

```
library(dplyr)
```

```
rladies_global %>%  
  filter(city == 'Buenos Aires')
```



# Taller R – cero



<https://github.com/pmnatural/R-cero>

@priscilla.minotti

@monicaa (Mónica Alonso)

@fflores (Fabiana Flores)



**1. La Interfase de R Studio y miniguía de supervivencia de R**

**2 . Sintaxis de R base**

**3. Ejercicios**

**[Datos ordenados]**

# 1. La interfase de RStudio



The image shows the RStudio interface with four callout boxes highlighting key features:

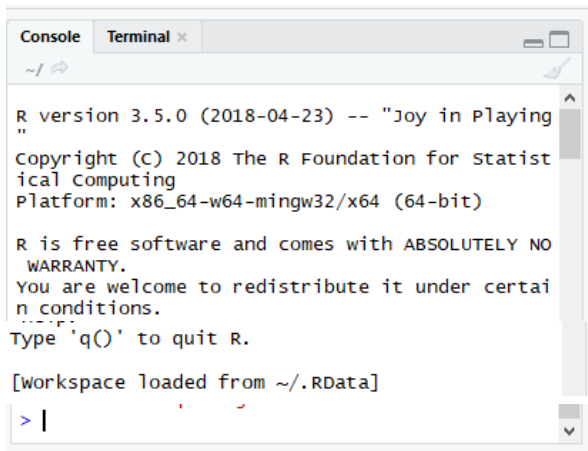
- Source o Fuente** (se usa para hacerlos scripts y también para mostrar datos o documentos): Points to the script editor on the left.
- Entorno e Historia** (Estado de objetos): Points to the Environment and History pane on the right.
- Consola** (funciona en el directorio de trabajo, aquí se ejecutan los comandos): Points to the console at the bottom left.
- Salidas graficas, ayuda, paquetes cargados, archivos del directorio**: Points to the Files, Plots, Packages, Help, and Viewer pane at the bottom right.

The background shows the RStudio window with a script titled "01-Introduction.Rnw" and a plot titled "Biomass estimation per plot with different models".

# La Consola

```
> 1+2+3+10+20+30
[1] 66
```

calculadora



```
X <- 5
X
```

Asignación

```
> x <- 5
> x
[1] 5
```

Autoimpresión



escribir y ejecutar instrucciones (*case sensitive*)

1 ubicación del directorio de trabajo

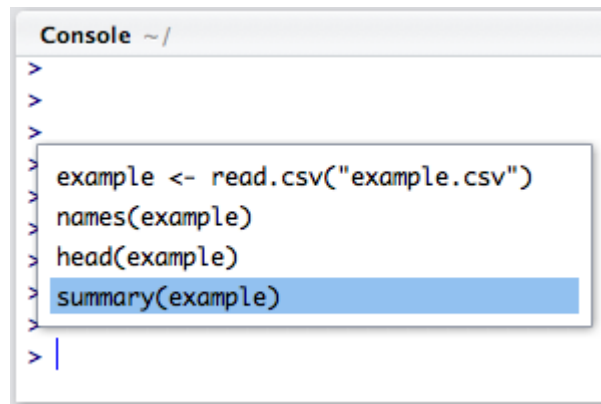
2 interrumpir procesos

# La Consola

## Recuperar comandos anteriores

Con las flechas

- **Arriba** — Recupera comandos anteriores
- **Abajo** — Reversa de Arriba
- **Ctrl+Arriba** -- Recupera la lista de los ultimos comandos y permite recorrerla para seleccionar alguno



The screenshot shows a R console window titled "Console ~/". It displays a list of previously entered commands, with "summary(example)" highlighted in blue. The commands listed are: ">", ">", ">", "example <- read.csv('example.csv')", "names(example)", "head(example)", "summary(example)", and "> |".

```
Console ~/  
>  
>  
>  
example <- read.csv("example.csv")  
names(example)  
head(example)  
summary(example)  
> |
```

Asignación

**Alt -**

## El directorio de trabajo (wd)

Es el ambiente de trabajo actual,  
donde se ejecutan las instrucciones dadas,  
donde se van guardando todos los objetos  
que vas creando en una sesión de trabajo

```
getwd()                                #dónde esta el directorio actual
```

```
> getwd()  
[1] "C:/Users/Admin/Documents"
```

## El directorio de trabajo

```
setwd("path_mi_dir")      #cambia de lugar el ws a mi_dir  
setwd("c:/docs/mi_dir")   #en windows se usa / en vez de \  
setwd("/usr/rob/mi_dir")  #en linux
```

También desde el menú ***Session, Set Working Directory, Choose Directory*** y navegar hasta elegir la carpeta donde instalaremos nuestra nueva área de trabajo

## La Consola - directorio de trabajo

- `ls()` # lista los objetos que creaste en el ws
- `dir()` # lista los archivos del directorio, pero no los objetos

```
rm(nombre_objeto) #remueve el objeto especificado
```

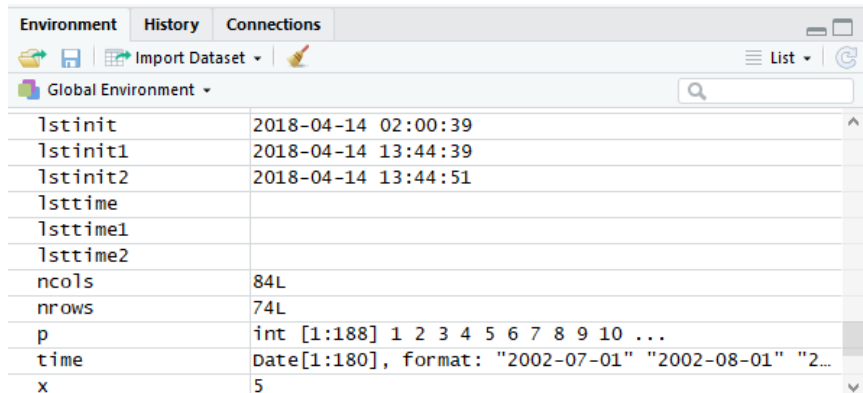
```
rm(list = ls())   #borra todos los objetos de R de tu ws
```



# Panel Entorno

## Environment

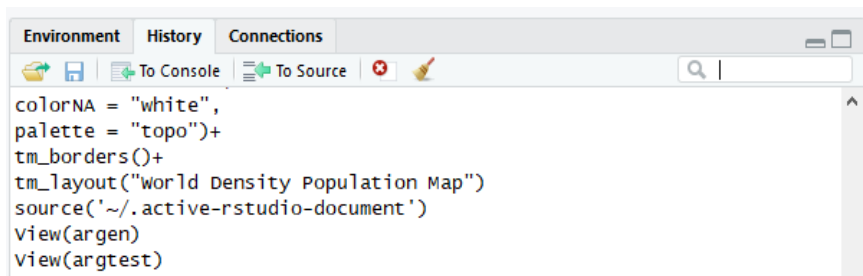
podés ver los objetos creados con un minidetalle de su estructura y rango de valores, importar un dataset



Object	Class	Attributes
l\$init	2018-04-14 02:00:39	
l\$init1	2018-04-14 13:44:39	
l\$init2	2018-04-14 13:44:51	
l\$time		
l\$stime1		
l\$stime2		
ncols	84L	
nrows	74L	
p	int [1:188]	1 2 3 4 5 6 7 8 9 10 ...
time	Date[1:180], format: "2002-07-01" "2002-08-01" "2...	
x	5	

## History

te aparece la lista de todos los comandos ejecutados durante la sesión o proyecto en marcha



```
colorNA = "white",
palette = "topo")+
tm_borders()+
tm_layout("world Density Population Map")
source('~/.active-rstudio-document')
view(argen)
view(argtest)
...
```

## Otros

Connections, Git



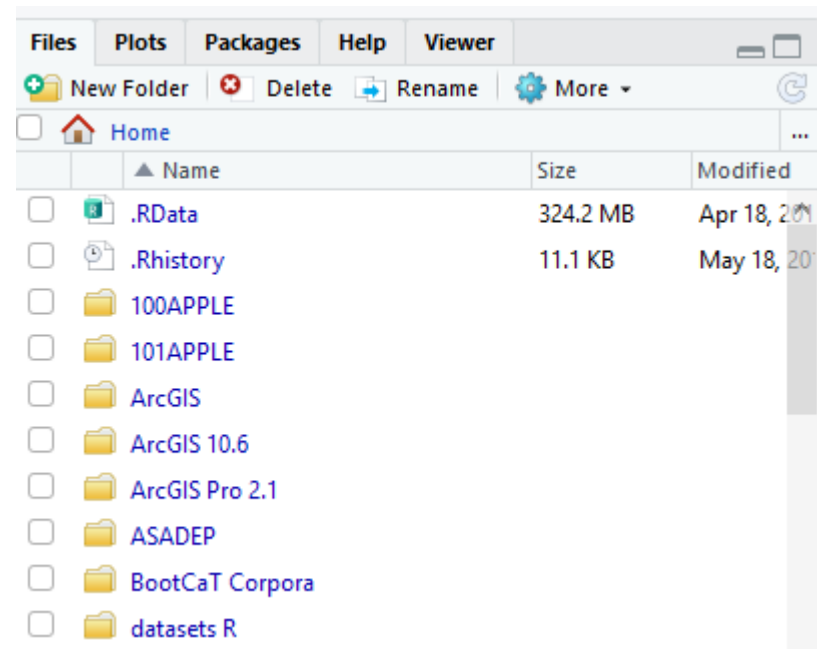
Connection	Status

# Panel Archivos, Paquetes, Ayuda y Otros



## Archivos

La pestaña *Files* nos muestra los archivos guardados en el *wd* y también gestionar archivos o navegar por otros directorios sin tener que salir de RStudio.

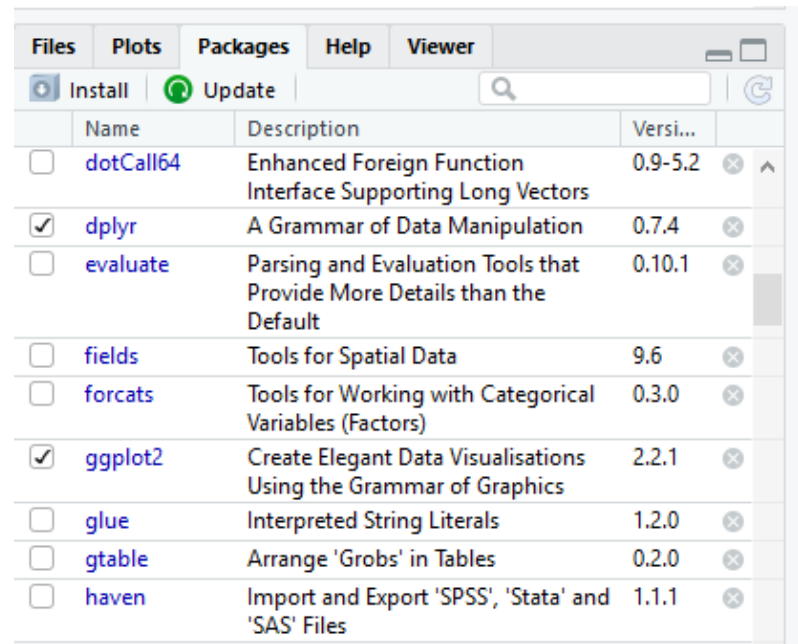


# Paquetes

Los paquetes o *packages* son colecciones de funciones, datos y código compilado de R en un formato definido

El directorio donde se guardan los paquetes se llama *librería*

Hay *System library* y *User library*



The screenshot shows the R Package Manager window with the 'Packages' tab selected. It displays a table of installed and available packages. The 'Install' button is highlighted. The table has columns for Name, Description, and Version. The packages listed are dotCall64, dplyr, evaluate, fields, forcats, ggplot2, glue, gtable, and haven. The 'dplyr' and 'ggplot2' packages are checked, indicating they are installed.

	Name	Description	Versi...
<input type="checkbox"/>	dotCall64	Enhanced Foreign Function Interface Supporting Long Vectors	0.9-5.2
<input checked="" type="checkbox"/>	dplyr	A Grammar of Data Manipulation	0.7.4
<input type="checkbox"/>	evaluate	Parsing and Evaluation Tools that Provide More Details than the Default	0.10.1
<input type="checkbox"/>	fields	Tools for Spatial Data	9.6
<input type="checkbox"/>	forcats	Tools for Working with Categorical Variables (Factors)	0.3.0
<input checked="" type="checkbox"/>	ggplot2	Create Elegant Data Visualisations Using the Grammar of Graphics	2.2.1
<input type="checkbox"/>	glue	Interpreted String Literals	1.2.0
<input type="checkbox"/>	gtable	Arrange 'Grobs' in Tables	0.2.0
<input type="checkbox"/>	haven	Import and Export 'SPSS', 'Stata' and 'SAS' Files	1.1.1

```
install.packages("paquete") # poner el nombre del paquete
```

una sola vez

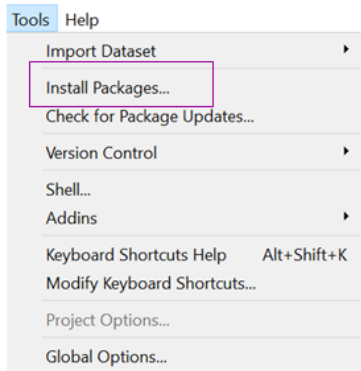
```
library(paquete) # poner el nombre del paquete
```

en cada sesión o proyecto

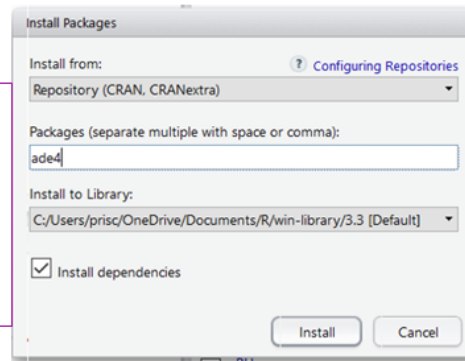


Desde RStudio los paquetes se pueden instalar desde el menú **Tools**

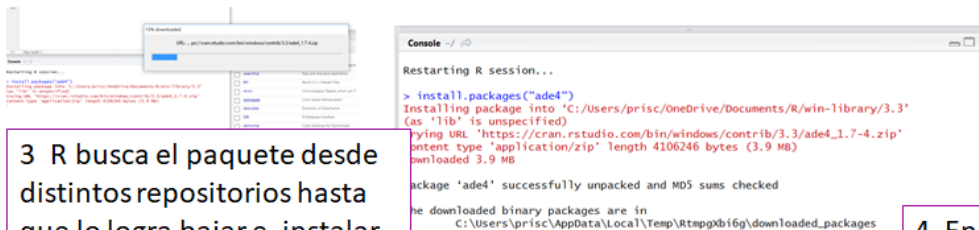
1 Seleccionar  
Install Packages



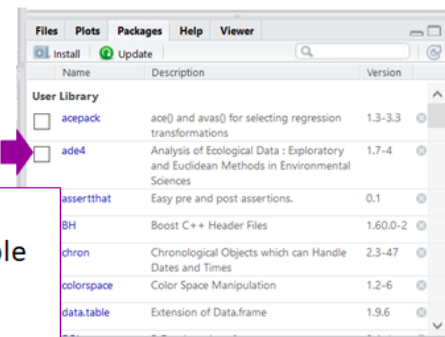
2 Empezar a  
escribir nombre de  
paquete buscado y  
luego  
seleccionarlo de la  
lista desplegable



3 R busca el paquete desde  
distintos repositorios hasta  
que lo logra bajar e instalar,  
e informa además donde  
guardo la descarga



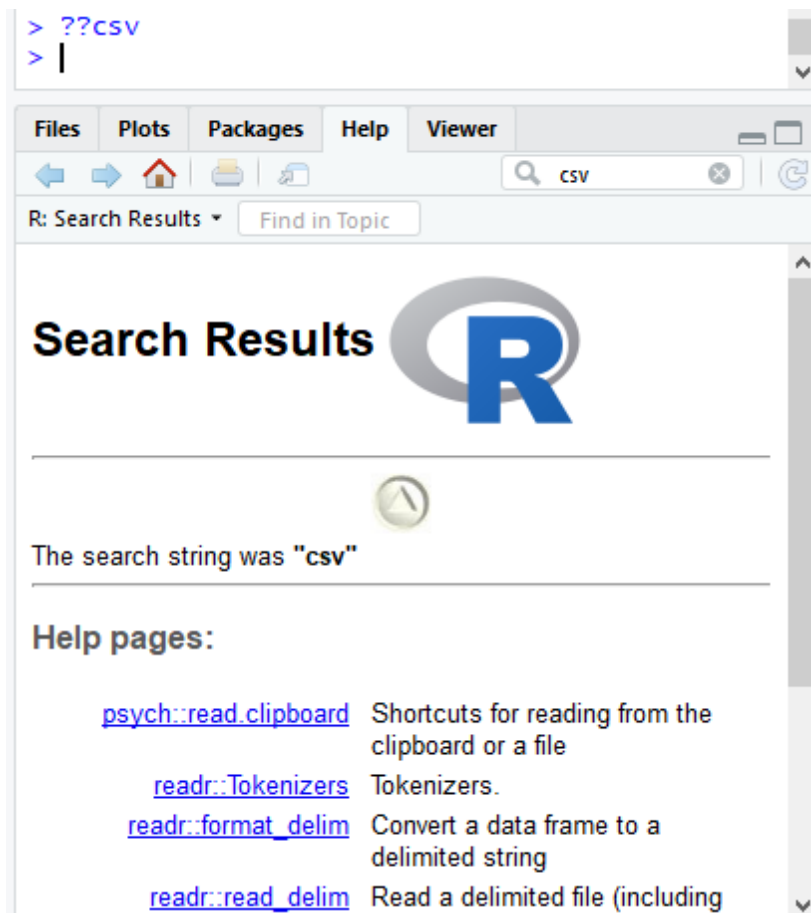
4 En Rstudio se  
muestra disponible  
en la pestaña de  
paquetes



# Ayuda

La pestaña *Help* permite hacer búsquedas de términos y despliega la ayuda

```
help.start()    # ayuda general
help(foo)       # ayuda sobre la función *foo*
?foo           # ayuda sobre la función *foo*
```



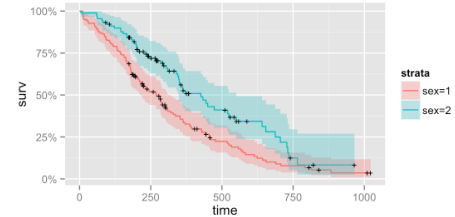
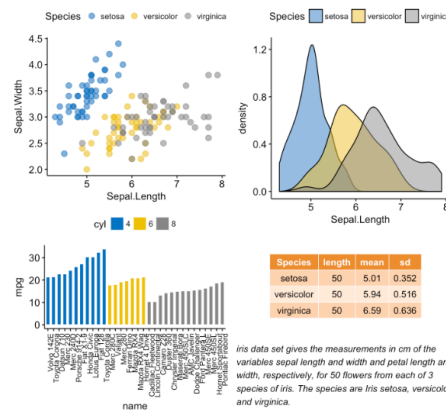
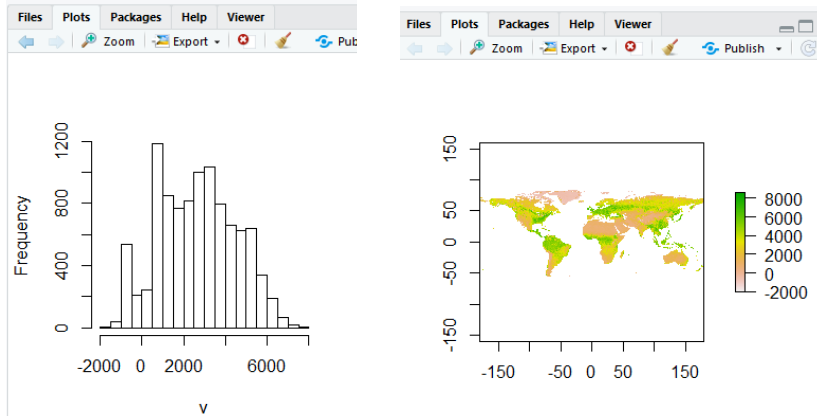
The screenshot shows the R Help Viewer interface. At the top, the R logo is visible. Below it, the 'Help' tab is selected in the menu bar. The search bar contains the text 'csv'. The main content area displays 'Search Results' with the R logo. Below this, a message states 'The search string was "csv"'. Under the heading 'Help pages:', there is a list of search results:

- [psych::read.clipboard](#) Shortcuts for reading from the clipboard or a file
- [readr::Tokenizers](#) Tokenizers.
- [readr::format\\_delim](#) Convert a data frame to a delimited string
- [readr::read\\_delim](#) Read a delimited file (including

# Gráficos

La pestaña *Plots* muestra los gráficos que resultantes de usar distintas funciones que generan gráficos.

Los gráficos de R base no son muy lindos, por eso se usan paquetes que mejoran las salidas gráficas (ej. *ggplot2*)



[http://rpubs.com/sinhrks/plot\\_surv](http://rpubs.com/sinhrks/plot_surv)

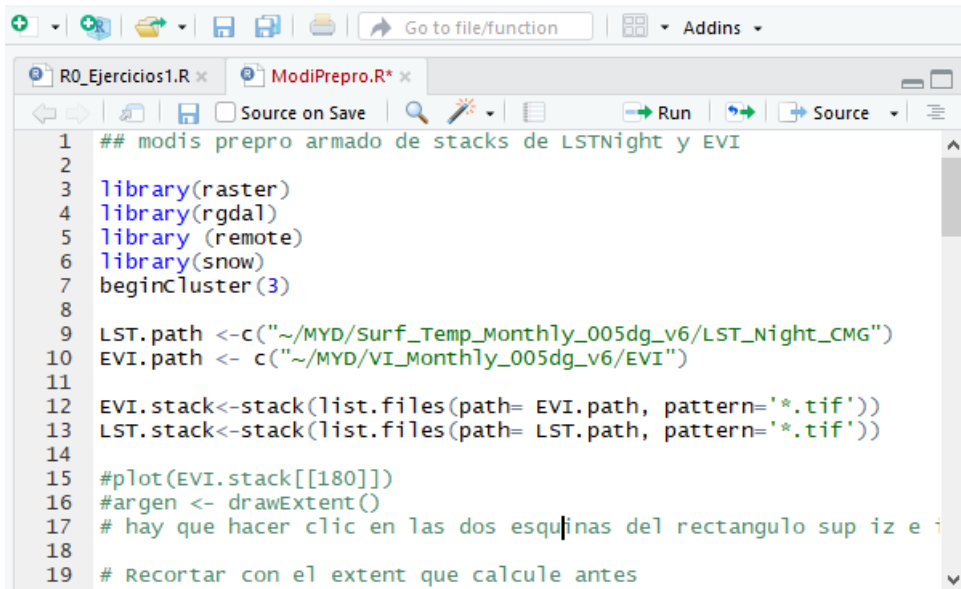
<http://www.sthda.com/english/articles/24-ggpubr-publication-ready-plots/81-ggplot2-easy-way-to-mix-multiple-graphs-on-the-same-page/>

# Panel Source, Fuente o Código

Es donde escribimos y documentamos nuestro código para guardarlo en un archivo y poder reproducirlo y modificarlo según sea necesario.

Las librerías que se van a usar las cargamos siempre al inicio del script.

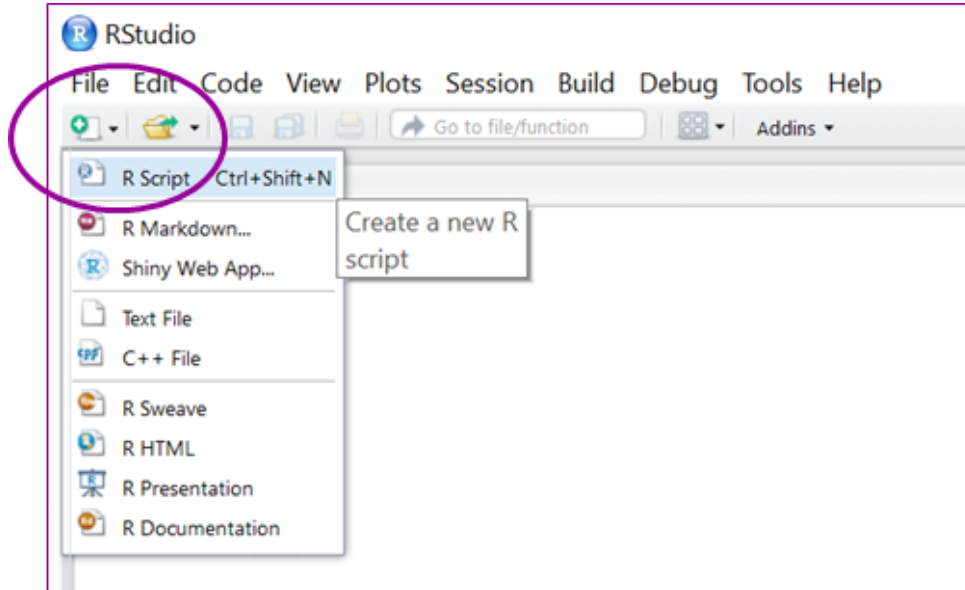
Para comentar el código usamos `#`.  
R no soporta comentarios multi-lineas o bloques de comentario.



```

1  ## modis prepro armado de stacks de LSTNight y EVI
2
3  library(raster)
4  library(rgdal)
5  library(remote)
6  library(snow)
7  beginCluster(3)
8
9  LST.path <- c("~/MYD/Surf_Temp_Monthly_005dg_v6/LST_Night_CMG")
10 EVI.path <- c("~/MYD/VI_Monthly_005dg_v6/EVI")
11
12 EVI.stack<-stack(list.files(path= EVI.path, pattern='*.tif'))
13 LST.stack<-stack(list.files(path= LST.path, pattern='*.tif'))
14
15 #plot(EVI.stack[[180]])
16 #argen <- drawExtent()
17 # hay que hacer clic en las dos esquinas del rectángulo sup iz e i
18
19 # Recortar con el extent que calcule antes
  
```

## Panel Source o Fuente o Código

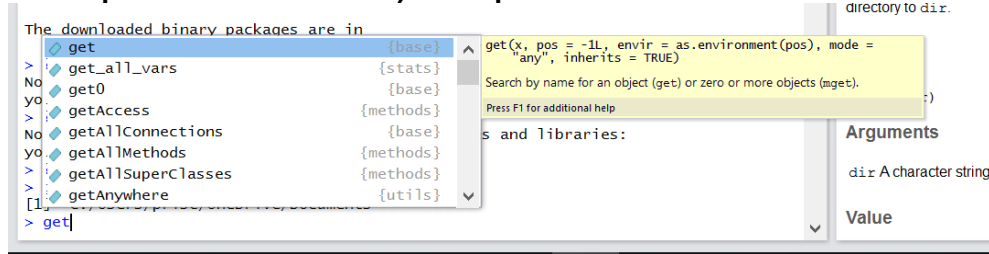


Crear un archivo nuevo



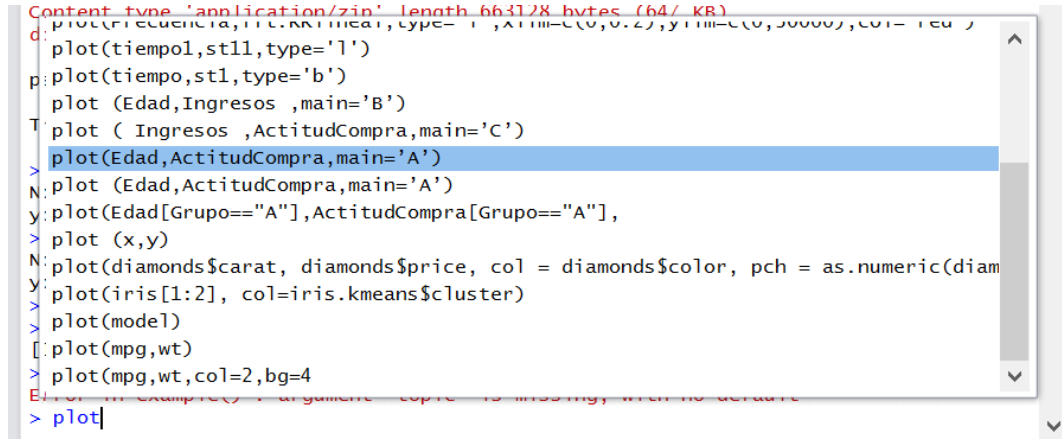
# Panel Source o Fuente o Código

empezar a escribir, empezar a escribir + TAB



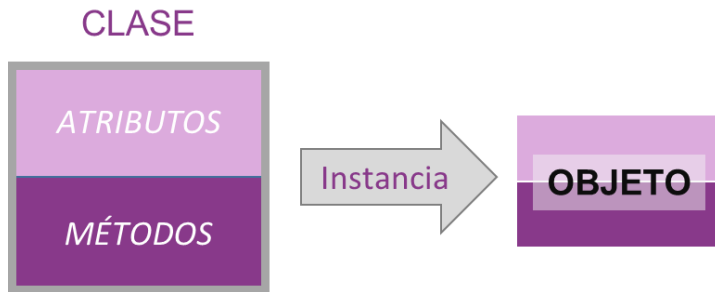
Autocompletado de código

empezar a escribir Control+ Arriba da listado de instancias donde se uso ese código



## 2. Elementos de la sintaxis de R

Todo en R son **objetos**



Un **objeto** es una ubicación de memoria con un valor y con un identificador para poder accederlo.

Los objetos tienen propiedades o estados (**atributos**) y comportamientos (**métodos**) característicos.

Una **clase** es un diseño de la estructura que tiene un objeto. Describe las propiedades y comportamiento común a todos los objetos de la misma clase. Las clases se utilizan para crear objetos, que son instancias de una clase dada.

# Elementos de la sintaxis de R

Todo en R son **objetos**

- Asignación y Evaluación
- Tipos de datos
- Estructuras de datos
- Operadores
- Funciones
- Variables globales vs locales o temporales

*Variable*      *Asignación*      *Función* (c)      *Tipo de dato* (numeric)

```

> edad <- c(22, 22, 23, 24, 25, 25, 26, 27, 28, 29, 29, 29, 29, 29,
31, 31, 32, 33, 34, 35, 35, 35, 36, 38, 39, 39, 42, 42, 44, 44, 45, 45,
45, 47, 48, 52, 59, 66, 67, 69, 69)
  
```

*Estructura de datos* (vector)

*Función*      *Variable*

```

> mean(edad)
[1] 38.26829
  
```

*Evaluación*

# Tipos de datos

- **Character** (alfanúmericos letras y números)
- **Integer** (enteros)
- **Numeric** (reales)
- **Logical** (lógicos: True / False)
- **Otros** : ej. **complex** (números complejos), **date** (fecha), **time** (fecha y hora)

## Valores especiales

<i>pi</i>	# Circunferencia de la Tierra en el Ecuador en km $2 * \pi * 6378$
<i>Inf</i>	1/0
<i>NULL</i>	vacío
<i>NA</i>	not available [character, logical, numeric]
<i>NaN</i>	not a number, indefinido, 0/0

## Coerción

**Coerción implícita** R interpreta automáticamente con qué tipo de datos está trabajando cuando lee datos de tipo character, numeric o logical. Si lee un dato y no lo puede asignar, pone NA y avisa con un *warning*.

**Coerción explícita** se le indica a R que interprete el tipo de datos de una manera determinada, usando funciones como:

```
as.character()  
as.numeric()  
as.logical()
```

## Testeo

`class(x)`

```
var1 <- 0+1i  
class(var1)  
[1] "complex"
```

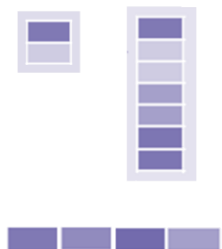
`is.tipo_de_objeto(x)`

```
is.character(x)  
is.logical(x)  
is.numeric(x)  
is.integer(x)
```

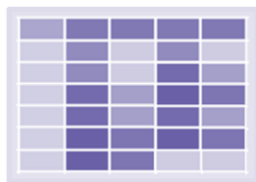
```
is.na(x)  
is.nan(x)
```

# Estructuras de datos

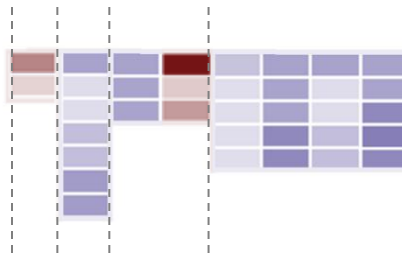
Otro tipo de objetos son las estructuras de datos, también llamados objetos datos, que son organizaciones de datos que quedan como objetos internos a R.



Vector



Matrix



List



Data.frame

(Tablas)



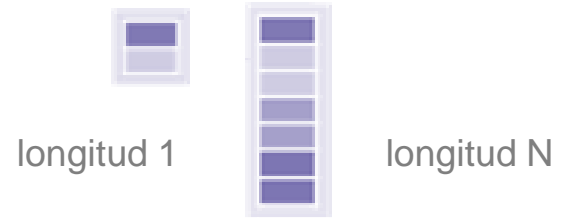
# Vector



Tiene sólo objetos de una **misma clase**  
(mismo tipo)

Conjunto de datos **ordenado** en una  
**dimensión**

Pueden contener 1 o más de un  
elementos



Los vectores tienen **longitud** y **clase**

```
length(n)  
class(n)
```

## Distintas formas de crear vectores



```
x <- c(1, 2, 3)
y <- c("a", "b", "c")
z <- c(TRUE, TRUE, FALSE)
```

función c() concatena valores

```
m <- vector()
m
w <- vector("numeric", length = 10)
w
```

```
n <- 1:20
n
```

```
1:10
```

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

```
> rep(1:4,2) #repetimos 1:4 dos veces.
```

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

```
> rep(1:4,each=2)
```

```
[1] 1 1 2 2 3 3 4 4 #intercalando el resultado.
```

función rep() permite repetir secuencias de valores

el operador : permite generar secuencias

## coerción implícita

R lleva a todos los valores al mismo tipo de datos

```
X <- c(1.7, "a")  
Y <- c(TRUE, 2)  
Z <- c("a", TRUE)
```

## coerción explícita

```
x <- 0:6  
class(x)  
## [1] "integer"  
as.numeric(x)  
## [1] 0 1 2 3 4 5 6  
as.logical(x)  
## [1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE
```

Si R no puede definir como realizar esta coerción, entonces el resultado es NA (not available)

```
z <- c("a", "b", "c")  
as.numeric(z)  
## Warning: NAs introducidos por coerción  
## [1] NA NA NA  
as.logical(z)  
## [1] NA NA NA  
as.complex(z)  
## Warning: NAs introducidos por coerción  
## [1] NA NA NA
```

## ¿Cómo extraer elementos de un vector según su posición?

- `x[i]`            `i` es un índice que se usa para denominar el `i`-ésimo elemento del objeto `x`  
`x[h:n]`        se puede usar un vector de números contiguos generados por el operador ":"  
`x[c(i,j,n)]`   se puede usar un vector armado con la concatenación de posiciones no contiguas

```
x <- c(1,4,3,7) #creamos un vector de 4 elementos
x[1]           #pedimos el primer elemento
x[4]           #pedimos el cuarto elemento
x[2:3]         #pedimos del segundo al tercer elemento
x[c(2,4)]      #pedimos el segundo y el cuarto elemento
```

# Matrix

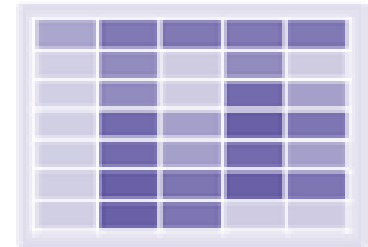
Conjunto de **datos del mismo tipo**

Ordenado en **dos dimensiones**, de longitudes  $N \times M$   
(o sea que es un conjunto de vectores)

atributo **dimensión**: vector de enteros de longitud 2 (número de filas  $N$  y número de columnas  $M$ ).

M columnas

N filas



Matrix

## Creando matrices



Las matrices se construyen **columna a columna**, empezando por la fila superior izquierda, completando las columnas para abajo hasta alcanzar la dimensión de filas, y luego sigue ese procedimiento agregando las columnas siguientes de a una por vez hacia la derecha.

```
m <- matrix(nrow = 2, ncol = 3)
m
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
dim(m)
## [1] 2 3
attributes(m)
## $dim
## [1] 2 3
```

```
m <- matrix(1:6, nrow = 2, ncol = 3)
m
dim(m)
attributes(m)
n <- matrix(1:6, nrow = 3, ncol = 2)
n
dim(n)
attributes(n)
```

## Creando matrices



También se puede crear una matriz directamente desde un vector, agregándole un atributo de dimensión.

```
m <- 1:10
m
## [1] 1 2 3 4 5 6 7 8 9 10
dim(m) <- c(2, 5)
m
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

¿Qué diferencias hay entre estas dos matrices?

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



## Creando matrices

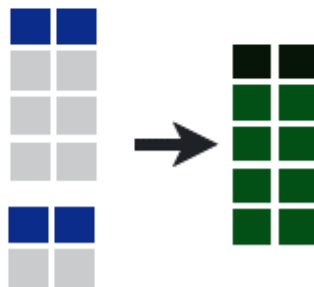


También se pueden crear agrupando por columnas o por filas con las funciones **cbind()** y **rbind()**

**cbind**



**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
```

# List

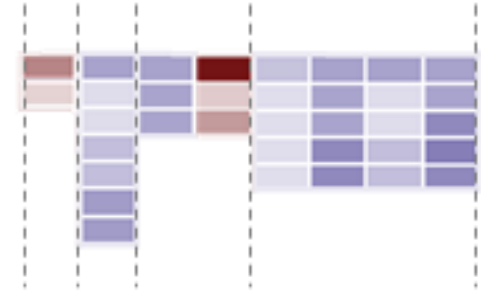
Conjunto de datos ordenado de **distinto tipo**

Varias dimensiones: cada posición del ordenamiento o elemento es una dimensión distinta

Cada dimensión pueden ser de distinto tamaño

Las listas se crean explícitamente con `list()`.

R no coerciona la entrada de objetos en una lista



## List

Tienen **clase** list

Atributos: dimensiones,  
nombre de dimensiones

Las dimensiones pueden tener nombres que se indican como:

```
nombre_lista$nombre_dimension
```

Para acceder a cada elemento primero se nombra la dimensión y luego el elemento dentro de la dimensión

Se hace referencia a la dimensión con `[[i]]` donde `i` es el índice de posición de la dimensión.

También se puede usar el nombre de la posición

```
x <- list(1, "a", c(TRUE,FALSE))
x
#[[1]] # Dimension 1
#[1] 1 # Un elemento numérico en la dimensión 1 de valor 1
#[[2]] # Dimension 2
#[1] "a" # Un elemento character en la dim 2 con valor "a"
#[[3]] # Dimension 3
#[1] TRUE FALSE # 2 elementos en la dim 3 con valores logicos T y F
```

Estructura con filas y columnas

Todas las **columnas** tienen el **mismo número de filas**

Cada **columna** tiene **datos del mismo tipo**

Pueden almacenar **distinto tipo de datos en cada columna**

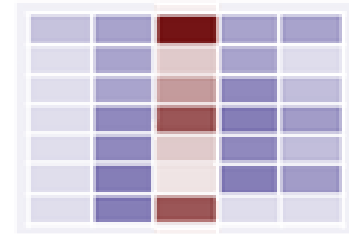
Es el equivalente de una tabla relacional

Es un caso especial de lista.

Es una lista de vectores que tienen todos la misma longitud.

Tiene dos dimensiones como las matrices

## Data.frame



### Data.frame

clase data.frame

atributos : nombres de filas y columnas, longitud (=ncols) y numero de filas (nrow())

`data.frame()` toma vectores con nombres como datos de entradas

```
> x <- data.frame(sitios = 1:4, muestreado = c(T, T, F, F))
```

```
> x
```

	sitios	muestreado
1	1	TRUE
2	2	TRUE
3	3	FALSE
4	4	FALSE

```
> attributes(x)
```

```
$names  
[1] "sitios"      "muestreado"
```

```
$row.names  
[1] 1 2 3 4
```

```
$class  
[1] "data.frame"
```

```
> x$sitios
```

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

```
> x$muestreado
```

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

## Creando Data.frames

Se pueden crear combinando data frames usando `cbind()` y `rbind()`

La forma mas común es leyendo un archivo, ej. usando las funciones `read.table()` o `read.csv()`.

Por defecto la creacion de data.frames convierte los datos de texto en factors. Hay que indicar *stringsAsFactors = FALSE* para evitar esto

## Testeo

Se usa `class()` o `is.data.frame()`

## Coerción

Se puede coercionar un objeto para que se convierta en un `data.frame` con `as.data.frame()`

Si el objeto es un vector, se generará un `data.frame` de una sola columna

Si es una lista, se va a crear una columna por cada elemento o dimension de la lista, si no tienen todos la misma longitud da error

Si es una matrix, crea un `data.frame` con igual numero de columnas y filas

# Atributos

Names, dimnames

Dimensions

Class

Length

Otros atributos/metadata definidos por el usuario

Objeto	Set nombres col	Set nombres filas
Data frame	<code>names()</code>	<code>row.names()</code>
Matrix	<code>colnames()</code>	<code>rownames()</code>



## Ejercicio 0

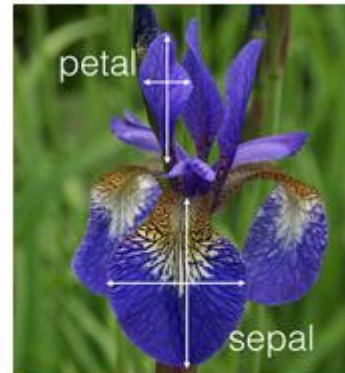
Crear un archivo nuevo de R en el directorio de trabajo y llamarlo iris.R

Escribir el siguiente código para explorar el dataset *iris* que trae con R base

```
class(iris)
dim(iris)
names(iris)
str(iris)
attributes(iris)
summary(iris)
```

Selecciona una o más líneas y hace  
Control+Enter para correrlas

*Petalos y sepalos*



*Iris setosa*



*Iris versicolor*



*Iris virginica*

Fisher, R. A. (1936) The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7, Part II, 179–188. The data were collected by Anderson, Edgar (1935). The irises of the Gaspé Peninsula, *Bulletin of the American Iris Society*, 59, 2–5.

```
> class(iris)
[1] "data.frame"
> dim(iris)
[1] 150 5
> names(iris)
[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
> str(iris)
'data.frame': 150 obs. of 5 variables:
 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ..
> attributes(iris)
$names
[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"

$row.names
 [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
[20] 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
[39] 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
[58] 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76
[77] 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
[96] 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114
[115] 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133
[134] 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150

$class
[1] "data.frame"

> summary(iris)
Sepal.Length Sepal.Width Petal.Length Petal.Width Species
Min. :4.300 Min. :2.000 Min. :1.000 Min. :0.100 setosa :50
1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300 versicolor:50
Median :5.800 Median :3.000 Median :4.350 Median :1.300 virginica :50
Mean :5.843 Mean :3.057 Mean :3.758 Mean :1.199
3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800
Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500

>
.
```

Y luego agregar esto

```
iris[1:5,]  
head(iris)  
tail(iris)  
iris[1:10, "Sepal.Length"]  
iris$Sepal.Length[1:10]
```



```
> iris[1:5,]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1           3.5          1.4          0.2  setosa
2          4.9           3.0          1.4          0.2  setosa
3          4.7           3.2          1.3          0.2  setosa
4          4.6           3.1          1.5          0.2  setosa
5          5.0           3.6          1.4          0.2  setosa
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1           3.5          1.4          0.2  setosa
2          4.9           3.0          1.4          0.2  setosa
3          4.7           3.2          1.3          0.2  setosa
4          4.6           3.1          1.5          0.2  setosa
5          5.0           3.6          1.4          0.2  setosa
6          5.4           3.9          1.7          0.4  setosa
> tail(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
145          6.7           3.3          5.7          2.5 virginica
146          6.7           3.0          5.2          2.3 virginica
147          6.3           2.5          5.0          1.9 virginica
148          6.5           3.0          5.2          2.0 virginica
149          6.2           3.4          5.4          2.3 virginica
150          5.9           3.0          5.1          1.8 virginica
> iris[1:10, "Sepal.Length"]
[1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
> iris$Sepal.Length[1:10]
[1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
```

... y luego agrega esto

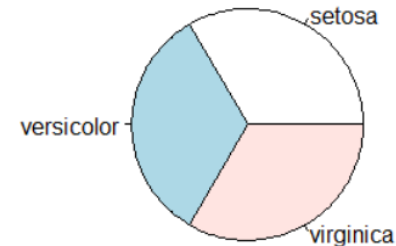
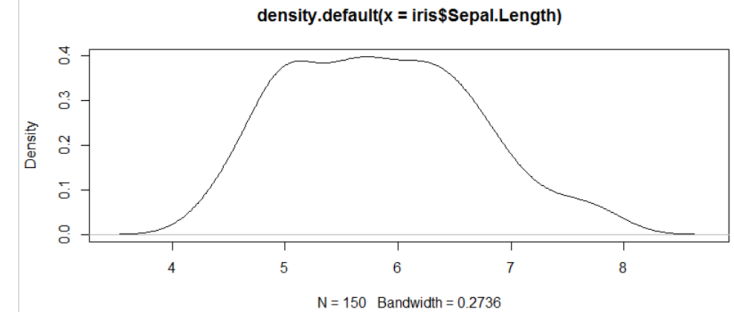
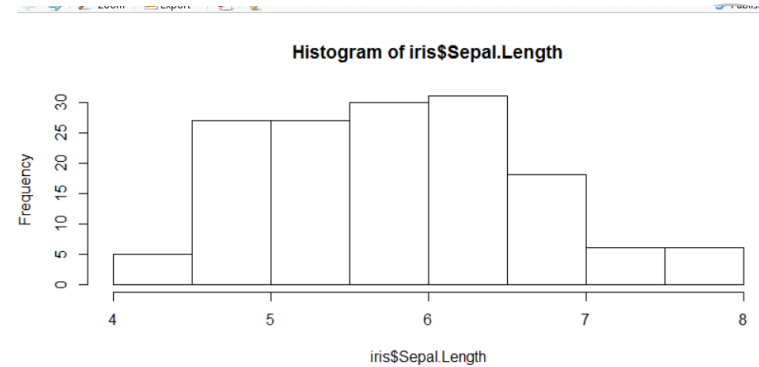
```
hist(iris$Sepal.Length)
```

```
plot(density(iris$Sepal.Length))
```

```
pie(table(iris$Species))
```

Guarda los cambios.

Hiciste tu primer script para explorar un dataset!



# Factor

Los factores son importantes en modelos estadísticos ya que definen los estratos o grupos a comparar o diferenciar y se usan especialmente en funciones que generan modelos como `lm()` y `glm()`.

es un vector que almacena datos categóricos  
puede ser atómico (está solo) o ser una columna de un dataframe  
puede tener etiquetas (*label*) que denominan a cada valor del factor y lo auto-describen  
R automáticamente setea los niveles de un factor en orden alfabético  
un factor tiene atributos de nivel (*level*) y clase "factor".

### Con factor()

```
x <- factor(c("yes", "yes", "no", "yes", "no"))
x
## [1] yes yes no  yes no
## Levels: no yes
table(x)
## x
##  no yes
##   2  3
```

Con **read.table()** se crean factores automáticamente cuando encuentran datos que interpreta como caracteres o cadenas. Los “levels” (niveles de un factor) son las posibles categorías

El orden de los niveles de un factor se pueden establecer usando el argumento *levels* en la función `factor()`

Un factor puede ser nominal o puede tener categorías ordenadas

```
> x <- factor(c("Hombre", "Mujer", "Hombre", "Hombre", "Mujer"))
>x
[1] Hombre Mujer  Hombre Hombre Mujer
Levels: Hombre Mujer
```

```
## tabla de frecuencia simple
>table(x)
x
Hombre  Mujer
      3      2
```



# Operadores

Los operadores son tipos de funciones especiales

Las operaciones pueden ser agrupadas usando paréntesis, y asignadas a variables de manera directa

Los operadores básicos son los aritméticos, lógicos, relacionales (de comparación)

Aritméticos		Relacionales		Lógicos	Otros
+	suma	==	exactamente igual que	& AND	~ tilde
-	resta	!=	diferente de	! NOT	: secuencia
*	multiplicación	<	menor que	OR	? ayuda
^ o ** exponenciación		<= menor o igual que		all	<- asignación izquierda
/	división	> mayor que		any	\$ subconjunto
%/% división entera		>= mayor o igual que			
%% residuo o módulo					
%*% multiplicación matricial					

## Operaciones vectorizadas

R permite trabajar con datos en vectores, y la mayoría de las operaciones aritméticas y funciones matemáticas están **vectorizadas**.

Típicamente se trabaja con vectores de la misma longitud, y la operación se define elemento a elemento entre los dos vectores. El resultado es un vector de la misma longitud que los originales, en el que se indica, elemento a elemento cuál es el resultado de la operación.

## Operaciones vectorizadas y reciclado de vectores

¿Qué pasa cuando los vectores operandos no son de la misma longitud?

En esos casos, el intérprete del lenguaje R procede a completar la operación **reciclando** los elementos del operador de menor longitud hasta alcanzar la longitud del operando mayor.

```
x<-c(1,2,3)
y<-"Alumno"
```

```
>z<- paste(y,x)
>z
[1] "Alumno 1" "Alumno 2" "Alumno 3"
```

```
>z<- paste(y,x, sep="")
>z
[1] "Alumno1" "Alumno2" "Alumno3"
```

## Reglas de reciclado (tomando como ejemplo la suma)

Si uno trata de sumar dos estructuras con distinto numero de elementos, la más corta es reciclada a la longitud de la mas larga

Si la longitud del vector más grande no es multiplo del más corto, da un mensaje de advertencia

Si una operación aritmética involucra un vector de longitud cero, entonces el vector resultante también tendrá longitud cero.

```
> c(1, 2, 3, 4) + c(0, 10)
[1]  1 12  3 14
```

```
> c(1, 2, 3, 4) + c(0, 10, 100)
[1]  1 12 103  4
Warning message:
In c(1, 2, 3, 4) + c(0, 10, 100) :
  longer object length is not a multiple of shorter object length
```

```
> a <- as.numeric(vector())
> b <- c(1,2,3)
> a+b
numeric(0)
```

Una función muy útil para crear vectores haciendo repeticiones con la función **rep()**, que usa reciclado de vectores.

```
rep(1:4, 2)
rep(1:4, each = 2)      # no es igual al primero
rep(1:4, c(2,2,2,2))    # igual al segundo
rep(1:4, c(2,1,2,1))
rep(1:4, each = 2, len = 4)  # longitud es 4!
rep(1:4, each = 2, times = 3) # longitud 4, 3 repeticiones completas
```

# Funciones

Una función es un conjunto de instrucciones para realizar una tarea específica

Puede aceptar argumentos o parametros

Puede devolver uno o más valores o ninguno

Usamos tanto las funciones que trae R preinstaladas como las que agregamos al cargar paquetes

Constan de :

- lista de argumentos (arglist)
- código (body)
- entorno en el cual son válidas las variables que se crean y usan para realizar la acción de la función

```
function(arglist){body}
```

```
mifuncion <- function(arg1, arg2, ... ){  
  instrucciones  
  return(object)  
}
```

¿Qué funciones vimos a lo largo de este taller?



# 3. Ejercicios

[https://github.com/pmnatural/R-cero/blob/master/Sintaxis\\_R/R0\\_Ejercicios1.R](https://github.com/pmnatural/R-cero/blob/master/Sintaxis_R/R0_Ejercicios1.R)

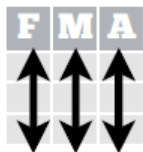
R0\_Ejercicios1.R



## 4. Datos ordenados (el enfoque tidyverse)

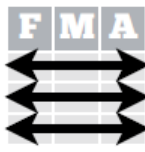
# Datos Ordenados - La base para domar datos en R

En un conjunto de datos ordenado:



Cada **variable** se tiene su propia **columna**

&

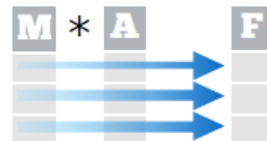


Cada **observación** tiene su propia **fila**

Datos ordenados complementan las

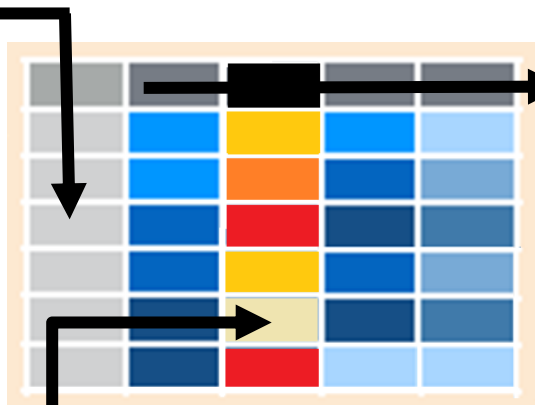
**operaciones vectorizadas** de R.

Automáticamente R preserva observaciones mientras manipulas las variables. No hay otro formato que funcione tan intuitivamente en R.



$M * A$

Nombres de observaciones/filas (en primera columna o por defecto es número de fila), permite extraer filas por nombre



Nombres de variables/columnas en primera fila, permite extraer variables por nombre

No hay observaciones nulas

Variables nominales u ordinales como factores para agrupar análisis o gráficos