

Introducción a R y RStudio

Esenciales de R



Zulemma Bazurto Blacio

Agenda

Introducción a herramientas

RStudio y sus ambientes

Buenas prácticas en R

Tipos de datos

Estructuras de datos

Los primeros pasos en análisis de datos

El proceso de tidyverse

Importando datos con R

Exploración y manipulación de datos

¿Qué es R?

"Lenguaje de programación para entender los datos"
Hadley Wickham, Chief Scientist, Rstudio



¿Por qué R?

- Software libre
- De código abierto
- Flexible
- Con contribución de la comunidad

R-Ecosistema

- Paquetes libres y pagados que extienden las capacidades de R
- Software de análisis basado en R
- Interfaces gráficas de desarrollo (IDEs)
- Interfaces gráficas de usuario (GUIs)
- Integración de Bases de datos con R
- Integración de herramientas de BI con R

RStudio

- Dato curioso: Antes era el nombre de la compañía que creaba/desarrollaba productos de software. ¡Hoy Posit!



- El producto más conocido es RStudio IDE (Integrated Developed Environment)

¡Echando a andar RStudio!

En windows y Mac

1. R (base): Comprehensive R Archive Network (CRAN)

<https://cran.r-project.org/>

2. RStudio IDE:

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

¿Podemos instalar primero RStudio IDE y luego R
(base)?

En Linux (Distribuciones)

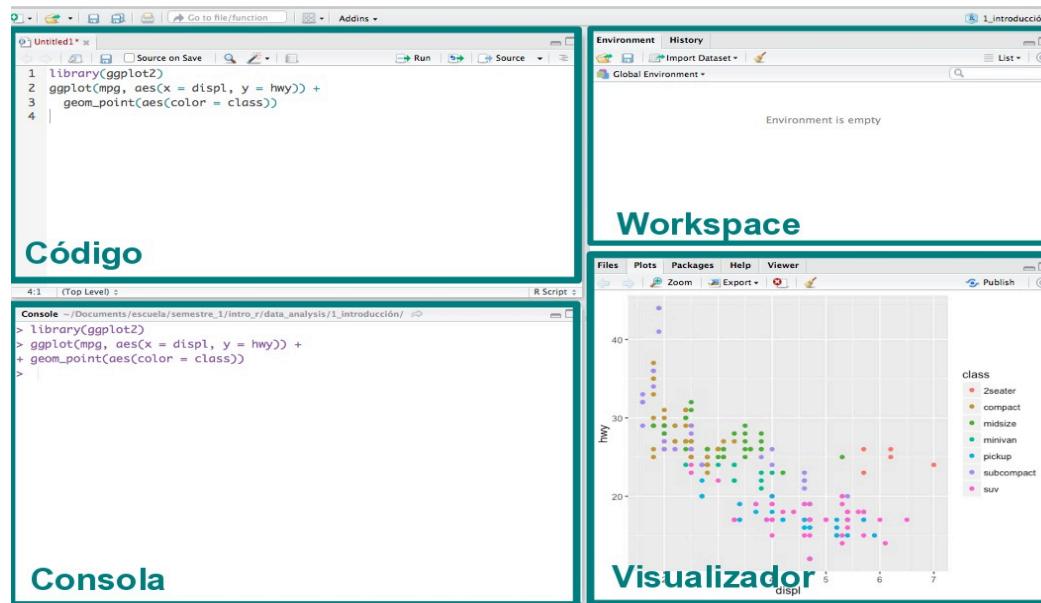
- Distribuciones basadas en Debian/Ubuntu tienen R en los repositorios oficiales
- Distribuciones basadas en Fedora/RedHat deben habilitar EPEL para tener R
- En Debian Estable para tener nuevas versiones se debe utilizar un ["backports"]

Pero antes necesitamos instalar: (<https://cran.r-project.org/bin/linux/debian/>)

Ventajas de R-Studio

- Todo en 1 ventana: Console, Workspace, History, Working directory, Files, Plot, Packages y Help
- Integración de la consola de R
- Ejecutar código desde script
- Resaltado de sintaxis
- Completado de sintaxis
- Manejo de proyectos con soporte para Git y Subversion
- Herramientas para Investigación Reproducible (knitr)

Si abres RStudio debe lucir así:

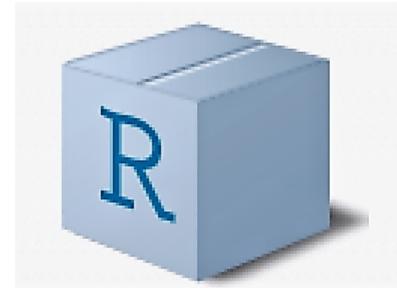


Ambientes de RStudio: una amigable analogía



¿Qué es un paquete?

- Contienen datos, funciones y códigos.
- En analogía, son como aplicaciones para celulares.
- Existen aproximadamente 19679 paquetes disponibles en CRAN que abordan una amplia gama de tópicos.



Instalemos algunos:

```
install.packages("tidyverse", repos = 'http://cran.us.r-project.org')
install.packages("dplyr", repos = 'http://cran.us.r-project.org')
install.packages("animation", repos = 'http://cran.us.r-project.org')
```

¿Sabes qué acabas de hacer?
Escribir tus primeras líneas de código en R.
¡Buen trabajo!

Pero, ¿puedes instalar paquetes de otra manera?

Vamos nuevamente al ambiente que denominamos "Visualizador"



¿Cuál es la diferencia entre un error y un warning?

```
install.packages(tidyverse)
```

Error in install.packages : object 'tidyverse' not found

```
library(gapminder)
```

Warning message: package 'gapminder' was built under R version 4.1.3

¿Qué novedades presentan los códigos anteriores?

Para trabajar eficientemente, necesitamos un ambiente de orden en R



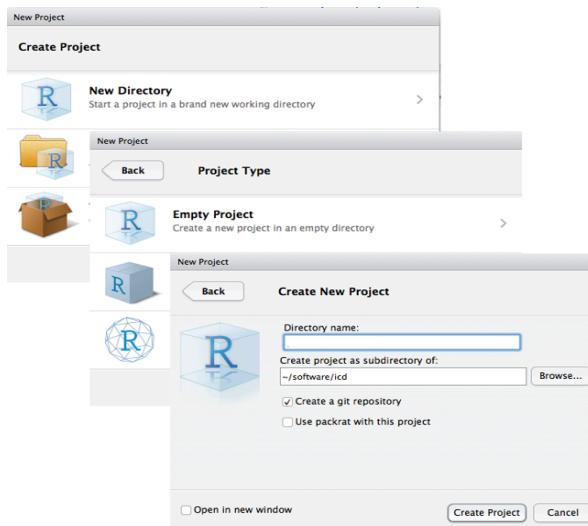
Por eso "**CREAMOS PROYECTOS**" en R.

Proyectos en R-Studio

- Un **Proyecto** es una carpeta que contiene todos los scripts y archivos .RData y .Rhistory
- Al abrir un proyecto antiguo RStudio lo abre con las pestañas que se tenía activas
- Permite colaboración utilizando GIT o Subversion
- Se sugiere tener una estructura interior, por ejemplo: Scripts, Data, Exports, Info

Iniciar un Proyecto en RStudio

File > New Project > New Directory > Empty Project > Poner nombre al Proyecto (se creará una carpeta con ese nombre) > Create Project



Creando un proyecto: Una mirada a las ventajas de realizarlo



- Permite administrar de mejor manera nuestros códigos y datos.
- Facilita la ubicación de archivos sin cambio de ruta en el computador.
- Rstudio ejecuta sesiones independientes de R par cada proyecto.

Tenemos todo listo para ¡poner manos a la obra!

Empecemos cargando paquetes
¿Y ÉSTO DE QUÉ VA AHORA?



Cargando un paquete

Cargar un paquete significa [activarlo](#)

Para cargar paquetes usamos la función [library\(\)](#)

```
library(tidyverse)
library(gapminder)
library(babynames)
```

Sabemos la función para cargar paquetes pero [no hemos ejecutado el código](#). Las opciones para lo anterior son:

- EL botón [Run](#)
- Ctrl + Enter (En Linux/Windows)

¡A ejecutar código!

Cargando un paquete

Al ejecutar library() los posibles mensajes (avisos) que se pueden obtener son:

- Cómo el paquete se ha cargado
- Conflicto: es un aviso de que las funciones de un paquete tienen el mismo nombre que funciones de otro paquete.

Buenas prácticas: Haciendo eficiente tu código



- LLamar todos los paquetes que ocuparemos al inicio de nuestro script.
¿Y si no sé qué paquetes usar?
- Comentar mi script.
¿Cómo lo hago?
- Al inicio comentar el objetivo y/o contexto del script.

A medida que vayas avanzando, combinarás los primeros pasos aprendidos con nuevos aprendizajes.

Generalidades 1

- Case sensitivity (`Abc` es diferente de `abc`)
- R, aparte de objetos, tiene:

Expresión.- Se evalúa, se imprime y el valor se pierde

```
5+5# Expresión
```

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

Asignación.- Evalúa la expresión y guarda el resultado en una variable (no lo imprime)

```
a <-  
a <-  
◀ □ ▶
```

R como calculadora

```
2 + 3*5
```

```
## [1] 17
```

```
log((1+2+3)/4) # log natural
```

```
## [1] 0.4054651
```

```
pi^2# pi y potencia
```

```
## [1] 9.869604
```

R como calculadora

```
abs(-2) # valor absoluto
```

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

```
factorial(3) # factorial
```

```
## [1] 6
```

```
floor(5.7) # funcion piso
```

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

Generar secuencias, repeticiones y aleatorios



```
1:10# secuencia de 1 a 10, de 1 en 1
```

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

```
seq(from= 0, to= 20, by= 5) # función seq
```

```
## [1] 0 5 10 15 20
```

```
seq(from= 5, by= 5, length.out= 5) # función seq
```

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

Generar secuencias, repeticiones y aleatorios



```
rep(x= 3, times= 5) # repetir 5 veces el # 3  
## [1] 3 3 3 3 3  
  
runif(n= 10, min= 1, max= 5) # Genera aleatorios uniformes  
  
## [1] 2.095761 3.609748 1.185472 2.659404 2.801273 3.955654 3.542410 3.723809 1.637400 1.410638  
  
rnorm(n= 10, mean= 100, sd= 10) # Genera aleatorios normales  
  
## [1] 95.59780 101.03928 102.08814 92.27226 99.96391 111.67140 102.57572 118.62975 92.67260 85.848
```

Asignaciones

Asigna el valor '5' a la variable 'a':

- `a <- 5`
- `5 -> a`
- `assign("a", 5)`

Asigna globalmente el valor '5' a la variable 'a', (dentro de una función 'a' seguirá valiendo 5):

- `a <<- 5`
- `5 ->> a`

No se recomienda usar '`a = 5`'. En RStudio verificar que exista la variable 'a' en la pestaña Environment.

Asignaciones

El resultado de una función de un objeto X puede ser asignada al mismo objeto X en la misma sentencia, es decir

```
a <- 5 # Expresión
```

```
a
```

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

```
a <- 2*a
```

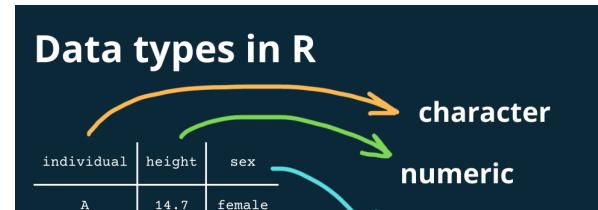
```
a
```

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

Tipos de datos

Existen los siguientes tipos de datos

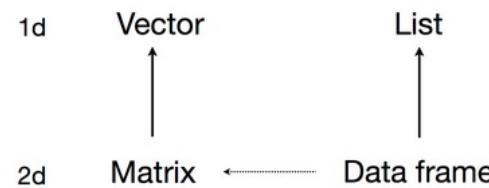
- Numérico (num)
- Carácter (chr)
- Factor
- Lógico
- Fecha (date)



Estructura de datos

Existen 5 estructuras de datos básicos:

- Vector
- Matriz
- Array
- Data Frame
- Lista



El uso de las estructuras de datos se diferencian por la clase de sus elementos:

	Homogeneous	Heterogeneous
1d	Atomic vector	List
2d	Matrix	Data frame
nd	Array	

Tipos de vectores

Numéricos

```
num_vec <- c(-1, 2.5, 3, 4, 5.1)
```

Carácter

```
cha_vec <- c("Mon", "Tue", "Wed", "Thu", "Sat", "Sun")
```

Lógico

```
boo_vec <- c(TRUE, FALSE, FALSE, TRUE, TRUE, FALSE)
```

¿Qué es c()?

Factores

Un factor en R es un tipo de vector con un enfoque estadístico que se usa para variables categóricas. Recuerda que las variables categóricas pueden ser medidas por 2 tipos de escala:

- Nominal
- Ordinal

En R un factor se define con la función `factor()`

```
#Variable categórica nominal
sex_vec <- c("F", "M", "M", "F", "M")
```

¿Qué sucede con los niveles de la variable en esta escala?

Vamos por las categóricas ordinales!

NOTA: Toma en cuenta qué pasa con los niveles por la naturaleza de la escala de medición

Si construimos el factor "temp_vec":

```
temp_fct <- factor(temp_vec,  
                     levels = c("Low", "Medium", "High"),  
                     ordered = T)
```

De manera que luciría

```
## [1] High    Low     Medium  Low     Low     Medium  High    Low     Medium  Low     Low  
## Levels: Low < Medium < High
```

temp
◀ □ ▶

```
temp_vec <- c("High", "Low", "Medium", "Low",
"Low", "Medium", "High", "Low",
"Medium", "Low", "Low")
temp_fct <- factor(temp_vec,
                     levels = c("Low", "Medium", "High"),
                     ordered = T)
temp_fct

## [1] High     Low      Medium Low      Low      Medium High    Low      Medium Low      Low
## Levels: Low < Medium < High
```

Modifiquemos los nombres de las etiquetas. Es muy útil cuando aquellos nombres son muy extensos.

```
levels(temp_fct) <- c("L", "M", "H")
temp_fct
```

```
## [1] H L M L L M H L M L L
## Levels: L < M < H
```

¿Y si necesito resumir la información del vector?
Tenemos otra función: `summary()`

Funciones útiles asociadas a este tipo de dato son: pueden ser medidas por 2 tipos de escala:

- Combinar o concatenar: `c()`
- Clase: `class()`
- Longitud de: `length()`
- Imprimir: `print()`

¡ A probarlas!

NOTA: Una forma de aprender la lógica de R, es probar lo que crees que debería ser como resultado contrastado con tu razonamiento en funciones de R.

Un breve vistazo a matrices

Una matriz es un arreglo de dos dimensiones en el que todos los elementos son del mismo tipo, por ejemplo: numéricos

Sintaxis de la función `matrix()`:

```
matrix(data,nrow,ncol,byrow)
```

`matrix()` crea una matriz de un vector especificando dimensiones.

```
matrix(data = 1:9, nrow = 3, ncol = 3, byrow = F)
```

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

Ejemplo

En el siguiente vector se presentan los ingresos totales y de lanzamiento de cada película de la saga Harry Potter

```
sales_hp <- c(497066400, 426630300, 401608200, 399302200, 377314200,  
359788300, 357233500, 328833900, 141823200, 189432500,  
142414700, 135197600, 99635700, 92756000, 134119300,  
138752100)
```

El reto: Toma el vector `sales_hp` y crea una matriz con 8 filas llenándola por columnas. **¿Necesito especificar las columnas?**

Funciones más útiles para las matrices

- `dim()`: entrega la dimensión de la matriz
- `nrow()`: entrega el número de filas de la matriz
- `ncol()`: entrega el número de columnas de la matriz
- `[,]`: selecciona los elementos de una matriz considerando las dimensiones de la misma.
- `rownames()`: agrega nombres a las filas de la matriz
- `colnames()`: agrega nombres a las columnas de la matriz

```
sales_hp <- matrix(sales_hp, nrow = 8)

dim(sales_hp)
nrow(sales_hp)
ncol(sales_hp)
```

El más usado: Dataframe

Un dataframe es un objeto de dos dimensiones en R. Puede verse como un arreglo de vectores de la misma dimensión, similar a una matriz.

¿Cuál es la diferencia con una matriz?

La función `dataframe()` permite:

- Crear un nuevo dataframe
- Transformar una matriz a dataframe

El más usado: Dataframe

```
#Creando un dataframe
muestra_df <- data.frame(secuencia = 1:5,
                           aleatorio = rnorm(5),
                           letras = c("a", "b", "c", "d", "e"))
muestra_df

##   secuencia   aleatorio letras
## 1      1 -1.38931429      a
## 2      2  0.07119483      b
## 3      3  1.48304348      c
## 4      4  0.06568348      d
## 5      5 -0.14767296      e
```

Transformando una matriz en Dataframe

```
#Transformando una matriz a dataframe
sales_df <- data.frame(sales_hp)
head(sales_df) # head() retorna por default las primeras 6 filas
```

```
##     sales_hp
## 1 497066400
## 2 426630300
## 3 401608200
## 4 399302200
## 5 377314200
## 6 359788300
```

Otras funciones para usar en dataframe:

- `rownames()` y `colnames()`
- `$`
- `str`

¡Son titanes!: Listas

Una lista en R es un objeto que permite una estructura de datos complicada, una super estructura. **Esto porque permite reunir diferentes tipos de objetos:**

- Vectores
- Matrices
- Dataframes
- Listas

Muchas funciones que usarás en el futuro, sobre todo de modelación, regresan resultados de estructuras complicadas y lo almacenan en listas. Por ejemplo, la función `lm()`

SEMANA 1: DONE!