Notas Curso Básico R

Tobías Chavarría

Actualizado el 16 Jan, 2021

Índice general

Capítulo 1

Introducción

Este documento contiene las bases del lenguaje de programación **R** para poder empezar a realizar análisis de datos, se va a iniciar con la instalación y las configuraciones básicas, y luego avanzaremos a los principios básicos del lenguaje y las herramientas necesarias que nos permitan realizar un análisis exploratorio de un conjunto de datos, construir funciones básicas y realizar visualizaciones.

Capítulo 2

Primeros pasos con R y RStudio

R es un entorno y lenguaje de programación con un enfoque al análisis estadístico

RStudio es un IDE por sus siglas en ingles Integrated Development Environment o Entorno De Desarrollo Integrado que facilita la interacción con el lenguaje de programación R y los procesos de carga de datos, instalación y administración de paquetes, exportación de gráficos y administración de archivos, entre otros.

El objetivo de este capítulo es conocer el entorno de trabajo que proporciona R y RStudio, además de aprender a instalar y cargar los paquetes que se necesiten para realizar análisis de datos.

2.1. Instalación

2.1.1. R

Para instalar R en Windows, la forma más simple es descargar la versión más reciente de R base desde el siguiente enlace de CRAN:

https://cran.r-project.org/bin/windows/base/

El archivo que necesitamos tiene la extensión **.exe** (por ejemplo 4.0.2-win.exe). Una vez descargado, lo ejecutamos como cualquier instalable.

Después de la instalación, estamos listos para usar R.

2.1.2. RStudio

Para instalar RStudio, es necesario descargar y ejecutar alguno de los instaladores disponibles en su sitio oficial. Están disponibles versiones para Windows, OSX y Linux.

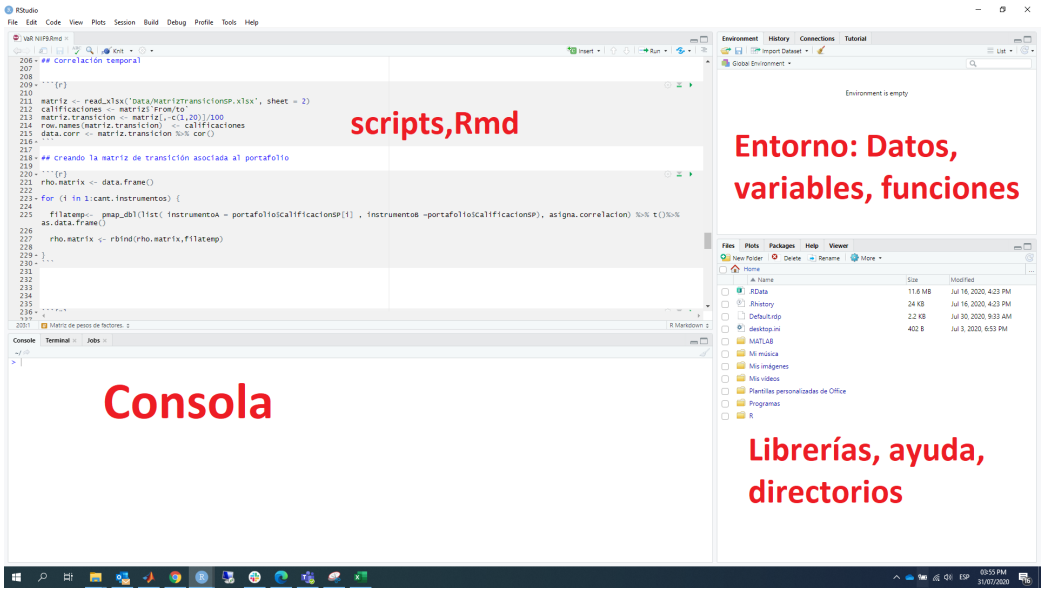
https://www.rstudio.com/products/rstudio/download/

Si ya hemos instalado R en nuestro equipo, RStudio lo detectará automáticamente y podremos utilizarlo desde este entorno. Si no instalamos RStudio antes que R, no hay problema, cada vez que iniciamos este programa, verificará la instalación de R.

2.2. Entorno de trabajo de RStudio.

En general se trabaja con la interfaz de RStudio antes que con la de R porque la primera es mucho "más amigable."

Al abrir RStudio veremos algo como esto:



Una vez estamos en RStudio, podemos escribir y ejecutar las órdenes de varias formas:

- Directamente en la consola
- A través de un script (.R)
- Con ficheros Rmarkdown (.Rmd)

2.2.1. Consola de R

La consola de RStudio nos permite interactuar con los comandos de R, es decir, ingresamos una instrucción en la consola y esta retornará el resultado de la ejecución de ese comando, aunque esta es una herramienta muy útil no es la mejor opción cuando nuestro código gana complejidad.

En la consola escribimos **expresiones**, el símbolo "<-" es el operador de asignación, aunque también se puede utilizar el símbolo "=."

Asignación de valores.

```
x <- 1  # Asignamos el valor 1 a la variable x

texto <- "Bienvenidos"  # Asignamos el valor "Bienvenidos" a la variable texto
```

En R el símbolo "#" indica que es un comentario, cualquier cosa que esté a su derecha (incluido el "#") será ignorado a la hora de ejecutar el código. Este es el único símbolo para hacer comentarios en R y además cabe mencionar que R no soporta comentarios en bloques o multilíneas.

Evaluación.

Cuando escribimos una expresión en la consola, podemos imprimir su valor sin una orden explícita.

```
x <- 13 # No imprime nada, solo asigna el valor
```

```
x # Se imprime el valor
```

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

```
print(x) # Orden explicita
```

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

2.2.2. Ayuda en R

Al comenzar a trabajar con R necesitaremos información sobre cada instrucción, función y paquete. Toda la documentación se encuentra integrada en RStudio, para accesar a esta información podemos usar la función **help()** o el signo de interrogación **?**, de la siguiente manera

```
help("funcion")

?funcion
??nombre_paquete
```

Al ejecutar estas instrucciones la información aparece en la pestaña de **help**.

```
help("read.table")

?read.table
```

2.2.3. Nombres en R

Al igual que la documentación de nuestro código, es importante el nombre que le demos a nuestros objetos (variables, funciones). En **R** los nombres de los objetos deben comenzar con una letra y solo pueden contener letras, números y los signos : " ", "." *Es bueno que los nombres sean descriptivos, es necesario adoptar una convención, la más común es la del guión bajo (snake_case) en la que los nombres se escriben en minúscula y separados por* .

```
yo_uso_guion_bajo  ## snake_case

OtraGenteUsaMayusculas

algunas.personas.usan.puntos ## Esto es peculiar de R, ya que en otros lenguajes el
## ya que tiene otras funciones

Y_algunasPocas.Personas_RENIEGANdelasconvenciones
```

Generalmente las variables son sustantivos y el nombre de las funciones verbos, se debe procurar que los nombres sean concisos y con significado.

```
## Correcto

dia_uno <- 10

## Incorrecto

primer_dia_del_mes <- 10
```

También se debe evitar utilizar nombres de funciones o variables comunes, esto causa confusión al leer el código.

```r
## Incorrecto

T <- FALSE
c <- 10

mean <- function(x) {
  sum(x)
}
```

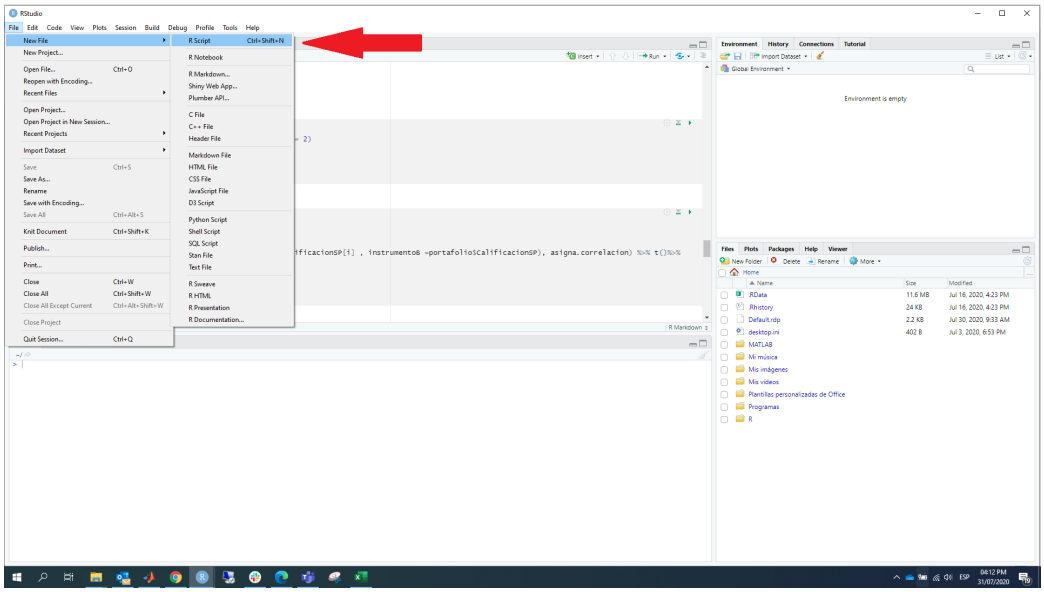
Existen muchas otras buenas prácticas a la hora de escribir código en **R**, el siguiente link contiene una guía del estilo *tidyverse*.

https://style.tidyverse.org/index.html

2.2.4. Scripts de R

Trabajar en la consola es muy limitado ya que las instrucciones se tienen que escribir una por una. Lo habitual es trabajar con scripts o ficheros de instrucciones. Estos ficheros tienen extensión **.R**.

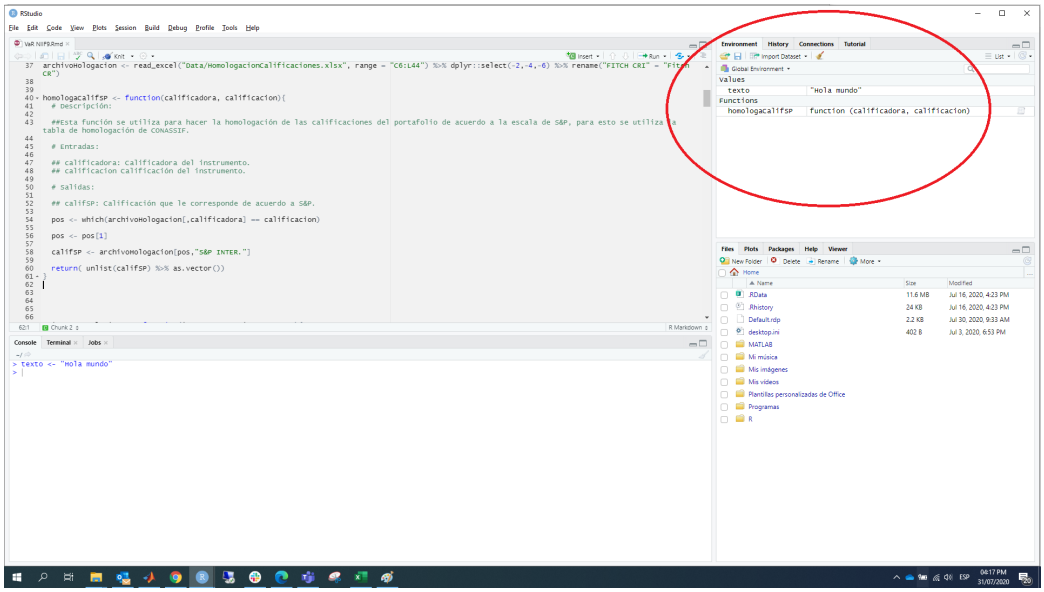
Se puede crear una script con cualquier editor de texto, pero nosotros lo haremos desde RStudio. Para hacer esto, seleccionamos la siguiente ruta de menús: File > New File > R script



2.2.5. Entorno

El panel de entorno esta compuesto de dos pestañas: Environment y History.

En el entorno se irán registrando los objetos que vayamos creando en la sesión de trabajo: datos, variables, funciones. También tenemos la opción de cargar y guardar una sesión de trabajo, importar datos y limpiar los objetos de la sesión. Estas opciones están accesibles a través de las de opciones de la pestaña.



2.2.6. Directorio de trabajo

El directorio o carpeta de trabajo es el lugar en la computadora en el que se encuentran los archivos con los que se van a trabajar en R. Este es el lugar donde R buscara archivos para importarlos y al que serán exportados, a menos que indiquemos otra cosa.

Para encontrar cuál es el directorio de trabajo actual se utiliza la función **getwd()**.

```
getwd()
```

```
## [1] "/Users/tchavarria/Documents/GitHub/curso-programacion-basico-r-bn"
```

Se mostrará en la consola la ruta del directorio que está usando R.

Se puede cambiar el directorio de trabajo usando la función setwd(), dando como argumento la ruta del directorio que se desea utilizar.

```
setwd("otra_ruta")
```

2.3. Paquetes

Cada paquete es una colección de funciones diseñadas para atender una tarea específica. Por ejemplo, hay paquetes para trabajo visualización, conexiones a bases de datos, minería de datos, interacción con servicios de internet, entre otros.

Estos paquetes se encuentran alojados en CRAN, así que pasan por un control riguroso antes de estar disponibles para su uso generalizado.

Se pueden instalar paquetes usando la función **install.packages()**, dando como argumento el nombre del paquete que deseamos instalar, entre comillas.

Por ejemplo, para instalar el paquete **dplyr**, ejecutamos lo siguiente.

```
install.packages("dplyr") ## En general se escribe  install.packages("nombre_paquete
```

Después de ejecutar esa instrucción, aparecerán algunos mensajes en la consola mostrando el avance de la instalación

Una vez concluida la instalación de un paquete, para poder utilizar sus funciones debemos ejecutar la función **library()** con el nombre del paquete que se quiere utilizar.

```
library(dplyr)   ## En general se escribe  library("nombre_paquete")
```

2.4. Scripts

Los scripts son documentos de texto con la extensión de archivo .R, por ejemplo mi_script.R.

Estos archivos son iguales a cualquier documentos de texto, pero R los puede leer y ejecutar el código que contienen.

Aunque R permite el uso interactivo, es recomendable guardar el código en un archivo .R, de esta manera se puede utilizar después y compartirlo con otras personas. En general, en proyectos complejos, es posible que sean necesarios múltiples scripts para distintos fines.

Se pueden abrir y ejecutar scripts en R usando la función **source()**, esta recibe como argumento la ruta del archivo .R en nuestra computadora, entre comillas.

Por ejemplo.

```
source("C:/Proyecto/limpiezaDatos.R")
```

Cuando usamos RStudio y abrimos un script con extensión .R, este programa abre un panel en el que se puede ver su contenido. De este modo se puede ejecutar todo el código que contiene o sólo partes de él.

2.5. Shortcuts

- Borrar toda la consola: CTRL + L.
- Ejecutar una línea o lo que se seleccione: CTRL+R

Capítulo 3

Objetos en R.

En R tenemos 5 clases de objeto básicos o atómicos:

- character
- numeric
- integer
- complex
- logical (TRUE/FALSE)

```
tipo.bien <- "Vivienda"   ## character

saldo <- 130500.34        ## numeric

meses <- 13               ## numeric

dias.mora <- 10L          ## integer

complejo <- 1 + 3i         ## complex

cobro.judicial <- TRUE    ## logical
```

Números.

- En R los número en general se tratan como objetos numeric (i.e números reales de doble precisión.)

- Existe el valor **Inf** que representa infinito y se asocia a operaciones como : 1/0.

```
1 / 0
```

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

```
-1 / 0
```

```
## [1] -Inf
```

```
100 / Inf
```

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

- El valor **NaN** significa not a number, este se asocia generalmente a datos ausentes pero también a una operación del tipo 0/0 que no está definida.

Atributos

Los objetos en R pueden tener los siguientes atributos

- names, dimnames (matrices, data frames)
- dimension (matrices, data frames)
- class
- length

más adelante veremos que con detalle el uso de estos.

Capítulo 4

Operadores

Operadores aritméticos

En R tenemos los siguientes operadores aritméticos:

Operador	Operación	Ejemplo	Resultado
+	Suma	3+1	4
-	Resta	4-6	-2
	Multiplicación	4*6	24
/	División	14/5	2.8
^	Potencia	2^3	8
%%	División entera	5%%2	1

Operadores relacionales

Los operadores relacionales son usados para hacer comparaciones y siempre devuelven como resultado TRUE o FALSE (verdadero o falso, respectivamente).

Operador	Operación	Ejemplo	Resultado
<	Menor estricto	10 < 3	FALSE
<=	Menor o igual	10 <= 3	FALSE
>	Mayor estricto	10 > 3	TRUE
>=	Mayor o igual	10 >= 3	TRUE
==	Igual	10 == 3	FALSE

Operador	Operación	Ejemplo	Resultado
!=	Distinto	10 != 3	TRUE

Operadores lógicos

Los operadores lógicos son usados para operaciones de álgebra Booleana, es decir, para describir relaciones lógicas, expresadas como verdadero (TRUE) o falso (FALSO).

Operador	Operación
|	or
&	and (conjunción)
!	negación

Los operadores | y & siguen estas reglas:

- | devuelve TRUE si alguno de los datos es TRUE
- & solo devuelve TRUE si ambos datos es TRUE
- | solo devuelve FALSE si ambos datos son FALSE
- & devuelve FALSE si alguno de los datos es FALSE

```
edad <- 16
notas <- 83

beca1 <- (edad > 18 & notas > 80)

beca1
```

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

```
beca2 <- (edad > 18 | notas > 80)

beca2
```

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

Capítulo 5

Estructuras de datos.

Las estructuras de datos básicas de R se pueden agrupar por su dimensionalidad y según si son homogéneas (todos los elementos son del mismo tipo) o heterogéneas (hay elementos de distintos tipos). En el siguiente cuadro se resumen estas:

Dimensión	Homogéneas	Heterogéneas
1d	Vector	Lista
2d	Matriz	Data Frame

5.1. Vectores

Los vectores en R son fundamentales ya que, es una de las estructuras de datos más utilizada, la propiedad más importante de los vectores, es que **solo pueden contener objetos de la misma clase**.

Vectores vacíos pueden crearse con la función **vector()**, por ejemplo:

```
vector("numeric", length = 10) ## Tiene los parámetros clase y longitud.
```

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

```
vector("character", 3)
```

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

Sin embargo, la forma más común para crear vectores es utilizando la función **c()** que hace referencia a la palabra concatenar.

```
vector.numerico <- c(1, 2, 3.5)   ## numeric

vector.logical <- c(TRUE, FALSE, T, T, F) ## logical

vector.char <- c("Azul", "Blanco", "Verde") ## character

vector.entero <- 1:13   ## integer
```

```
vector.numerico
```

```
## [1] 1.0 2.0 3.5
```

```
vector.logical
```

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

```
vector.char
```

```
## [1] "Azul"   "Blanco" "Verde"
```

```
vector.entero
```

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

5.1.1. Coerción

Como dijimos anteriormente los vectores solo contienen una misma clase de datos, sin embargo, cuando mezclamos diferentes clases ocurre automáticamente un proceso que se llama **coerción**, esto básicamente transforma todos los elementos del vector a una misma clase.

```
vector.prueba1 <- c(1.3, "Diego") # character
vector.prueba1
```

```
## [1] "1.3"   "Diego"
```

```
vector.prueba2 <- c(TRUE, 13, FALSE) # numeric  #TRUE =1 , # FALSE =0
vector.prueba2
```

```
## [1]  1 13  0
```

Una función muy útil es **length** ya que nos devuelve el tamaño de nuestro vector.

```
length(vector.prueba2)
```

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

Reglas de coerción:

- Valores lógicos se convierten en numéricos: TRUE = 1, FALSE=0.
- El orden de coerción es el siguiente:
    - logical -> integer -> numeric -> character

Es decir todos los elementos del vector se van a transformar a la clase correspondiente siguiendo el orden anterior.

```
vector.prueba3 <- c(12, TRUE, "Azul")

vector.prueba3
```

```
## [1] "12"   "TRUE" "Azul"
```

Esto es lo que hace R de forma automática, sin embargo, podemos realizar la coerción de forma explícita utilizando la función **as.***

```
# Vector numerico 0,1,2,3,4,5.
x <- 0:5
# Se transforma a clase logical
as.logical(x) # 0 es FALSE, y cualquier otro número es TRUE.
```

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

```
# Se transforma a clase character
as.character(x)
```

```
## [1] "0" "1" "2" "3" "4" "5"
```

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

```
## [1] "0" "1" "2" "3" "4" "5"
```

Al hacer la coerción de forma explícita es común obtener el siguiente warning:

"Nas introduced by coercion"

Esto significa que dentro del vector hay valores que no tiene sentido convertirlos a la clase que queremos, por ejemplo, convertir

```
y <- c("A", "B", "C")
as.numeric(y)
```

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

Tal vez en el ejemplo anterior no tiene mucho sentido, sin embargo puede pasar que haya una variable que debería contener solo números, pero aparece una letra por error.

```
y <- c(1:3, "A", 5:7)
y
```

```
## [1] "1" "2" "3" "A" "5" "6" "7"
```

```
as.numeric(y)
```

```
## [1]  1  2  3 NA  5  6  7
```

5.2. Listas

Las listas son un tipo especial de vectores, cuya principal diferencia es que puede contener diferentes clases de elementos, básicamente es un vector, cuyas entradas son vectores, lo podemos crear con la función **list()**, por ejemplo

```
lista1 <- list(1, "A", TRUE) # integer, character, logical
lista1
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "A"
##
## [[3]]
## [1] TRUE
```

```
lista2 <- list(c(100, 200, 300), c("A", "B", "C"), c(T, F, T)) # numeric, cha
lista2
```

```
## [[1]]
## [1] 100 200 300
##
## [[2]]
## [1] "A" "B" "C"
##
## [[3]]
## [1]  TRUE FALSE  TRUE
```

Como vemos nuestra lista esta conformada por tres vectores, y cada uno de estos son de clases distintas. Cuando creamos listas lo más común es nombrar cada uno de los vectores para luego poder extraer esos datos con el signo "$."

```
lista_clientes <- list(saldo = runif(3, min = 1000, max = 2000), nombres = c("
lista_clientes
```

```
## $saldo
## [1] 1900.874 1836.383 1050.230
##
## $nombres
## [1] "Juan"   "Luis"   "Carlos"
##
## $tarjeta.credito
## [1]  TRUE  TRUE FALSE
```

```
# runif(cantidad, min, max) genera números aleatorios.
```

```
lista_clientes$saldo
```

```
## [1] 1900.874 1836.383 1050.230
```

Otro ejemplo:

```
datos.lucas <- list(Nombre = "Lucas", Edad = 33, Tarjeta.Credito = FALSE)

datos.lucas
```

```
## $Nombre
## [1] "Lucas"
##
## $Edad
## [1] 33
##
## $Tarjeta.Credito
## [1] FALSE
```

Una ventaja del objeto lista es que podemos accesar a cada uno de sus argumentos mediante el símbolo "$," por ejemplo si quisieramos ver su Edad y luego si posee tarjeta de crédito o no, podemos escribir

```
datos.lucas$Edad
```

```
## [1] 33
```

```
datos.lucas$Tarjeta.Credito
```

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

5.3. Matrices

Las matrices son vectores pero con el atributo de dimensión. Este atributo es un vector de longitud 2 que contiene el número de filas y el número de columnas (nrow,ncol). Las matrices se crean con la función **matrix**

```
matriz1 <- matrix(nrow = 2, ncol = 3) # No le doy los valores entonces coloca NA en

matriz1
```

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

Esta función nos devuelve la cantidad de filas y columnas de nuestra matriz.

```
dim.matriz <- dim(matriz1)

dim.matriz
```

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

Es muy común utilizar solo uno de estos valores, estos se pueden obtener mediante la siguiente instrucción.

```
filas <-  dim.matriz[1]
columnas <- dim.matriz[2]

filas
```

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

```
columnas
```

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

Las matrices se construyen por defecto por columnas, empezando por el valor de la entrada $(1, 1)$.

```
matriz2 <- matrix(1:9, nrow = 3, ncol = 3)
matriz2
```

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

Sin embargo se puede cambiar a que se construyan por filas asignando el parámetro **byrow** en TRUE.

```
matriz2 <- matrix(1:9, nrow = 3, ncol = 3, byrow = T)
matriz2
```

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

Un par de funciones útiles a la hora de crear matrices o conjuntos de datos son **rbind** y **cbind** que permiten concatenar objetos en forma de filas y columnas respectivamente.

```
x <- 1:5

y <- 11:15

rbind(x, y)
```

```
##   [,1] [,2] [,3] [,4] [,5]
## x    1    2    3    4    5
## y   11   12   13   14   15
```

```
cbind(x, y)
```

```
##      x  y
## [1,] 1 11
## [2,] 2 12
## [3,] 3 13
## [4,] 4 14
## [5,] 5 15
```

5.4. Factores

Esta clase de datos se utiliza para representar datos categóricos. Estos pueden ser ordenados y sin orden. Podemos pensar en los factores como un vector de enteros, donde cada número representa una categoría.

- Los factores tienen un tratamiento especial en las funciones de modelación como lm() y glm(). (Regresiones lineales)
- Es mejor utilizar factores que utilizar enteros, por ejemplo tener la variable Estado civil, con los valores "Casado," "Soltero" es mejor que utilizar los valores 1 y 2.

La función para crear variables categóricas es **factor()**

```
estado.deuda <- factor(c("NORMAL", "NORMAL", "VENCIDA", "COBRO JUDICIAL", "VEN
estado.deuda
```

```
## [1] NORMAL         NORMAL         VENCIDA        COBRO JUDICIAL VENCIDA
## Levels: COBRO JUDICIAL NORMAL VENCIDA
```

Como podemos ver, al imprimir nuestra variable categórica tenemos tres niveles : COBRO JUDICIAL NORMAL VENCIDA. Estos representan las categorías en nuestra variable. R automáticamente hace está asignación por orden alfabético, si queremos definir nosotros el orden, podemos hacerlo utilizando el parámetro **levels**.

```
estado.deuda <- factor(c("NORMAL", "NORMAL", "VENCIDA", "COBRO JUDICIAL", "VEN
estado.deuda
```

```
## [1] NORMAL         NORMAL         VENCIDA        COBRO JUDICIAL VENCIDA
## Levels: NORMAL VENCIDA COBRO JUDICIAL
```

Una forma de saber cuantos individuos hay en cada categoría es mediante la función **table()**.

```
table(estado.deuda)
```

```
## estado.deuda
##         NORMAL        VENCIDA COBRO JUDICIAL
##              2              2              1
```

5.5. Data Frames

Los data frames son la estructura de datos que se utiliza con más frecuencia, es la forma de almacenar datos en forma de tabla.

- Se pueden pensar como un tipo especial de lista donde cada elemento de la lista tienen la misma cantidad de elementos.
- Cada elemento de la lista, son las variables y representan las columnas de la tabla, y la cantidad de elementos el numero de filas (individuos).
- A diferencia de las matrices data frames pueden almacenar distintas clases de datos (como las listas).
- Los data frames tienen los atributos **row.names** y **col.names**.
- Usualmente se crean leyendo datos con las funciones **read.table()** y **read.csv()**, pero también podemos utilizar la función **data.frame()**.
- Se pueden convertir a matrices utilizando la función **data.matrix()** o **as.matrix()**.

```
# Creamos las variables de nuestro data frame.

emisor <- c("BL", "BARCL", "CSGF", "CVS", "DBK", "G", "NOMUR", "USTES", "BNSFI")

monto.facial <- c(5000000, 2500000, 10000000, 40000000, 5000000, 40000000, 10000000,

categoria <- c("COSTO AMORTIZADO", "COSTO AMORTIZADO", "COSTO AMORTIZADO", "VR CON CA
```

```
calificacion.SP <- c("BBB+", "AA+", "B+", "B+", "B", "B", "A+", "BB-", "BB")

isin <- c("US06738EAL92", "US225433AH43", "US126650CK42", "XS2127535131", "CRG

# Creamos el data frame.

portafolio <- data.frame(ISIN = isin, Emisor = emisor, Monto.Facial = monto.fa


portafolio
```

```
##           ISIN Emisor Monto.Facial       Categoria_NIIF Calificacion
## 1 US06738EAL92     BL      5.0e+06     COSTO AMORTIZADO         BBB+
## 2 US225433AH43  BARCL      2.5e+06     COSTO AMORTIZADO          AA+
## 3 US126650CK42   CSGF      1.0e+07     COSTO AMORTIZADO           B+
## 4 XS2127535131    CVS      4.0e+07 VR CON CAMBIO EN ORI           B+
## 5 CRG0000B82H3    DBK      5.0e+06     COSTO AMORTIZADO            B
## 6 US404280BJ78      G      4.0e+07 VR CON CAMBIO EN P/G            B
## 7 ONRBNCR00465  NOMUR      1.0e+07     COSTO AMORTIZADO           A+
## 8 US9127962Z13  USTES      5.6e+06     COSTO AMORTIZADO          BB-
## 9 US9128283G32  BNSFI      5.0e+07     COSTO AMORTIZADO           BB
```

```
dim(portafolio)
```

```
## [1] 9 5
```

5.6. Valores ausentes

Los valores ausentes se denotan por **NA** (not avaiable) o **NaN**(not a number), las siguientes funciones se utilizan para verificar y encontrar valores ausentes.

1. **is.na()**: Se utiliza para verificar y encontrar los valores **NA** en un objeto.

2. **is.nan()**: Se utiliza para verificar y encontrar los valores **NaN** en un objeto.
3. Un valor **NaN** es un **NA** pero la otra dirección no es cierta.

```
## Creamos un vector que contenga un NA
x <- c(1, 3, NA, 10, 3)

## Retorna un vector de la misma longitud de x, con TRUE donde hay un NA y FALSE do
is.na(x)
```

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

```
## Creamos un vector que contenga un NA y NaN.
x <- c(1, 3, NA, 10, 3, NaN)

## Retorna un vector de la misma longitud de x, con TRUE donde hay un NA y FALSE do
is.na(x)
```

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

```
## Retorna un vector de la misma longitud de x, con TRUE donde hay un NA y FALSE do
is.nan(x)
```

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

con el ejemplo anterior se puede verificar el punto **3.**.