Fundamentals R

AUTHOR
Juan David Dueñas Garavito & Germán Camilo Rodriguez

Introducción

En este documento se abordarán los fundamentos del lenguaje de programación estadística R. R es uno de los lenguajes más populares para el análisis de datos y la ciencia de datos y es utilizado por muchos profesionales y organizaciones en todo el mundo. El objetivo de este documento es proporcionar una guía práctica para aquellos que quieren aprender a programar en R, así como para aquellos que ya tienen experiencia en R pero quieren mejorar sus habilidades y ampliar sus conocimientos.

Este documento está diseñado para ser utilizado como una referencia práctica para aquellos que trabajan con R en su trabajo o proyectos personales. Al final del documento, se espera que los lectores tengan las habilidades necesarias para trabajar con datos en R y aplicar técnicas de análisis de datos y visualización en sus proyectos de ciencia de datos.

Acerca de R

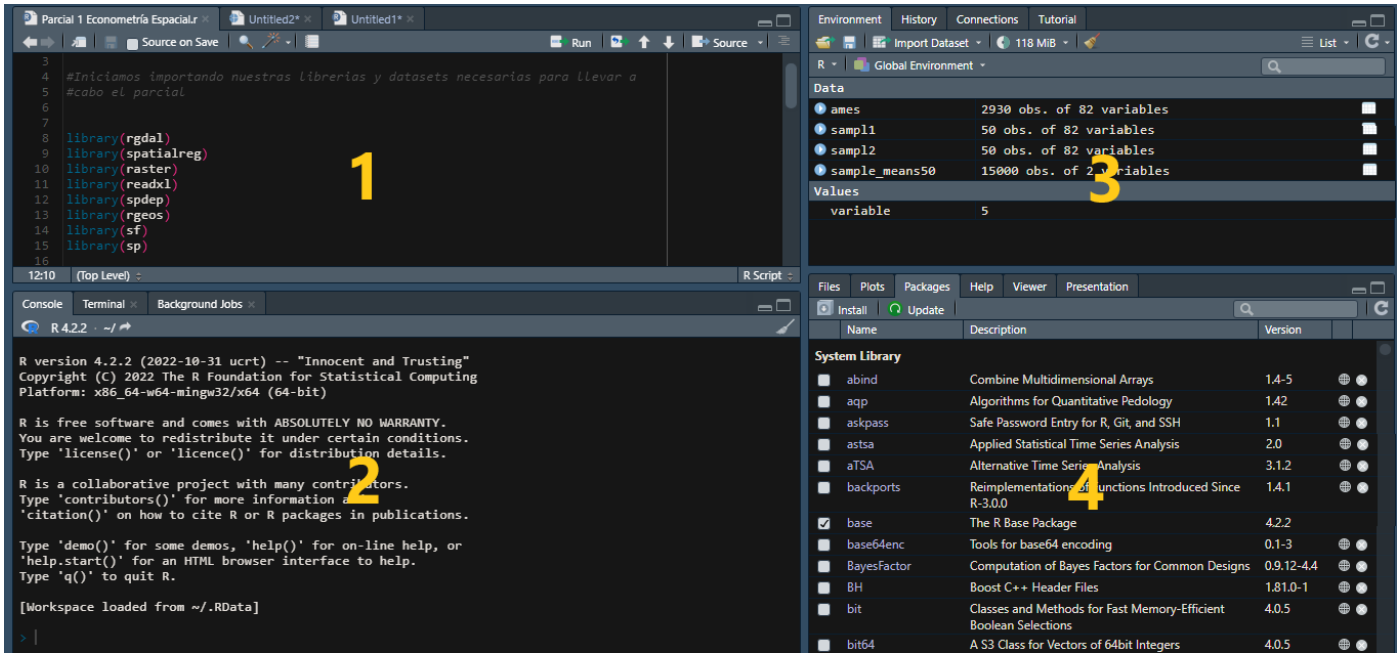
R es un lenguaje de programación y entorno de desarrollo estadistico gratuito y de codigo abierto, cualquier persona puede descargarlo utilizarlo y modificarlo esto permite que tenga una gran comunidad y documentación bien detallada de cada una de sus funcionalidades y paquetes, R es orientado a objetos lo que significa que los datos se organizan en objetos que se pueden manipular y analizar, R cuenta con una interfaz gráfica de usuario llamada RStudio que facilita el trabajo con el lenguaje y permite la creación y visualización de datos de manera sencilla.

R permite hacer diferentes tipos de analisis, Estadistica Descriptiva, Estadistica Bayesiana, Modelos Estado Espacio, entre otros es decir en R podemos hacer macroeconometria, microeconometria, econometría fianciera, evaluación de impacto, econometría bayesiana, econometria espacial entre otros analisis estadisticos.

Interfaz de RStudio

RStudio proporciona una interfaz gráfica de usuario (GUI) que facilita el trabajo con R, lo que lo hace más accesible para los usuarios que no están familiarizados con la programación en línea de comandos.

Al ingresar a RStudio nos vamos a encontrar con cuatro principales paneles:



1. **Editor :** En el editor será donde colocaremos el codigo y los comandos de ejecución.
2. **Consola:** Alli veremos los resultados o los outputs del codigo.
3. **Espacio de Trabajo:** Aqui podemos ver el historial de comandos que hemos usado, las diferentes variables, datasets, objetos y funciones que hemos declarado entre otras cosas
4. **Archivos / graficas / paquetes:** Allí podremos visualizar los plots que generemos en el codigo, las librerias que tengamos instaladas, los archivos de nuestro PC entre otros

Nos vamos a centrar en el trabajo que se hace en el primer panel.

Comentarios

Es una buena practica comentar nuestro codigo, esto significa ir colocando texto no ejecutable que explique que esta haciendo nuestro codigo, esto facilitará más adelante nuestra vida al corregir el codigo y a otras personas que quieran entender lo que estamos haciendo, para comentar el codigo usamos el #, con esto el texto que se encuentre despues del numeral no será ejecutado.

```
#Esto es un comentario
```

Paquetes y limpieza de entorno

R siempre guarda las diferentes variables, datasets entre otros objetos, es por esto que es una buena practica al principio de cada codigo limipar nuestro entorno, esto lo hacemos con el siguiente comando.

```
remove(list = ls())
```

Dado que R es un lenguaje Open Source, los usuarios pueden crear paquetes para facilitar o realizar nuevas operaciones que el R base no puede, sin embargo para hacer uso de estos paquetes debemos instalarlos y

luego activarlos, la instalación solo se hace una vez, la activación si debe hacerse en cada codigo que deseemos volver a abrir RStudio, es una buena practica activar todos los paquetes en las primeras lineas de codigo.

```
options(repos = c(CRAN = "http://cran.rstudio.com"))
install.packages("tinytex")
install.packages("TeX")
install.packages("dplyr")
install.packages("ggplot2")
```

```
library(dplyr)
library(ggplot2)
library(tinytex)
```

Operadores Aritmeticos

R incluye varios operadores aritmeticos que se utilizan para realizar operaciones matematicas. Los operadores aritmeticos incluyen:

- + : Suma dos números
- - : Resta dos números
- * : Multiplica dos números
- / : Divide dos números
- ** : Eleva un número a una potencia

Variables y tipos de objetos

Diremos que una variable es un “contenedor de información” en la que podemos “guardar” diferntes objetos, generalmente se escriben en minúscula, sin espacios ni tildes, comas, puntos, signos de interrogación o cualquier otro caracter a excepción del guión bajo, para declarar variable en R podemos usar <- como tambien =

Para notación de este trabajo usaremos unicamente el igual.

```
nombre_de_la_variable <- "Valor de la variable"
nombre_de_la_variable = "Valor de la variable"
```

para visualizar nuestra variable en la consola (Panel #2) solo basta con llamarla de nuevo o utilizar la función print, o podemos usar un ; y volver a llamar la funciín en la misma linea

```
nombre_de_la_variable
```

```
[1] "Valor de la variable"
```

```
print(nombre_de_la_variable)
```

```
[1] "Valor de la variable"
```

```
nombre_de_la_variable = "Valor de la variable"; nombre_de_la_variable
```

```
[1] "Valor de la variable"
```

Los objetos que guardamos en las variables pueden ser de diferentes tipos y en función de esto la forma de guardarlos cambia.

```
objeto_numeric = 3
objeto_logic = FALSE
objeto_character = "Hola Mundo"
objeto_complex = 1+0i
```

Para verificar de que tipo o clase es un objeto podemos usar la función class, puede darse el caso de que requiero convertir un objeto de un tipo a otro tipo, para esto debo usar alguna de las siguientes funciones según corresponda

```
x = 1
class(x)
```

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

```
as.numeric(x)
```

```
[1] 1
```

```
as.logical(x)
```

```
[1] TRUE
```

```
as.character(x)
```

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

Para hacer operaciones entre variables nos aseguramos que las variables sean de tipo numerico y operamos:

```
a = 5
b = 7
```

```
c = a + b
c
```

```
[1] 12
```

Vectores

Los vectores son entes matematicos que se usan para guardar datos de cierto tipo especifico, para crear vectores en R debemos usar el comando C, los vectores nos permiten hacer operaciones entre varios datos al mismo tiempo sin tener que recurrir a bucles

```
mi_primer_vector = c(25,5,3,6,3,6)
vector_dos = c("Hola", "mundo")
vector_tres = c(FALSE, TRUE, FALSE, TRUE)
```

Como los vectores solamente pueden guardar un tipo de dato si llegamos a introducir diferentes tipos de datos el vector automaticamente va a transformar estos datos para que el vector quede solamente en función de un tipo de dato.

```
y <- c(1.7, "a")   ## character
y <- c(TRUE, 2)    ## numeric
y <- c("a", TRUE)  ## character
```

Podemos crear vectores con determinadas caracteristicas de maneras mucho más eficientes, por ejemplo:

```
#Crear un vector secuencia 1 a 1.
Dias = 0:15; Dias
```

```
 [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

```
#Crar un vector secuencia que avance cada j unidades: seq(a,b,j)
Medicion = seq(1,5, 0.5); Medicion
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
#Crear un vector donde los objetos se repiten n veces: rep(objeto,n)
Numerico = rep(1,10); Numerico
```

```
 [1] 1 1 1 1 1 1 1 1 1 1
```

```
#Crear un vector de repeticiones
Dummy = c(rep(0,5), rep(1,10), rep(0,5)); Dummy
```

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

Para concatenar vectores simplemente debemos:

```
vector_a = c(2,54,3)
vector_b = c(3,7,8)
vector_c = c(vector_a, vector_b); vector_c
```

```
[1]  2 54  3  3  7  8
```

Las operaciones con vectores funcionan de manera similar a las operaciones con variables, en ultimas seguimos operando variables solo que en este caso las variables no contienen un solo valor sino contienen un vector de valores, los vectores entre si deben ser iguales o multiplos. En caso de ser iguales se hará la operación uno a uno, en cambio si son multiplos se repetira la operación en los diferentes vectores, tal que:

```
#Operación uno a uno
vector_a = c(2,54,3)
vector_b = c(3,7,8)
vector_c = vector_a + vector_b; vector_c
```

```
[1]  5 61 11
```

```
#Operación entre multiplos
vector_a = c(2,54,3,4,8,5)
vector_b = c(3,7,8)
vector_c = vector_a + vector_b; vector_c
```

```
[1]  5 61 11  7 15 13
```

podemos hacer estadistica descriptiva de los vectores usando la función summary

```
summary(vector_a)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   2.00    3.25    4.50   12.67    7.25   54.00 
```

entre otros posibles metodos de la clase vector

```
#describir cuales son los elementos que no se repiten del vector
unique(vector_a)
```

```
[1]  2 54  3  4  8  5
```

```
#Ordenar elementos de menor a mayor
sort(vector_a)
```

```
[1]  2  3  4  5  8 54
```

```r
#Posición ordenada de menor a mayor
order(vector_a)
```

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

```r
#Ordenar elementos de mayor a menor:
sort(vector_a, decreasing = TRUE)
```

```
[1] 54  8  5  4  3  2
```

Matrices

Una matriz es una estructura de datos bidimensional que consiste en una colección de elementos organizados en filas y columnas. Se puede representar como una tabla o cuadrícula de números o valores. Cada elemento en la matriz se identifica por su posición, que se determina por el número de fila y el número de columna.

Para crear matrices en R podemos hacerlo directamente o a partir de vectores previamente creados, para crear matrices directamente usamos:

```r
MAT1=matrix(1:10,nrow = 5,byrow = TRUE);MAT1
```

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

```r
MAT2=matrix(10:19,ncol=5, nrow = 2, byrow = FALSE);MAT2
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   12   14   16   18
[2,]   11   13   15   17   19
```

donde el primer parametro corresponde a con que vamos a llenar esas matrices, este intervalo debe ser multiplo del numero de filas o columnas que queremos crear, el segundo si vamos a hacer la creación de la matriz desde la columna o desde las filas, podemos hacerlo tambien desde ambas.

Ahora bien, si queremos crear matrices a partir de vectores ya establecidos previamente usaremos el siguiente codigo:

```r
a = c(1,4,5,7,9,7,10,6,5,7,9)
b = c(3,4,6,8,2,3,11,1,2,5,6)
MAT3 = cbind(a,b); MAT3
```

```
      a  b
 [1,]  1  3
 [2,]  4  4
 [3,]  5  6
 [4,]  7  8
 [5,]  9  2
 [6,]  7  3
 [7,] 10 11
 [8,]  6  1
 [9,]  5  2
[10,]  7  5
[11,]  9  6
```

```
MAT4 = rbind(a,b); MAT4
```

```
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
a    1    4    5    7    9    7   10    6    5     7     9
b    3    4    6    8    2    3   11    1    2     5     6
```

como nos podemos dar cuenta al usar vectores tambien podemos crear las matrices desde las filas o desde las columnas

Operaciones entre matrices

Como sabemos las matrices tienen formas de operarse de manera diferente a los escalares o constantes, por lo cual vamos a detallar cada una de las operaciones principales:

```
#Suma escalar - matriz
MAT1 + 10
```

```
     [,1] [,2]
[1,]   11   12
[2,]   13   14
[3,]   15   16
[4,]   17   18
[5,]   19   20
```

```
#Suma matriz - matriz
MAT3 + t(MAT4)
```

```
      a  b
 [1,]  2  6
 [2,]  8  8
 [3,] 10 12
 [4,] 14 16
 [5,] 18  4
 [6,] 14  6
 [7,] 20 22
 [8,] 12  2
```

```
 [9,] 10  4
[10,] 14 10
[11,] 18 12
```

```
#Multiplicar escalar - matriz
MAT3*10
```

```
       a   b
 [1,]  10  30
 [2,]  40  40
 [3,]  50  60
 [4,]  70  80
 [5,]  90  20
 [6,]  70  30
 [7,] 100 110
 [8,]  60  10
 [9,]  50  20
[10,]  70  50
[11,]  90  60
```

Podemos a su vez tambien crear nuevas matrices especiales

```
#Matriz Transpuesta: t()
t(MAT1)
```

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

```
#Matriz identidad: diag()
diag(5)
```

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

```
#Matriz diagonal: diag()
Mat_diag = diag(1:10)
```

Finalmente calcularemos la multiplicación matricial que recordemos no se hace punto a punto sino por medio de un proceso diferente, tambien la inversa de una matriz y la determinante

```
#Producto de matrices
dim(MAT3) #conocer las dimensiones de una matriz: 11x2
```

```
[1] 11  2
```

```
dim(MAT4) #2x11, son multiplicable. por ende:
```

```
[1]  2 11
```

```
MAT3%*%MAT4
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
 [1,]   10   16   23   31   15   16   43    9   11    22    27
 [2,]   16   32   44   60   44   40   84   28   28    48    60
 [3,]   23   44   61   83   57   53  116   36   37    65    81
 [4,]   31   60   83  113   79   73  158   50   51    89   111
 [5,]   15   44   57   79   85   69  112   56   49    73    93
 [6,]   16   40   53   73   69   58  103   45   41    64    81
 [7,]   43   84  116  158  112  103  221   71   72   125   156
 [8,]    9   28   36   50   56   45   71   37   32    47    60
 [9,]   11   28   37   51   49   41   72   32   29    45    57
[10,]   22   48   65   89   73   64  125   47   45    74    93
[11,]   27   60   81  111   93   81  156   60   57    93   117
```

```
#Inversa de una matriz: solve()
solve(Mat_diag)
```

```
     [,1] [,2]      [,3] [,4] [,5]      [,6]      [,7]  [,8]      [,9] [,10]
 [1,]    1  0.0 0.0000000 0.00  0.0 0.0000000 0.0000000 0.000 0.0000000   0.0
 [2,]    0  0.5 0.0000000 0.00  0.0 0.0000000 0.0000000 0.000 0.0000000   0.0
 [3,]    0  0.0 0.3333333 0.00  0.0 0.0000000 0.0000000 0.000 0.0000000   0.0
 [4,]    0  0.0 0.0000000 0.25  0.0 0.0000000 0.0000000 0.000 0.0000000   0.0
 [5,]    0  0.0 0.0000000 0.00  0.2 0.0000000 0.0000000 0.000 0.0000000   0.0
 [6,]    0  0.0 0.0000000 0.00  0.0 0.1666667 0.0000000 0.000 0.0000000   0.0
 [7,]    0  0.0 0.0000000 0.00  0.0 0.0000000 0.1428571 0.000 0.0000000   0.0
 [8,]    0  0.0 0.0000000 0.00  0.0 0.0000000 0.0000000 0.125 0.0000000   0.0
 [9,]    0  0.0 0.0000000 0.00  0.0 0.0000000 0.0000000 0.000 0.1111111   0.0
[10,]    0  0.0 0.0000000 0.00  0.0 0.0000000 0.0000000 0.000 0.0000000   0.1
```

```
#Determinante de la matriz: det()
det(Mat_diag)
```

```
[1] 3628800
```

Estos anteriores procesos si no cumplen con determinadas caracteristicas no se pueden llevar a cabo, solo se puede invertir matrices invertibles, es decir que su determinante sea diferente de 0, entre otras caracteristicas

para acceder a elementos especificos de la matriz debemos hacer uso del indice en formato (x,y) tal que sea (fila, columna)

```
MAT2
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   12   14   16   18
[2,]   11   13   15   17   19
```

```
#Mostrar el elemento ij de la matriz: MAT[i,j]
MAT2[2,1]
```

```
[1] 11
```

```
MAT2[1,2]
```

```
[1] 12
```

podemos hacer esto tambien para mostrar solo una fila o solo una columna de la matriz

```
MAT2
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   12   14   16   18
[2,]   11   13   15   17   19
```

```
#Mostrar los elementos de la fila i: MAT[i,]
MAT2[1,]
```

```
[1] 10 12 14 16 18
```

```
#Mostrar los elmentos de la columna j: MAT[,j]
MAT2[,1]
```

```
[1] 10 11
```

para mejor entendimiento de nuestra matriz podemos cambiarle los nombres a las filas y las columnas, esto tambien nos facilitará el trabajo posterior

```
rownames(MAT1) = c("a","b","c","d","e")
colnames(MAT1) = c("f","g")
MAT1
```

```
  f  g
a 1  2
b 3  4
c 5  6
d 7  8
e 9 10
```

Operadores logicos y de comparación

Los operadores logicos se utilizan para combinar expresiones lógicas y producir un resultado verdadero o falso. Además, se pueden utilizar los operadores de comparación para crear expresiones lógicas. Los operadores logicos y de comparación entre vectores de las matrices son los siguientes:

```
#Elementos mayores o iguales a j en el vector
a>=4
```

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

```
#Elementos menores o iguales a j en el vector
a<=4
```

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

```
#Elementos iguales a j en el vector
a==4
```

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

```
#Elementos diferentes a i en el vector
b!=6
```

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

```
#Dos afirmaciones verdaderas: "&"
a>=1 & b==3 # Recuerden que & es un "y" matemático
```

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

```
#Al menos una es cierta: "|"
a>=1 | b==3 # Recuerden que | es un "o" matemático
```

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

```
#Negación: "!"
!a < 0 #a no es menor que cero
```

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

Na y NaN

Cuando el codigo nos reporta un dato Na tenemos un missing value, es decir, datos vacios. Por otra parte cuando tenemos un NaN se traduce como un “non a number” es decir que el resultado de la operación no se puede calcular dado que es indeterminado