Introducción a R

Clase 08

Breve Introducción a Machine Learning

Dr. Ramón Caraballo
SECIU Red Académica Uruguaya
UDELAR

Objetivo

En esta Sesión:

- **Introducción al Lenguaje R**
- **Tipos de Datos y Operadores**
- **Estructuras de Datos en R y Operaciones Básicas**

Introducción a R

Qué es y porqué aprender R?

Ambiente gratuito de Software para Análisis de datos y Visualización

- Lenguage de Programación muy extensible
- Herramienta de Visualización
- Análisis Estadístico, etc.

Ross Ihaka & Robert Gentleman Universidad de Auckland, Nueva Zelanda (1993).
Crearon R como un lenguaje para ayudar a enseñar estadísticas a sus estudiantes. Basaron R en el lenguaje S desarrollado anteriormente en Bell Labs en la década de 1970. Después, pusieron a disposición R como un proyecto GNU de código abierto. La comunidad de R es muy activa y existe en todo el mundo. R se considera un lenguaje específico de dominio, ya que fue diseñado principalmente para análisis de datos. R es un laenguage interpretado.

Usos de R

Como Analista/Científico de Datos
Realizar estudios exahustivos en análisis de datos que no se pueden hacer exclusivamente mediante herramientas de tipo de hoja de cálculo.

Como Desarrollador de Software
Cumplir las demandas de los usuarios de implementar resultados de analítica de datos en aplicaciones nuevas o existentes.

Usos de R

R for Data Analysts

R – as an interactive "data analysis tool"

- classical statistical tests
- linear and nonlinear modeling
- time-series analysis
- classification
- clustering
- more..

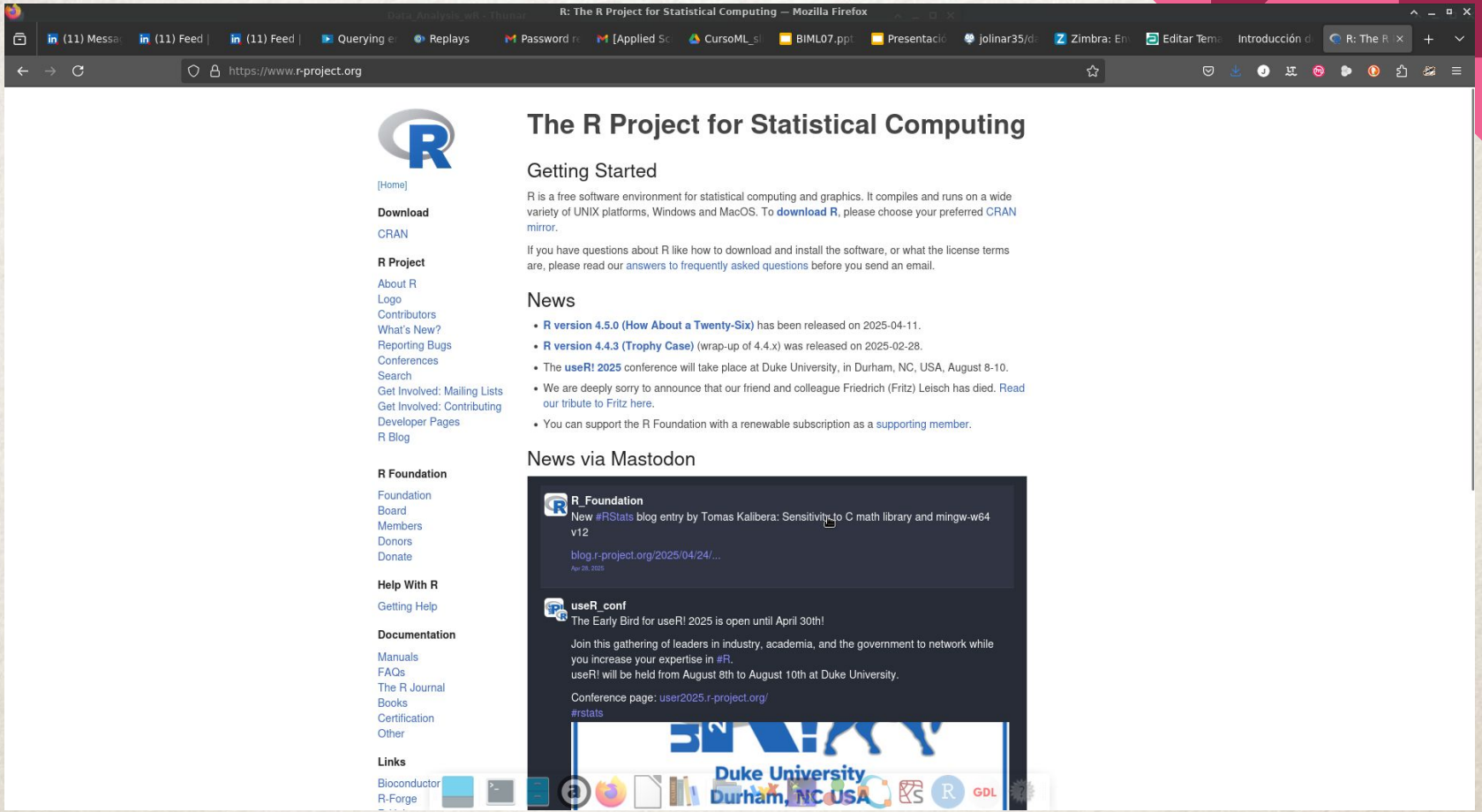
R for Developers

R – as a "programming language"

- Expressions
- Built-in Functions
- Data structures
  - Vectors, Matrices, Lists, Data Frames
- External libraries – packages
- User-defined Functions
- User-defined Classes

Obteniendo R

www.r-project.org



Installing R on Windows

Windows

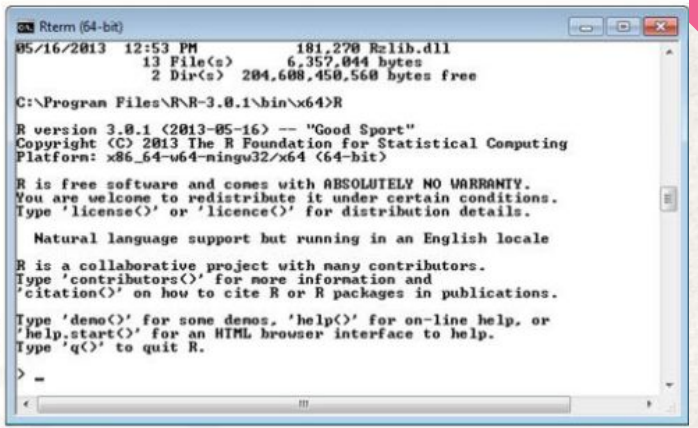
- CRAN mirror site
- Download MSI (32 / 64 bit)
- Install

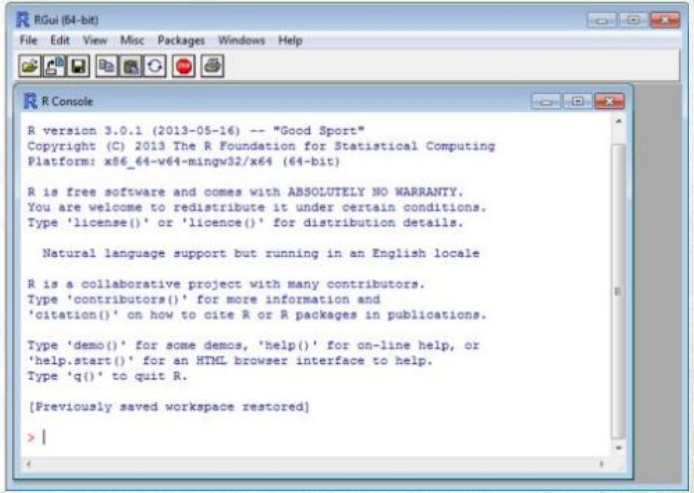
Using R

- Command Line:

```
C:\Program Files\R\R-3.0.1\bin
```

- GUI – Start Menu





Installing R on Linux

Linux

- CRAN mirror site
- Download RPM

```
rpm -ivh <.rpm>
```

- OR

```
yum install R
```

▪ Using R

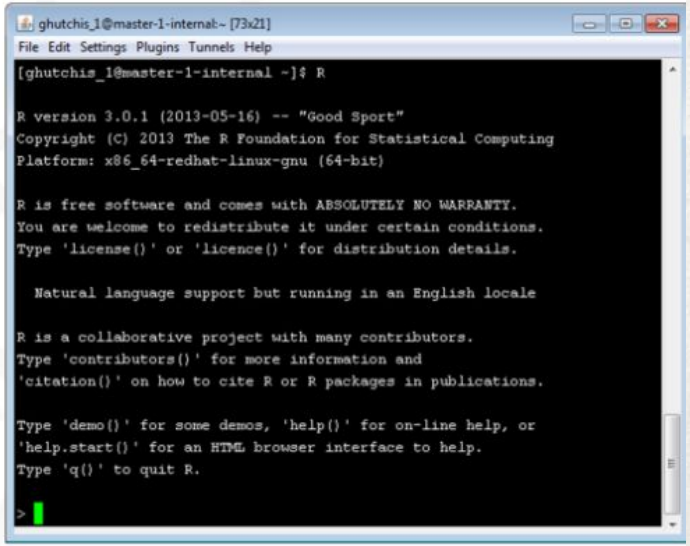
- Command Line:

```
/usr/bin/R
```

- GUI

```
/usr/bin/R -g Tk &
```

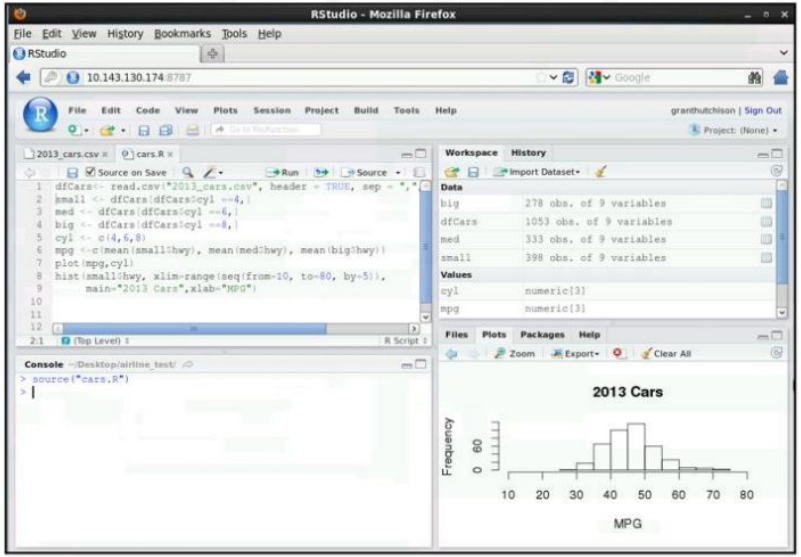
- RCmdr, RStudio (optional – next slide)

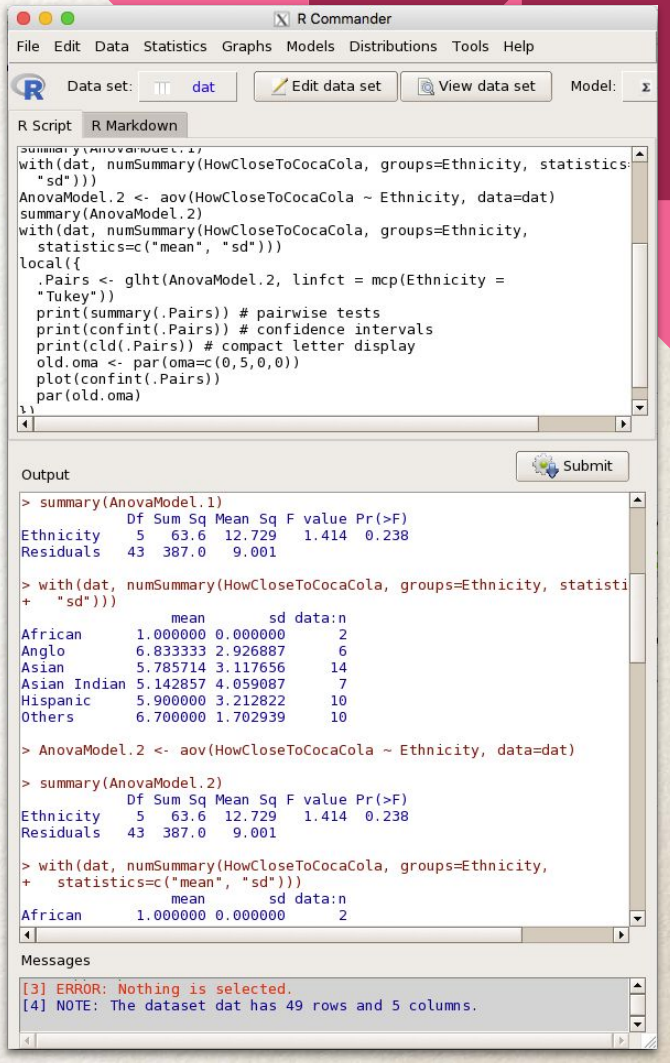


GUIs para R

GUI Alternatives - RStudio

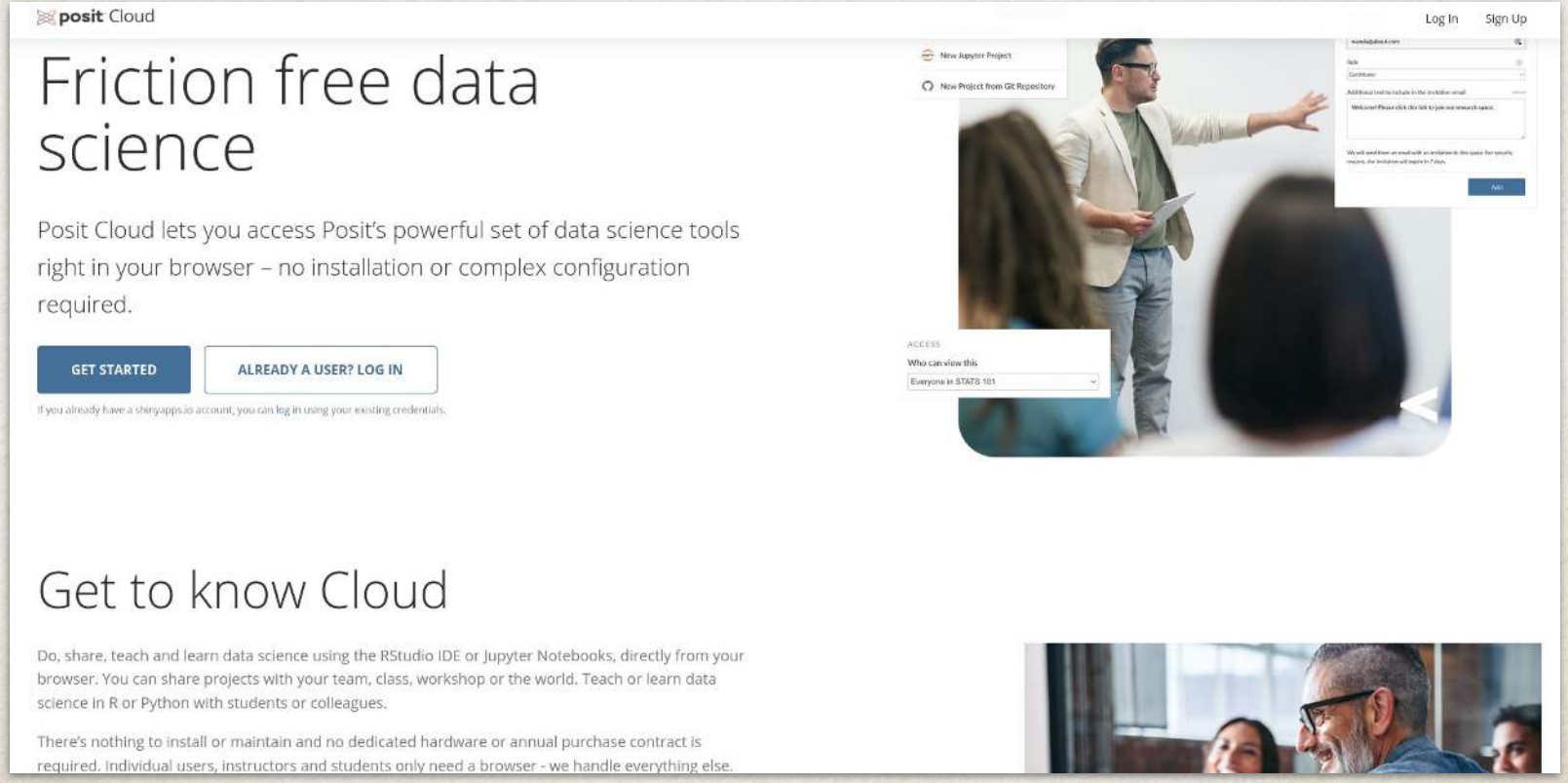
Rstudio is a free IDE that makes it easier to create and test R scripts. (rstudio.org)





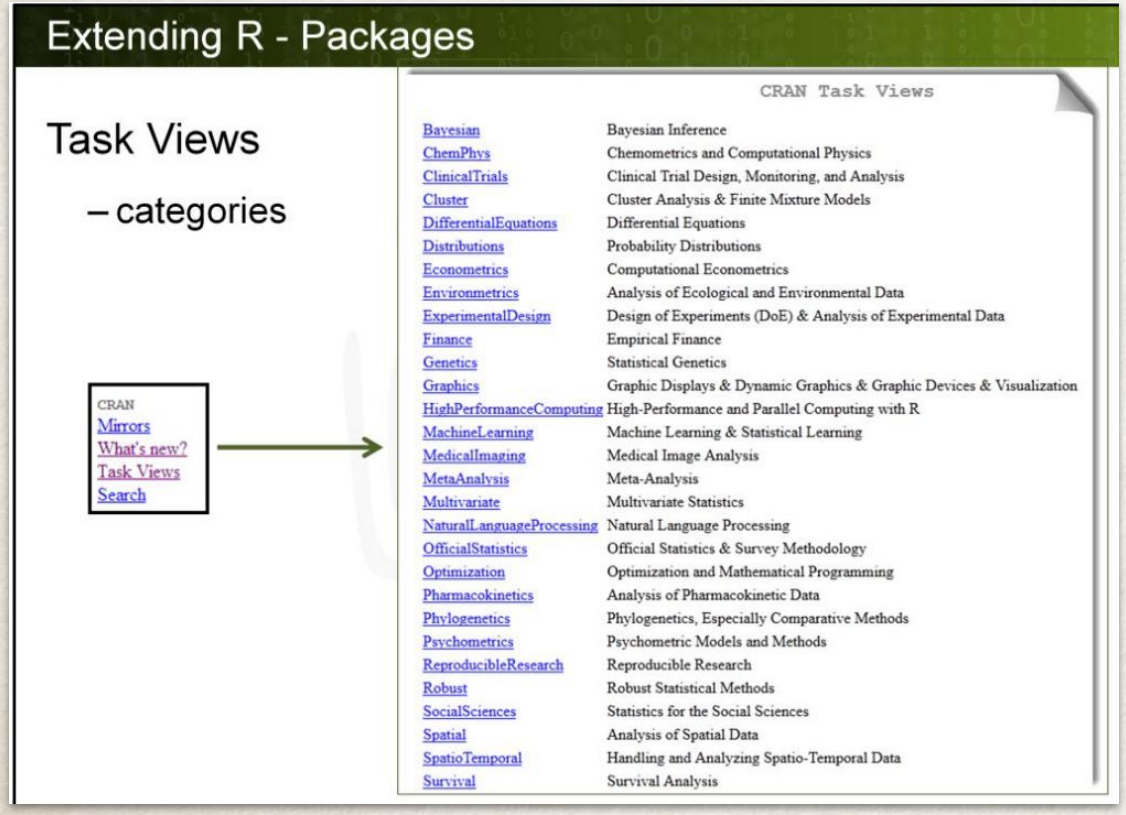
Online IDE para R

https://posit.cloud/



Extendiendo funcionalidades de R

Paquetes



Hay más de 4000 paquetes diferentes disponibles en CRAN y se están agregando más frecuentemente.
Los paquetes publicados en CRAN se clasifican en función de su funcionalidad en Task Views.

Manejando Paquetes en R

R - Packages

Working with addition R packages

- Installing new packages from CRAN

```
>install.packages("RJDBC")
```

- Installing new packages from a downloaded file

```
>install.packages("file.tar.gz")
```

- List the currently loaded packages

```
>library()
```

- Loading a package into a session

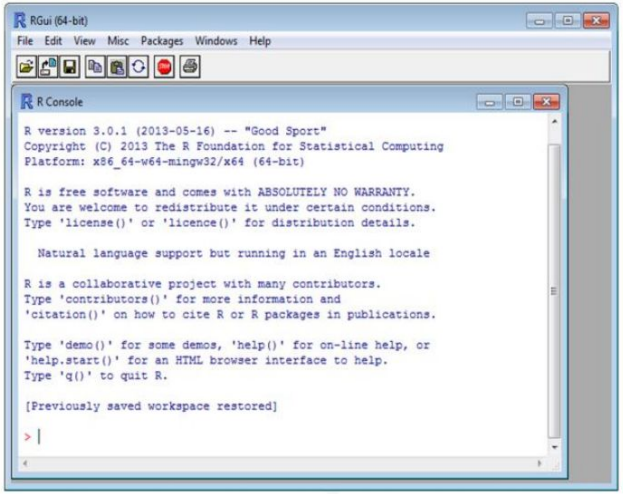
```
>library("stringr")
>require("stringr")
```

Paquetes más comunes de R

- ggplot2 & ggraph
- tidyr & tidyverse
- dplyr
- shiny
- caret
- e1071
- mlr3
- xgboost
- knitr

Trabajando con la Consola de R

R - Console



R Console prompt is the '>' symbol

- If the expression is complete the it is executed
- Otherwise, the prompt changes to the '+" symbol

R - Working Directory

- Each R session maintains a **workspace**
- Workspace - stores the active data structure

  `.RData` file in the workspace directory

- Setting the working directory

```
>setwd("location of directory")
```

- Determining the current working directory

```
>getwd()
```

Variables

Asignación de Variables

```
a <- 23
```

El operador '<-' se usa para asignar valores a una variable

```
> height <- 2

> height
[1] 2

> width <- 4

> width
[1] 4
```

Listando variables y Ejecutando Scripts en R

Listar Variables

```
ls()
```

`'ls'` y `'objects'` devuelven un vector de cadenas de caracteres que dan el nombres de los objetos en el entorno especificado. Cuando se invoca sin ningún argumento, `'ls'` muestra los datos, conjuntos y funciones que un usuario ha definido. Cuando se invoca dentro de una función, `'ls'` devuelve los nombres de los variables locales de las funciones.

Para listar todas las variables

```
ls(all.names = TRUE)
```

R - Batch Mode

Executing R scripts outside the console

– Windows

```
c:\Program Files\R\R-3.0.1\bin\x64\R CMD BATCH makePlots.R
c:\Program Files\R\R-3.0.1\bin\x64\Rscript makePlots.R
```

– Linux

- Execute the following from a Linux prompt

```
$ R CMD BATCH makePlots.R
$ Rscript makePlots.R
```

Output (default)

`makePlots.Rout`

- Add the following to the first line of the `makePlots.R` file

```
#! /usr/bin/Rscript
```

Manejando Variables en R

```
> ls()
[1] "area"   "height" "width"

> rm(area)

> ls()
[1] "height" "width"

> area
Error: object 'area' not found
```

Scripts en R

rectangle.R

```
# Create variables height and width
height <- 3
width <- 6

# Calculate the area
area <- height * width

# Print the area
area

# x <- 3    not executed!
```

- Archivo con extension **.R**
- Lista de comandos R que se ejecutan secuencialmente.
- Usamos '**#**' para comentar una o varias líneas de texto
- Permiten automatizar tareas
- Se ejecutan mediante **Rscript**

```
> Rscript scriptname.R
```

Espacio de Trabajo R

Salvando y Cargando Variables

```
data_1 <- c(4, 1, 8, 10, 15)          # Create simple example data
data_2 <- 5                           # Create another data object
data_3 <- "Hello R User"              # Create a third data object
```

```
# Save whole workspace to working directory
save.image("C:/ ... Your Path ... /all_data.RData")
```

```
# Load workspace back to RStudio
load("C:/ ... Your Path ... /all_data.RData")
```

```
# Save multiple data objects to working directory
save(data_1, data_2,
     file = "C:/ ... Your Path ... /data_1_and_2.RData")
```

Example 3: Save & Load a Single Data Object (saveRDS Function)

We can even save only a single data object to our PC. Either, we can use the code of Example 1 and specify only a single argument within the save command; Or we can use the saveRDS function, which provides further flexibility when saving single data objects:

```
# Save single data object to working directory
saveRDS(data_1,
        file = "C:/ ... Your Path ... /single_data_object.RData")
```

saveRDS also creates an RData file in the currently used folder on your computer. However, in order to reload RData filed created by saveRDS it is advantageous to use the corresponding readRDS function in order to read the data back to R:

```
# Load workspace back to RStudio
data_1_reloaded <- readRDS("C:/ ... Your Path ... /single_data_object.RData")
data_1_reloaded
# 4  1  8 10 15
```

As you can see based on the previous R code, the readRDS package allows to rename a data object during the data import (in our case we used the new name data_1_reloaded).

Tipos de Datos y Operadores

Basic Data Types	Values	Examples
Numeric	Set of all real numbers	"numeric_value <- 3.14"
Integer	Set of all integers, Z	"integer_value <- 42L"
Logical	TRUE and FALSE	"logical_value <- TRUE"
Complex	Set of complex numbers	"complex_value <- 1 + 2i"
Character	"a", "b", "c", ..., "@", "#", "$", ..., "1", "2", ...etc	"character_value <- "Hello Geeks"
raw	as.raw()	"single_raw <- as.raw(255)"

```
> class(2)
[1] "numeric"

> class(2L)
[1] "integer"
```

```
> is.numeric(2)
[1] TRUE

> is.numeric(2L)
[1] TRUE

> is.integer(2)
[1] FALSE

> is.integer(2L)
[1] TRUE
```

integer **is** numeric
numeric **not always** integer

Coerción de Tipos

Convertir un tipo de datos en otro

```
> as.numeric(TRUE)
[1] 1
> as.numeric(FALSE)
[1] 0
> as.character(4)
[1] "4"
> as.numeric("4.5")
[1] 4.5

> as.integer("4.5")
[1] 4
> as.numeric("Hello")
[1] NA
Warning message:
NAs introduced by coercion
```

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
^	Exponent	x ^ y
%%	Modulus (Remainder from division)	x %% y
%/%	Integer Division	x%/%y

Asignacion

```
my_var <- 3

my_var <<- 3

3 -> my_var

3 ->> my_var

my_var # print my_var
```

R Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description
&	Element-wise Logical AND operator. Returns TRUE if both elements are TRUE
&&	Logical AND operator - Returns TRUE if both statements are TRUE
|	Elementwise- Logical OR operator. Returns TRUE if one of the statements is TRUE
||	Logical OR operator. Returns TRUE if one of the statements is TRUE
!	Logical NOT - Returns FALSE if statement is TRUE

Operator	Description
:	Creates a series of numbers in a sequence
%in%	Find out if an element belongs to a vector
%*%	Matrix Multiplication

Estructuras de Datos en R

Vectores y Matrices

Vector: Tupla de elementos del mismo tipo (numérico, lógico, caracteres)
Se crean con el constructor `c()`

```
> c("hearts", "spades", "diamonds", "diamonds", "spades")
[1] "hearts"   "spades"   "diamonds" "diamonds" "spades"

> drawn_suits <- c("hearts", "spades", "diamonds",
                   "diamonds", "spades")

> drawn_suits
[1] "hearts"   "spades"   "diamonds" "diamonds" "spades"

> is.vector(drawn_suits)
[1] TRUE
```

Vectores

Nominar elementos

Podemos nominar los elementos de un vector (etiquetarlos)

option 1

```
> remain <- c(11, 12, 11, 13)
> suits <- c("spades", "hearts", "diamonds", "clubs")
> names(remain) <- suits
```

option 2

```
> remain <- c(spades = 11, hearts = 12,
              diamonds = 11, clubs = 13)
```

option 3

```
> remain <- c("spades" = 11, "hearts" = 12,
              "diamonds" = 11, "clubs" = 13)
```

```
> str(remain)
 Named num [1:4] 11 12 11 13
 - attr(*, "names")= chr [1:4] "spades" "hearts"
                                "diamonds" "clubs"
```

Más Ejemplos

Single value = vector

```
> my_apples <- 5
> my_oranges <- "six"

> is.vector(my_apples)
[1] TRUE
> is.vector(my_oranges)
[1] TRUE

> length(my_apples)
[1] 1
> length(my_oranges)
[1] 1

> length(drawn_suits)
[1] 5
```

Los vectores son homogéneos

- Solo elementos del mismo tipo
- Vectores atómicos <> listas
- Coersión automática si es necesario

```
> drawn_ranks <- c(7, 4, "A", 10, "K", 3, 2, "Q")

> drawn_ranks
[1] "7"  "4"  "A"  "10" "K"  "3"  "2"  "Q"

> class(drawn_ranks)
[1] "character"
```

Aritmética de Vectores

```
> my_apples <- 5
> my_oranges <- 6
> my_apples + my_oranges
[1] 11
```

`my_apples` **is a vector!**
`my_oranges` **is a vector!**

Las operaciones se realizan por elemento

```
> earnings <- c(50, 100, 30)

> earnings * 3
[1] 150 300 90
```

```
> earnings/10
[1]  5 10  3

> earnings - 20
[1] 30 80 10

> earnings + 100
[1] 150 200 130

> earnings^2
[1]  2500 10000   900
```

Operaciones “per elemento”

```
> earnings <- c(50, 100, 30)
> expenses <- c(30, 40, 80)

> earnings - expenses
[1]  20  60 -50

> earnings + c(10, 20, 30)
[1]  60 120  60

> earnings * c(1, 2, 3)
[1]  50 200  90

> earnings / c(1, 2, 3)
[1] 50 50 10
```

multiplication and division are done element-wise!

```
> earnings <- c(50, 100, 30)
> expenses <- c(30, 40, 80)

> bank <- earnings - expenses
> bank
[1]  20  60 -50

> sum(bank)
[1] 30

> earnings > expenses
[1]  TRUE  TRUE FALSE
```

Segmentación (Subsetting) de Vectores

Selección por índices

Importante: los índices comienzan en 1!!

```
> remain <- c(spades = 11, hearts = 12,
              diamonds = 11, clubs = 13)

> remain[1]
spades
    11

> remain[3]
diamonds
      11
```

`[1]` -> **take element at index 1**

result is a (named) vector too!

Segmentación (Subsetting) de Vectores

Selección por nombres

```
> remain <- c(spades = 11, hearts = 12,
              diamonds = 11, clubs = 13)

> remain["spades"]
spades
    11

> remain["diamonds"]
diamonds
      11
```

Segmentación (Subsetting) de Vectores

Selección de múltiples elementos

```
> remain <- c(spades = 11, hearts = 12,
              diamonds = 11, clubs = 13)

> remain_black <- remain[c(1, 4)]
> remain_black
spades  clubs
    11     13

> remain[c(4, 1)]
 clubs spades
    13     11

> remain[c("clubs", "spades")]
 clubs spades
    13     11
```

order in selection vector matters!

Segmentación (Subsetting) de Vectores

Todos excepto algunos elementos

```
> remain <- c(spades = 11, hearts = 12,
              diamonds = 11, clubs = 13)

> remain[-1]
  hearts diamonds    clubs
      12       11       13

> remain[-c(1, 2)]
diamonds    clubs
      11       13

> remain[-"spades"]
Error in -"spades" : invalid argument to unary operator
```

All but index 1 are returned

los elementos 1 y 2 no son incluidos

Segmentación (Subsetting) de Vectores

Usando un vector de booleanos

```
> remain <- c(spades = 11, hearts = 12,
              diamonds = 11, clubs = 13)

> remain[c(TRUE, FALSE)]
  spades diamonds
      11       11
> remain[c(TRUE, FALSE, TRUE, FALSE)]
  spades diamonds
      11       11

> remain[c(TRUE, FALSE, TRUE)]
  spades diamonds    clubs
      11       11       13
> remain[c(TRUE, FALSE, TRUE, TRUE)]
  spades diamonds    clubs
      11       11       13
```

R recycles `c(T, F)` to `c(T, F, T, F)`

Definiendo Matrices

Create a matrix `matrix()`

```
> matrix(1:6, nrow = 2)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> matrix(1:6, ncol = 3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> matrix(1:6, nrow = 2, byrow = TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Reciclando índices

Broadcasting

Create a matrix: recycling

```
> matrix(1:3, nrow = 2, ncol = 3)
     [,1] [,2] [,3]
[1,]    1    3    2
[2,]    2    1    3

> matrix(1:4, nrow = 2, ncol = 3)
     [,1] [,2] [,3]
[1,]    1    3    1
[2,]    2    4    2
Warning message:
In matrix(1:4, nrow = 2, ncol = 3) :
  data length [4] is not a sub-multiple or multiple of the
                                    number of columns [3]
```

Definición por columnas o filas

cbind() & rbind()

```
> cbind(1:3, 1:3)
     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    3    3

> rbind(1:3, 1:3)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    3
```

```
> m <- matrix(1:6, byrow = TRUE, nrow = 2)

> rbind(m, 7:9)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9

> cbind(m, c(10, 11))
     [,1] [,2] [,3] [,4]
[1,]    1    2    3   10
[2,]    4    5    6   11
```

Nominado columnas o filas

rownames() & colnames()

```
> m <- matrix(1:6, byrow = TRUE, nrow = 2)

> rbind(m, 7:9)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9

> cbind(m, c(10, 11))
     [,1] [,2] [,3] [,4]
[1,]    1    2    3   10
[2,]    4    5    6   11
```

```
> m <- matrix(1:6, byrow = TRUE, nrow = 2,
              dimnames = list(c("row1", "row2"),
                              c("col1", "col2", "col3")))

> m
     col1 col2 col3
row1    1    2    3
row2    4    5    6
```

Naming a matrix `rownames()`, `colnames()`

```
> m <- matrix(1:6, byrow = TRUE, nrow = 2)

> rownames(m) <- c("row1", "row2")

> m
     [,1] [,2] [,3]
row1    1    2    3
row2    4    5    6

> colnames(m) <- c("col1", "col2", "col3")

> m
     col1 col2 col3
row1    1    2    3
row2    4    5    6
```

Coerción de Tipos

```
> num <- matrix(1:8, ncol = 2)
> char <- matrix(LETTERS[1:6], nrow = 4, ncol = 3)

> cbind(num, char)
     [,1] [,2] [,3] [,4] [,5]
[1,] "1"  "5"  "A"  "E"  "C"
[2,] "2"  "6"  "B"  "F"  "D"
[3,] "3"  "7"  "C"  "A"  "E"
[4,] "4"  "8"  "D"  "B"  "F"
```

Segmentación (Subsetting) de Matrices

Subset column or row

```
> m[3,]
[1] 6 1 4 2

> m[,3]
[1] 15  8  4

> m[4]
[1] 11

> m[9]
[1] 4
```

```
> m
     [,1] [,2] [,3] [,4]
[1,]    5   11   15    3
[2,]   12   14    8    9
[3,]    6    1    4    2
```

Segmentación (Subsetting) de Matrices

Subset multiple elements

```
> m[2, c(2, 3)]
[1] 14  8

> m[c(1, 2), c(2, 3)]
     [,1] [,2]
[1,]   11   15
[2,]   14    8

> m[c(1, 3), c(1, 3, 4)]
     [,1] [,2] [,3]
[1,]    5   15    3
[2,]    6    4    2
```

```
> m
     [,1] [,2] [,3] [,4]
[1,]    5   11   15    3
[2,]   12   14    8    9
[3,]    6    1    4    2
```

Segmentación (Subsetting) de Matrices

Subset by name

```
> rownames(m) <- c("r1", "r2", "r3")
> colnames(m) <- c("a", "b", "c", "d")
> m
    a  b  c d
r1  5 11 15 3
r2 12 14  8 9
r3  6  1  4 2
```

```
> m[2,3]
[1] 8

> m["r2","c"]
[1] 8
```

```
> m[2,"c"]
[1] 8

> m[3, c("c", "d")]
c d
4 2
```

Segmentación (Subsetting) de Matrices

Subset with logical vector

```
> m[c(FALSE, FALSE, TRUE),
    c(FALSE, FALSE, TRUE, TRUE)]
c d
4 2

> m[c(FALSE, FALSE, TRUE),
    c(FALSE, TRUE)]
b d
1 2

> m[c(FALSE, FALSE, TRUE),
    c(FALSE, TRUE, FALSE, TRUE)]
b d
1 2
```

```
> m
    a  b  c d
r1  5 11 15 3
r2 12 14  8 9
r3  6  1  4 2
```

Estructuras de Datos en R

Factors

Factors: estructura especial para encapsular variables categóricas

`factor()`

```
> blood <- c("B", "AB", "O", "A", "O", "O", "A", "B")
> blood
[1] "B"  "AB" "O"  "A"  "O"  "O"  "A"  "B"

> blood_factor <- factor(blood)
> blood_factor
[1] B  AB O  A  O  O  A  B
Levels: A AB B O

> str(blood_factor)
 Factor w/ 4 levels "A","AB","B","O": 3 2 4 1 4 4 1 3
```

R sorts levels alphabetically!

Cada uno de los posibles valores de la variable se denomina nivel (i.e. level)

Factors

Reordenando Niveles

```
> blood_factor2 <- factor(blood,
                          levels = c("O", "A", "B", "AB"))

> blood_factor2
[1] B  AB O  A  O  O  A  B
Levels: O A B AB

> str(blood_factor2)
 Factor w/ 4 levels "O","A","B","AB": 3 4 1 2 1 1 2 3

> str(blood_factor)
 Factor w/ 4 levels "A","AB","B","O": 3 2 4 1 4 4 1 3
```

si predefinimos los niveles de antemano no aparecen ordenados por defecto

Factors

Renombrando Niveles

Podemos renombrar los niveles de un factor de manera directa

```
> blood <- c("B", "AB", "O", "A", "O", "O", "A", "B")
> blood_factor <- factor(blood)

> levels(blood_factor) <- c("BT_A", "BT_AB", "BT_B", "BT_O")
> blood_factor
[1] BT_B  BT_AB BT_O  BT_A  BT_O  BT_O  BT_A  BT_B
Levels: BT_A BT_AB BT_B BT_O

> factor(blood, labels = c("BT_A", "BT_AB", "BT_B", "BT_O"))
[1] BT_B  BT_AB BT_O  BT_A  BT_O  BT_O  BT_A  BT_B
Levels: BT_A BT_AB BT_B BT_O
```

Factors

Renombrando Niveles

```
> blood <- c("B", "AB", "O", "A", "O", "O", "A", "B")
> blood_factor <- factor(blood)


> factor(blood,
         levels = c("O", "A", "B", "AB"),
         labels = c("BT_O", "BT_A", "BT_B", "BT_AB"))
[1] BT_B  BT_AB BT_O  BT_A  BT_O  BT_O  BT_A  BT_B
Levels: BT_O BT_A BT_B BT_AB
```

Factores Ordenados

Ciertas variables ordinales conllevan un ordenamiento implícito.
p. ej: tallas, posiciones en un ranking, etc. (variables ordinales)

```
> blood <- c("B", "AB", "O", "A", "O", "O", "A", "B")
> blood_factor <- factor(blood)
> blood_factor[1] < blood_factor[2]
[1] NA
Warning message:
In Ops.factor(blood_factor[1], blood_factor[2]) :
  '<' not meaningful for factors

> tshirt <- c("M", "L", "S", "S", "L", "M", "L", "M")

> tshirt_factor <- factor(tshirt, ordered = TRUE,
                          levels = c("S", "M", "L"))
> tshirt_factor
[1] M L S S L M L M
Levels: S < M < L
```

Sintaxis

```
factor(x = character(),          # Vector de entrada
       levels,                   # Valores únicos de x (opcional)
       labels = levels,          # Etiquetas para los niveles (opcional)
       exclude = NA,             # Valores a excluir de los niveles
       ordered = is.ordered(x),  # Si los niveles de entrada están ordenados o
       nmax = NA)                # Máximo número de niveles
```

Factores Ordenados

```
> tshirt <- c("M", "L", "S", "S", "L", "M", "L", "M")

> tshirt_factor <- factor(tshirt, ordered = TRUE,
                          levels = c("S", "M", "L"))

> tshirt_factor
[1] M L S S L M L M
Levels: S < M < L

> tshirt_factor[1] < tshirt_factor[2]
[1] TRUE
```

la comparación entre elementos anterior no era posible porque **no existía** un ordenamiento definido en el factor.

Estructuras de Datos en R

Listas

- Las listas permiten incluir elementos de diferente tipo
- Podemos etiquetar sus elementos
- No hay coercion de tipos

Create list `list()`

```
> c("Rsome times", 190, 5)
[1] "Rsome times"   "190"    "5"

> list("Rsome times", 190, 5)
[[1]]
[1] "Rsome times"

[[2]]
[1] 190

[[3]]
[1] 5

> song <- list("Rsome times", 190, 5)
> is.list(song)
[1] TRUE
```

Name list

```
> song <- list("Rsome times", 190, 5)

> names(song) <- c("title", "duration", "track")

> song
$title
[1] "Rsome times"

$duration
[1] 190

$track
[1] 5
```

Listas Anidadas

```
> song <- list(title = "Rsome times",
               duration = 190,
               track = 5)

> str(song)
List of 3
 $ title   : chr "Rsome times"
 $ duration: num 190
 $ track   : num 5
```

Podemos incluir una lista dentro de otra lista

```
> similar_song <- list(title = "R you on time?",
                       duration = 230)

> song <- list(title = "Rsome times",
               duration = 190, track = 5,
               similar = similar_song)

> str(song)
List of 4
 $ title   : chr "Rsome times"
 $ duration: num 190
 $ track   : num 5
 $ similar :List of 2
  ..$ title   : chr "R you on time?"
  ..$ duration: num 230
```

Segmentación (Subsetting) de Listas

```
> similar_song <- list(title = "R you on time?",
                       duration = 230)
> song <- list(title = "Rsome times",
               duration = 190, track = 5,
               similar = similar_song)

> song
List of 4
 $ title   : chr "Rsome times"
 $ duration: num 190
 $ track   : num 5
 $ similar :List of 2
  ..$ title   : chr "R you on time?"
  ..$ duration: num 230
```

Output of calling `str(song)`

Subsetting e Indexación de Elementos en listas Anidadas

[versus [[

```
> song[1]
List of 1
 $ title: chr "Rsome times"

> song[[1]]
[1] "Rsome times"

> song[c(1, 3)]
List of 2
 $ title: chr "Rsome times"
 $ track: num 5
```

```
> song
List of 4
 $ title   : chr "Rsome times"
 $ duration: num 190
 $ track   : num 5
 $ similar :List of 2
  ..$ title   : chr "R you on time?"
  ..$ duration: num 230
```

Usamos '[' o '$' para acceder/obtener una sublista y '[[' para obtener un elemento especifico

$ and extending

```
> song$duration
[1] 190

> friends <- c("Kurt", "Florence",
               "Patti", "Dave")
> song$sent <- friends
> song
List of 5
 $ title   : chr "Rsome times"
 $ duration: num 190
 $ track   : num 5
 $ similar :List of 2
  ..$ title   : chr "R you on time?"
  ..$ duration: num 230
 $ sent    : chr [1:4] "Kurt" "Florence" "Patti" "Dave"
```

```
> song
List of 4
 $ title   : chr "Rsome times"
 $ duration: num 190
 $ track   : num 5
 $ similar :List of 2
  ..$ title   : chr "R you on time?"
  ..$ duration: num 230
```

Podemos extender la lista

```
> song[["sent"]] <- friends

> song$similar$reason <- "too long"

> song
List of 5
 $ title   : chr "Rsome times"
 $ duration: num 190
 $ track   : num 5
 $ similar :List of 3
  ..$ title   : chr "R you on time?"
  ..$ duration: num 230
  ..$ reason  : chr "too long"
 $ sent    : chr [1:4] "Kurt" "Florence" "Patti" "Dave"
```

Uso de Vectores lógicos

Subset by logicals

```
> song[c(FALSE, TRUE, TRUE, FALSE)]
List of 2
 $ duration: num 190
 $ track   : num 5

> song[[c(FALSE, TRUE, TRUE, FALSE)]]
Error : attempt to select less than one element

> song[[F]][[T]][[T]][[F]]
Error : attempt to select less than one element
```

```
> song
List of 4
 $ title   : chr "Rsome times"
 $ duration: num 190
 $ track   : num 5
 $ similar :List of 2
  ..$ title   : chr "R you on time?"
  ..$ duration: num 230
```

Segmentación por Nombres

Subset by names

```
> song[["duration"]]
[1] 190

> song["duration"]
List of 1
 $ duration: num 190

> song[c("duration", "similar")]
List of 2
 $ duration: num 190
 $ similar :List of 2
  ..$ title    : chr "R you on time?"
  ..$ duration: num 230
```

```
> song
List of 4
 $ title    : chr "Rsome times"
 $ duration: num 190
 $ track    : num 5
 $ similar :List of 2
  ..$ title    : chr "R you on time?"
  ..$ duration: num 230
```

Todas las operaciones de segmentación de vectores y matrices se pueden aplicar a listas

Estructuras de Datos en R

Dataframes

Generalidades sobre los conjuntos de Datos (Datasets)

- En general contienen datos de diferente tipos
- Observaciones sobre diferentes variables
- Marcas de Tiempo (timestamps)
- Valores perdidos (NA/NaN/NULL/etc)
- Variables categoricas y cuantitativas
- Metadata
- Etc

Que estructura usar?

- ¿Vectores y Matrices?: no adecuados
- ¿Factores y Listas?: muy limitados

Solución:

Dataframes

- Tablas extendidas, tipo hoja de calculo.
- Toleran diferentes formatos y multitud de operaciones sobre filas y columnas

Dataframes

Específicamente concebidos para manejar conjuntos de datos

Paradigma Tidy Data

- Filas = observaciones (personas)
- Columnas = variables (age, name, etc.)
- Contienen elementos de diferentes tipos
- Elementos en la misma columna: mismo tipo

Se pueden crear con **`data.frame()`**
Importar desde diferentes fuentes de datos (e.g. .csv, .xls, SQL queries, etc)

```
> name <- c("Anne", "Pete", "Frank", "Julia", "Cath")
> age <- c(28, 30, 21, 39, 35)
> child <- c(FALSE, TRUE, TRUE, FALSE, TRUE)

> df <- data.frame(name, age, child)
```

column names match variable names

```
> df
   name age child
1  Anne  28 FALSE
2  Pete  30  TRUE
3 Frank  21  TRUE
4 Julia  39 FALSE
5  Cath  35  TRUE
```

name	age	child
Anne	28	FALSE
Pete	30	TRUE
Frank	21	TRUE
Julia	39	FALSE
Cath	35	TRUE

Definiendo & Renombrando columnas

```
> names(df) <- c("Name", "Age", "Child")
> df
   Name Age Child
1  Anne  28 FALSE
2  Pete  30  TRUE
   ...
5  Cath  35  TRUE

> df <- data.frame(Name = name, Age = age, Child = child)
> df
   Name Age Child
1  Anne  28 FALSE
2  Pete  30  TRUE
   ...
5  Cath  35  TRUE
```

Dataframes: Consultando la Estructura

Factor instead of character

```
> str(df)
'data.frame':5 obs. of  3 variables:
 $ Name : Factor w/ 5 levels "Anne","Cath",..: 1 5 3 4 2
 $ Age  : num  28 30 21 39 35
 $ Child: logi  FALSE TRUE TRUE FALSE TRUE

> data.frame(name[-1], age, child)
Error : arguments imply differing number of rows: 4, 5

> df <- data.frame(name, age, child,
                   stringsAsFactors = FALSE)
> str(df)
'data.frame':5 obs. of  3 variables:
 $ name : chr  "Anne" "Pete" "Frank" "Julia" ...
 $ age  : num  28 30 21 39 35
 $ child: logi  FALSE TRUE TRUE FALSE TRUE
```

Segmentando y Extendiendo Dataframes

Aplican todos los métodos de vectores, matrices y listas

```
> name <- c("Anne", "Pete", "Frank", "Julia", "Cath")
> age <- c(28, 30, 21, 39, 35)
> child <- c(FALSE, TRUE, TRUE, FALSE, TRUE)
> people <- data.frame(name, age, child,
                       stringsAsFactors = FALSE)

> people
   name age child
1  Anne  28 FALSE
2  Pete  30  TRUE
3 Frank  21  TRUE
4 Julia  39 FALSE
5  Cath  35  TRUE
```

```
> people
   name age child
1  Anne  28 FALSE
2  Pete  30  TRUE
3 Frank  21  TRUE
4 Julia  39 FALSE
5  Cath  35  TRUE
```

```
> people[3,2]
[1] 21

> people[3,"age"]
[1] 21

> people[3,]
   name age child
3 Frank  21  TRUE

> people[,"age"]
[1] 28 30 21 39 35
```

Más ejemplos

```
> people
    name age child
1   Anne  28 FALSE
2   Pete  30  TRUE
3 Frank  21  TRUE
4 Julia  39 FALSE
5   Cath  35  TRUE
```

```
> people[c(3, 5), c("age", "child")]
  age child
3  21  TRUE
5  35  TRUE

> people[2]
  age
1  28
2  30
3  21
4  39
5  35
```

Data Frame ~ List

```
> people$age
[1] 28 30 21 39 35

> people[["age"]]
[1] 28 30 21 39 35

> people[[2]]
[1] 28 30 21 39 35
```

Dataframes: Agregar columnas

Varios métodos

```
> height <- c(163, 177, 163, 162, 157)

> people$height <- height

> people[["height"]] <- height

> people
   name age child height
1  Anne  28 FALSE    163
2  Pete  30  TRUE    177
3 Frank  21  TRUE    163
4 Julia  39 FALSE    162
5  Cath  35  TRUE    157
```

```
> weight <- c(74, 63, 68, 55, 56)

> cbind(people, weight)
   name age child height weight
1  Anne  28 FALSE    163     74
2  Pete  30  TRUE    177     63
3 Frank  21  TRUE    163     68
4 Julia  39 FALSE    162     55
5  Cath  35  TRUE    157     56
```

Dataframes: Agregar Filas

```
> tom <- data.frame("Tom", 37, FALSE, 183)

> rbind(people, tom)
Error : names do not match previous names

> tom <- data.frame(name = "Tom", age = 37,
                    child = FALSE, height = 183)
> rbind(people, tom)
   name age child height
1  Anne  28 FALSE    163
2  Pete  30  TRUE    177
3 Frank  21  TRUE    163
4 Julia  39 FALSE    162
5  Cath  35  TRUE    157
6   Tom  37 FALSE    183
```

Dataframes: Ordenamiento

Sorting

```
> sort(people$age)
[1] 21 28 30 35 39

> ranks <- order(people$age)
> ranks
[1] 3 1 2 5 4

> people$age
[1] 28 30 21 39 35
```

`21` **is lowest: its index,** `3`**, comes first in** `ranks`

`28` **is second lowest: its index,** `1`**, comes second in** `ranks`

`39` **is highest: its index,** `4`**, comes last in** `ranks`

```
> people
   name age child height
1  Anne  28 FALSE    163
2  Pete  30  TRUE    177
3 Frank  21  TRUE    163
4 Julia  39 FALSE    162
5  Cath  35  TRUE    157
```

Ordenamiento

Sorting

```
> sort(people$age)
[1] 21 28 30 35 39

> ranks <- order(people$age)
> ranks
[1] 3 1 2 5 4

> people[ranks, ]
   name age child height
3 Frank  21  TRUE    163
1  Anne  28 FALSE    163
2  Pete  30  TRUE    177
5  Cath  35  TRUE    157
4 Julia  39 FALSE    162
```

```
> people
   name age child height
1  Anne  28 FALSE    163
2  Pete  30  TRUE    177
3 Frank  21  TRUE    163
4 Julia  39 FALSE    162
5  Cath  35  TRUE    157
```

Ordenamiento

Sorting

```
> sort(people$age)
[1] 21 28 30 35 39

> ranks <- order(people$age)
> ranks
[1] 3 1 2 5 4

> people[order(people$age, decreasing = TRUE), ]
   name age child height
4 Julia  39 FALSE    162
5  Cath  35  TRUE    157
2  Pete  30  TRUE    177
1  Anne  28 FALSE    163
3 Frank  21  TRUE    163
```

```
> people
   name age child height
1  Anne  28 FALSE    163
2  Pete  30  TRUE    177
3 Frank  21  TRUE    163
4 Julia  39 FALSE    162
5  Cath  35  TRUE    157
```

Con R tambien podemos desarrollar modelos de Machine Learning y Redes Neuronales

Sumario

- R es un lenguage de programación de alto nivel muy flexible y extensible
- Existen mas de 4K paquetes de R que extienden sus funcionalidades desde las aplicaciones puramente estadísticas hasta Machine Learning y Webapps
- La sintaxis de R es similar a Matlab y Python con algunas características particulares que hacen mas dificultosa su curva de aprendizaje.
- R y Python son los paquetes de software más usados en Ciencia de Datos e Investigación Científica.
- Esta presentación es una breve introducción al manejo básico de R