c(2, 4, 6)               #numérico (enteros)
a <- c(-1, 5, 9, 10.5)        #numérico (continuo)
d <- c(4+2i, 2+5i)            #números complejos
y <- c("a", "b", "c")         #caracteres
z <- c(TRUE, TRUE, FALSE, T)  #lógico
```

Si quisiéramos concatenar `x` y `a` podríamos utilizar como argumentos de la función `c()`, a los objetos que quisiéramos agrupar.

```
x <- c(2, 4, 6)
a <- c(-1, 5, 9, 10.5)
x_a <- c(x, a)
```

```
x_a
## [1]  2.0  4.0  6.0 -1.0  5.0  9.0 10.5
length(x_a)
## [1] 7
```

Si quiséramos comprobar que el objeto `x_a` tiene una longitud de 7 elementos (igual a la suma de elementos de los objetos `x` y `a`), podemos utilizar la función `length()`.

```
length(x) + length(a)
## [1] 7
length(x_a)
## [1] 7
```

Note que en el ejemplo anterior `T` y `F` es la forma corta de especificar `TRUE` y `FALSE`. Es recomendable utilizar la forma explícita de `TRUE` y `FALSE` antes que la forma corta, dado que `T` y `F` son símbolos que pueden redefinirse, por lo que no se debería asumir que siempre se van a evaluar como operadores lógicos.

3.1.1 Secuencias

```
x <- c(1, 2, 3, 4, 5)
x <- 1:10
y <- -5:3
```

Para generar secuencias de números enteros consecutivos se puede utilizar `:`, pero si se desea generar otros tipos de secuencias, por ejemplo la secuencia 4,6,8,...,20, se debe utilizar la función `seq`. Los argumentos de esta función permiten generar secuencias con saltos o longitud definida por el usuario.

```
seq(from = 4, to = 20, by = 2)
```

```
## [1]  4  6  8 10 12 14 16 18 20
```

Los argumentos de la función `seq`, permiten generar secuencia, desde (`from` y, hasta (`to`) los valores especificados. Se pueden especificar el incremento de cada valor (`by`), o puede definirse el largo de la secuencia deseada (`length.out`).

3.1.2 Vectores con valores repetidos

Cuando se desea generar un vector con valores repetidos se puede utilizar la función `rep`. Esta función replica los valores que se espicifican en el primer argumento, tantas veces o hasta alcanzar la longitud total que se especifique.

```
rep(1, 5)
## [1] 1 1 1 1 1
x <- 1:3
rep(x, 2)
## [1] 1 2 3 1 2 3
rep(x, c(2,4,1)) #En este caso repetirá el 1 dos vecesm el 2 cuatro veces y el 3 una vez.
## [1] 1 1 2 2 2 2 3
rep(x, length = 8)
## [1] 1 2 3 1 2 3 1 2
```

3.1.3 Vectores de factores

Los vectores que se generan pueden convertirse en factores, para ello se utiliza la función `as.factor`.

```
x_f <- as.factor(x)
```

Tambien pueden generarse vectores que contiene factores utilizando `gl`. A esta función se le debe especificar el número de niveles del factor y el número de repeticiones. Se le puede especificar el largo del vector y las etiquetas (`labels`) de los factores.

```
gl(3, 5)
```

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

```
gl(3, 5, length = 30)
```

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

```
gl(3, 5, labels = c("grupo A", "grupo B", "grupo C"))
```

```
##  [1] grupo A grupo A grupo A grupo A grupo A grupo B grupo B grupo B grupo B
## [10] grupo B grupo C grupo C grupo C grupo C grupo C
## Levels: grupo A grupo B grupo C
```

3.1.4 Seleccion de elementos de un vector

Los corchetes (`[ ]`) se utilizan para indicar posición de un objeto. Se utilizan del lado derecho del objeto. Dado que los vectores son elementos de una dimensión, si se desea seleccionar el primer elemento del objeto `x` se debe indicar `x[1]`.

```
x <- c(3,52,-8,2,1,7,11,-3,0,6,23,17)
x[1]
```

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

```
x[3]
```

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

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

```
## [1]  3 -8
```

Si se desea sustituir un elemento del vector se puede utilizar el signo de asignación. Por ejemplo, si se desea sustituir el tercer elemento de x por 88:

```
x[3] <-  88
x
```

```
##  [1]  3 52 88  2  1  7 11 -3  0  6 23 17
```

Si se quiere obtener un vector sin algunos elementos, se debe anteponer el signo - al valor del índice.

```
x[-3]
```

```
##  [1]  3 52  2  1  7 11 -3  0  6 23 17
```

3.2 Matrices

Las matrices son vectores con atributo de dimensión (2 dimensiones: filas y columnas). A diferencia de los `data.frame`s, todas las columnas de las matrices son de una misma clase. Para generar matrices se puede utilizar la función `matrix`.

```
x <- 1:20
matrix(x, nrow = 5, ncol = 4)
```

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

Las matrices pueden ser creadas uniendo filas o columnas mediante las funciones `cbind()` y `rbind()`.

```
x <- 1:3
y <- 10:12
cbind(x, y)
```

```
##      x  y
## [1,] 1 10
## [2,] 2 11
## [3,] 3 12
```

```
rbind(x, y)
```

```
##   [,1] [,2] [,3]
## x    1    2    3
## y   10   11   12
```

3.2.1 Operaciones con matrices:

- `A %*% B` : producto de matrices
- `t(A)` : traspuesta de la matriz A
- `solve(A)` : inversa de la matriz A
- `solve(A,b)` : solución del sistema de ecuaciones Ax=b.
- `svd(A)` : descomposición en valores singulares
- `qr(A)` : descomposición QR
- `eigen(A)` : valores y vectores propios
- `diag(b)` : matriz diagonal. (b es un vector)
- `diag(A)` : matriz diagonal. (A es una matriz)
- `A %o% B == outer(A,B)` : producto exterior de dos vectores o matrices

3.3 Listas

Una lista es la forma generalizada de un vector que puede contener elementos de diferentes clases (número, vector, matriz, lista, etc.). Para crear lista se puede utilizar la función `list()`. Dada su flexibilidad son contenedores generales de datos. Muchas funciones devuelven un conjunto de resultados de distinta longitud y distinto tipo en forma de lista.

```
n <-  c(2, 4, 6)
s <-  c("aa", "bb", "cc", "dd", "ee")
b <-  c(TRUE, FALSE, TRUE, FALSE, FALSE)
x <-  list(n, s, b)
x
```

```
## [[1]]
## [1] 2 4 6
##
## [[2]]
## [1] "aa" "bb" "cc" "dd" "ee"
##
## [[3]]
## [1]  TRUE FALSE  TRUE FALSE FALSE
```

3.4 Hojas de datos (Data frames)

Es el objeto más común en R para almacenar datos. Sus columnas pueden ser de diferentes clases por ejemplo variables continuas y categóricas. Este tipo de objetos puede generarse mediante la función `data.frame()`.

`data.frame` convierte los vectores de caracteres en factores automáticamente.

```
x1 <- 1:10
x2 <- 24:33
x3 <- gl(2, 5, labels = c("si","no"))
x4 <- letters[1:10]
data.frame(A = x1, B = x2, C = x3, D = x4)
```

```
##     A  B  C D
## 1   1 24 si a
## 2   2 25 si b
## 3   3 26 si c
## 4   4 27 si d
## 5   5 28 si e
## 6   6 29 no f
## 7   7 30 no g
## 8   8 31 no h
## 9   9 32 no i
## 10 10 33 no j
```

3.5 Algunas funciones básicas predefinidas

- summay()
- mean()
- var()
- sd()
- cor()
- sum()
- min()
- max()
- seq()
- which()
  - which.min()
  - which.min()
- length()
- table()
- is.na()
- is.null()

- `complete.cases()`
- `as.character()`
- `as.numeric()`
- `paste()`
- `gsub()`
- `unique()`
- `lm()`
- `dim()`
- `nrow()`
- `ncol()`
- `colnames()`
- `rownames()`
- `edit()`
- `cbind()`
- `rbind()`
- `order()`
- `install.packages()`
- `library()`

3.5.1 Ejercitación

1. Vectores

   Genere un vector `b` el cual contenga los valores de `x` y `a` ¿Cuantos elementos tiene el vector `b`?

2. Secuencias

   Genere la secuencia 8,7,6,5,4

   seq(4,20,2) ¿Este comando da eror? ¿Por qué?

   Genere usando comandos para secuencias el vector de componentes: 1, 2, 3, 4, 5, 6, 73, 72, 71, 70, 69, 68, 3, 6, 9, 12, 15, 18.

3. Repetir valores

   Genere un vector de componentes “azul”, “azul”,“azul”, “azul”, “amarillo”, “amarillo”, “verde”, “verde”,“verde”, llamado `col`. ¿Es un vector de factores?

4. Matrices

   Calcule la inversa y los autovalores y autovectores de `A = matrix(c(1,3,3,9,5,9,3,5,6), nrow = 3)`

Capítulo 4

Control de flujo

4.1 Construcción condicional `if`

Es de la forma `if (expr 1) expr 2 else expr 3` donde `expr 1` debe producir un valor lógico. Si `expr 1` es verdadero (`TRUE`), se ejecutara `expr 2`. Si `expr 1` es falso (`FALSE`), y se ha escrito la opcion else, que es opcional, se ejecutara `expr 3`.

```
if (3 > 2)
  print("yes")
```

```
## [1] "yes"
```

```
if (2 > 3)
  print("yes")

if (2 > 3) {
  print("yes")
} else{
  print("no")
}
```

```
## [1] "no"
```

Ejemplo con dos condiciones:

Supongamos que `x <- 75` es la nota numérica de examen de un estudiante, queremos asignar nota “A”, “B” o “C”

```
if (x < 60)
  nota = "C"
if (x >= 60 & x < 80)
  nota = "B"
if (x >=  80)
  nota = "A"
```

`ifelse` es la versión vectorizada de `if`, eto sifnigica que para cada elemento del vector lógico, ejecutará una u otra función dependiendo del valor.

Ejemplo:

```
nota.num <-
  c(39, 51, 60, 65, 72, 78, 79, 83, 85, 85, 87, 89, 91, 95, 96, 97, 100, 100)

prueba <- ifelse(nota.num >= 60, "aprobado", "desaprobado")
prueba
```

```
##  [1] "desaprobado" "desaprobado" "aprobado"    "aprobado"    "aprobado"
##  [6] "aprobado"    "aprobado"    "aprobado"    "aprobado"    "aprobado"
## [11] "aprobado"    "aprobado"    "aprobado"    "aprobado"    "aprobado"
## [16] "aprobado"    "aprobado"    "aprobado"
```

4.2 Construcción repetitiva `for`

Es de la forma `for (nombre in obj) {expr}` donde `nombre` es la variable de control de iteración, esto sifnifica que asumirá cada valor del objeto `obj` es un vector (puede ser una secuancia `m:n`), y `expr` es una expresión, a menudo agrupada, en cuyas sub-expresiones puede aparecer la variable de control, `nombre`. `expr` se evalua repetidamente conforme `nombre` recorre los valores del objeto `obj`.

```
for (i in 1:10) print(i)
```

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

```
x = numeric(10)
for (i in 1:10) x[i] = i^2

y = 0
for (i in 1:10) y = y + i
```

4.3 Construccion repetitiva `while`

Es de la forma `while (expr1) expr2`, indicando que se quiere repetir la acción `expr2` mientras que ocurra `expr1`.

```
i = 0
while (i < 15) {
  print(i)
  i = i + 1
}
```

```
## [1] 0
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
```

4.3.1 Ejercitación

1. Construcción condicional

   Escriba el comando neccesario para poner notas “A”, “B” o “C” dependiendo del valor de `final_score`: “C” si final_score <60, “B” si 60 =< final_score < 80, “A” si 80 =< final_score =< 100.

2. Construcción repetitiva

   Usar un ciclo `for` para contar la cantidad de números mayores a 10 en el vector `x <- c(2,5,3,9,8,11,6,8,12,3,57,56)`, ¿Se podría hacer sin la función `for`?

Capítulo 5

Generar nuevas funciones

R es un lenguaje que permite crear nuevas funciones. Una función se define con una asignación de la forma:

```
nombre <- function(arg1, arg2, ...) {
   expresion
 }
```

La `expresion` es una fórmula o grupo de fórmulas (o sentencias) que utilizan los argumentos para calcular uno o varios valores. El resultado de dicha expresión es el valor que proporciona R en su salida y éste puede ser un número, un vector, un gráfico, una lista. Una función devuelve el valor de la última expresión evaluada.

Ejemplos:

```
funcion_1 <- function(x) { y = x + 4}

(a <- funcion_1(5))
```

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

En el caso siguiente, si se desea guardar el resultado en un objeto sólo se almacenará el rango (último valor impreso en consola).

```
funcion_2 <- function(muestra) {     #El único argumento es un vector de datos
  media = mean(muestra, na.rm = T)
  varianza = var(muestra, na.rm = T)
  rango = max(muestra, na.rm = T) - min(muestra, na.rm = T)
  print(paste("Media:", round(media, 2)))
  print(paste("Varianza:", round(varianza, 2)))
```

```
  print(paste("Rango:", round(rango, 2)))
}

miValor <- funcion_2(rnorm(40,5,16))
```

```
## [1] "Media: 8.67"
## [1] "Varianza: 224"
## [1] "Rango: 59.92"
```

```
miValor
```

```
## [1] "Rango: 59.92"
```

Para que guarde los tres resultados hay que especificar que se haga una lista o vector.

```
funcion_3 <- function(muestra) {
  med = mean(muestra, na.rm = T)
  vari = var(muestra, na.rm = T)
  rang = max(muestra, na.rm = T) - min(muestra, na.rm = T)
  # list(media = med, varianza = vari ,rango = rang)
  c("Media" = med,
    "Var" = vari,
    "Rango" = rang)
}

ej <- funcion_3(1:20)
ej
```

```
## Media   Var Rango
##  10.5  35.0  19.0
```

Los diferentes argumentos de las funciones se separan con `,`. Éstos pueden tener un valor por defecto. Para indicarlo, en el momento de definir la nueva función, se determina con el signo `=` cuál es el valor que se usará si el usuario no lo especifica explícitamente.

```
funcion_4 <- function(a, b, c = 4, d = FALSE) {
  if (d == FALSE) {
    x1 <- a * b
  } else {
    x1 <- a * b + c
  }
  x1
}
```

Para utilizar la función, no es necesario explicitar el nombre de cada argumento. R seguirá el orden de la lista de argumentos que se utilizó en la definición de la función. Así, es recomendable que los argumentos sin valores por defecto, se coloquen como últimos argumentos.

```
funcion_4(a = 2, b = 5)
```

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

```
funcion_4(2, 5)
```

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

```
# Especifica valores de 2, 5 y 3 para a, b y c, sin tener que aclarar
# los nombres de dichos argumentos
funcion_4(2, 5, 3)
```

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

```
funcion_4(2, 5, 3, TRUE)
```

```
## [1] 13
```

En el caso de no asignar un valor a uno de los argumentos y optar por utilizar el valor por defecto, es necesario el uso de "d = " pues en caso contrario el tercer argumento corresponde a c.

```
funcion_4(2, 5, d = TRUE)
```

```
## [1] 14
```

5.0.1 Ejercitación

1. Funciones

   Genere una función que grafique una variable en función de otra y coloque nombre al eje x que por defecto sea: "mi eje x"

Bibliografía

Di Rienzo, J., Casanoves, F., Balzarini, M., Gonzalez, L., Tablada, M., and Robledo, C. (2018). *InfoStat*.

Ihaka, R. and Gentleman, R. (1996). R: a language for data analysis and graphics. *Journal of computational and graphical statistics*, 5(3):299–314.

R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

RStudio Team (2016). *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA.