DATA DISCOVERY

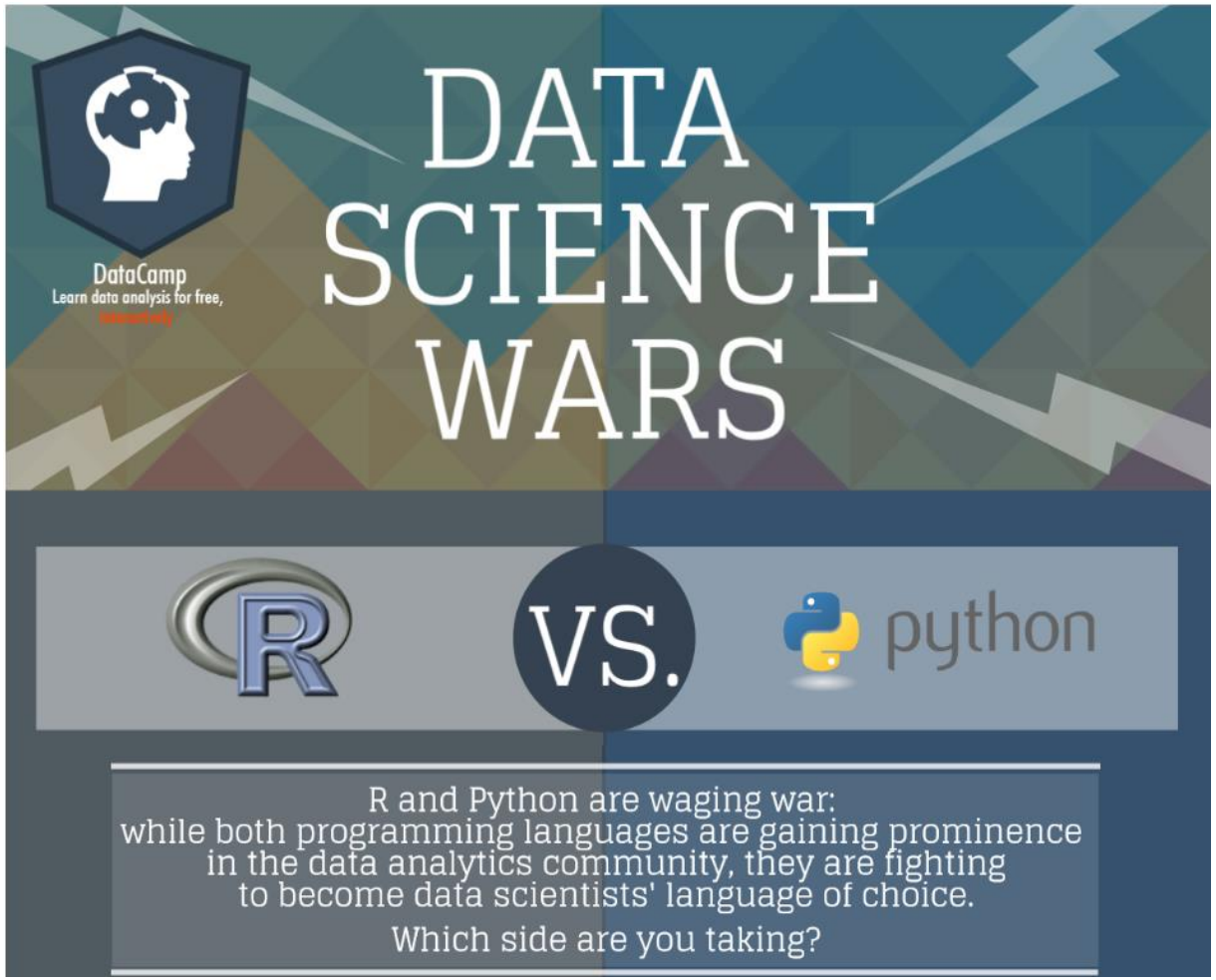
Introducción a R

MIGUEL GARCÍA MENA

datahack
BIG DATA FAMILY

Índice

R vs Python



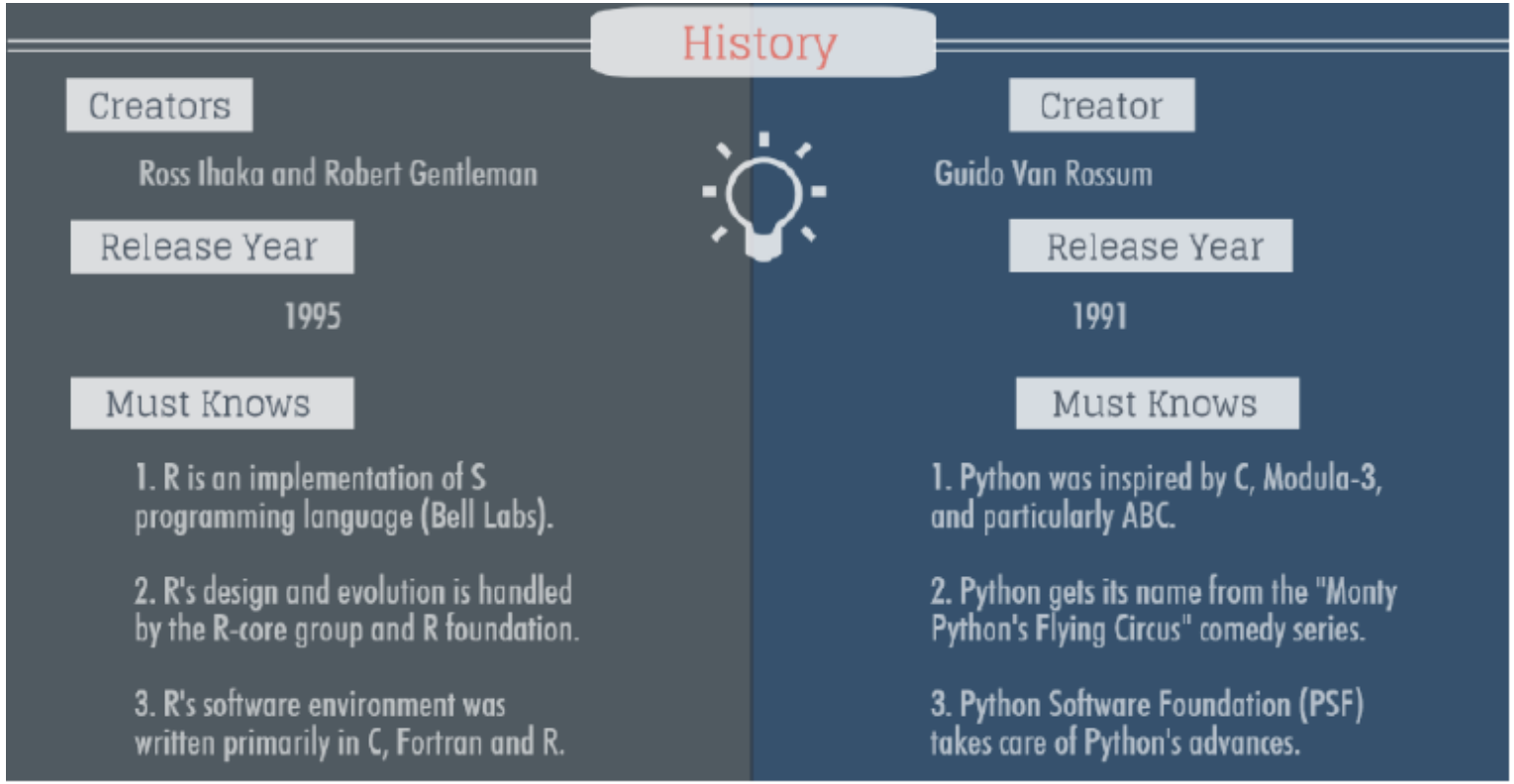
Antes de empezar a meternos a fondo con R.

Vamos a ver una comparativa de los dos principales lenguajes usados actualmente para el análisis de datos:

R y Python.

fuente: datacamp.com

R vs Python : Historia



Ambos lenguajes nacen en los 90's, basándose en otros lenguajes , con la idea de mejorarlos:

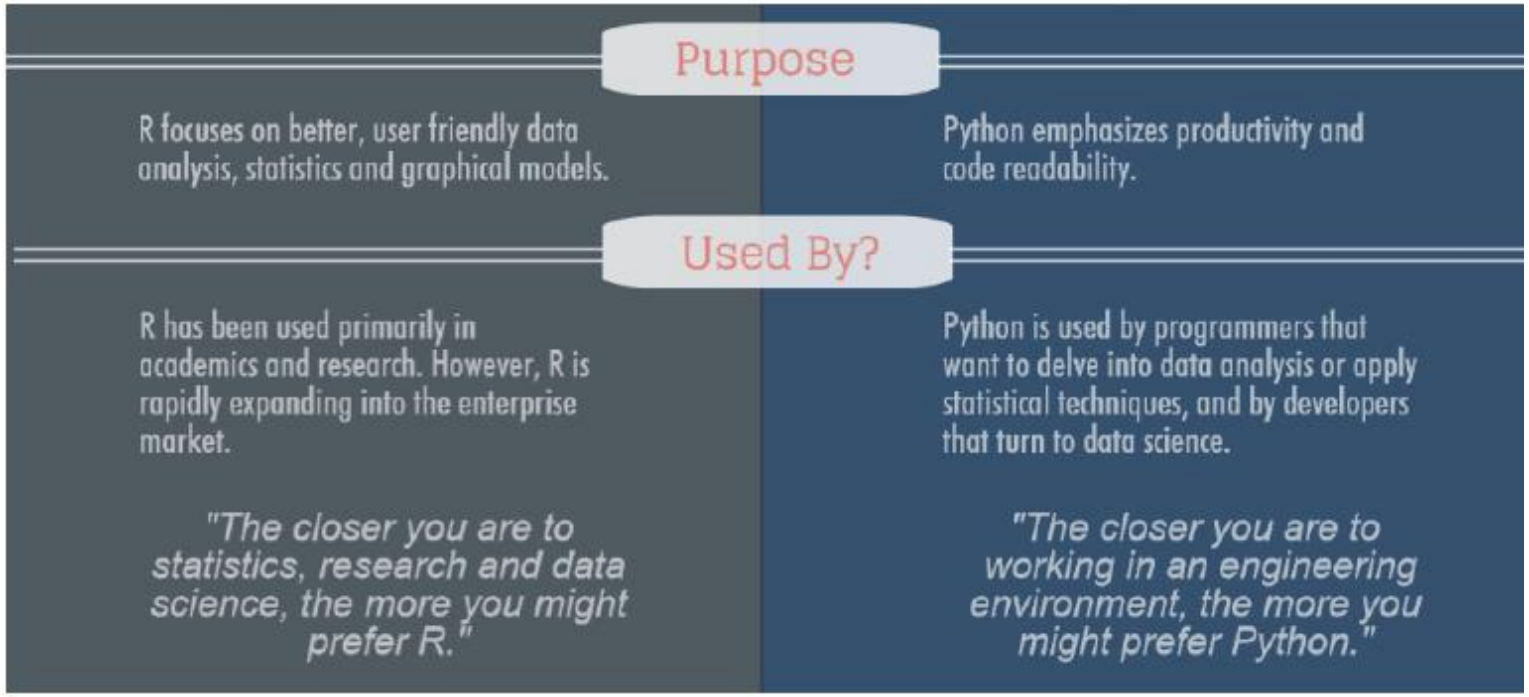
S->R

C-> Python.

Dos fundaciones están detrás de su mantenimiento y desarrollo.

fuente: datacamp.com

R vs Python : Propósito y Finalidad



R se centra en el análisis de datos de forma estadística y en la visualización de datos.

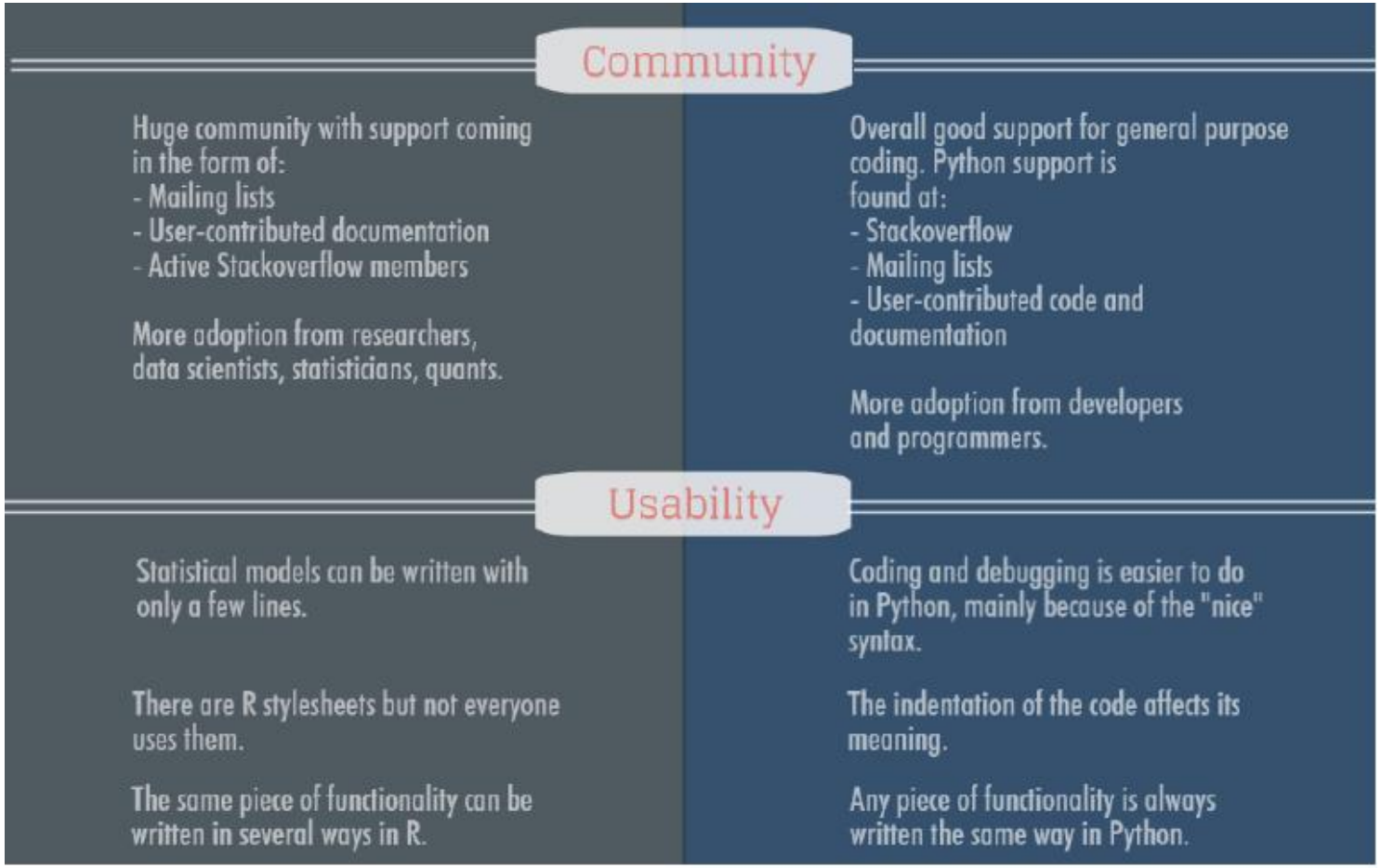
R tuvo al inicio más aceptación del mundo académico.

Python se centra en la productividad y en la fácil lectura del código.

Python se usa más por desarrolladores que migran hacia el análisis de datos

fuente: datacamp.com

R vs Python : Comunidad y forma de uso



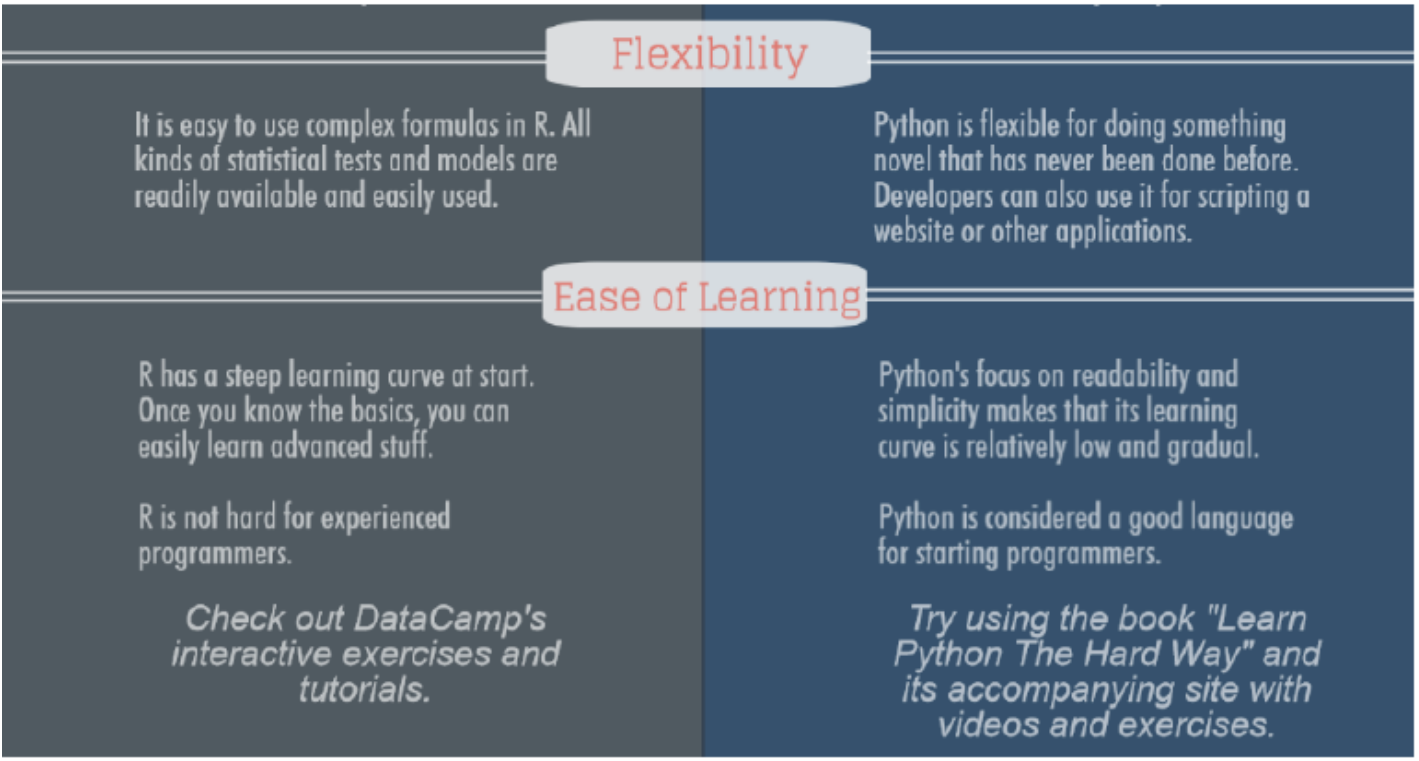
Uno de los principales activos de ambos lenguajes es la gran comunidad que da soporte.

En el modo de uso empezamos a encontrar más diferencias:

- Guías de estilo
- Indentación
- Flexibilidad para hacer la misma tarea, en R es mucho más flexible.

fuente: datacamp.com

R vs Python : Flexibilidad y Aprendizaje

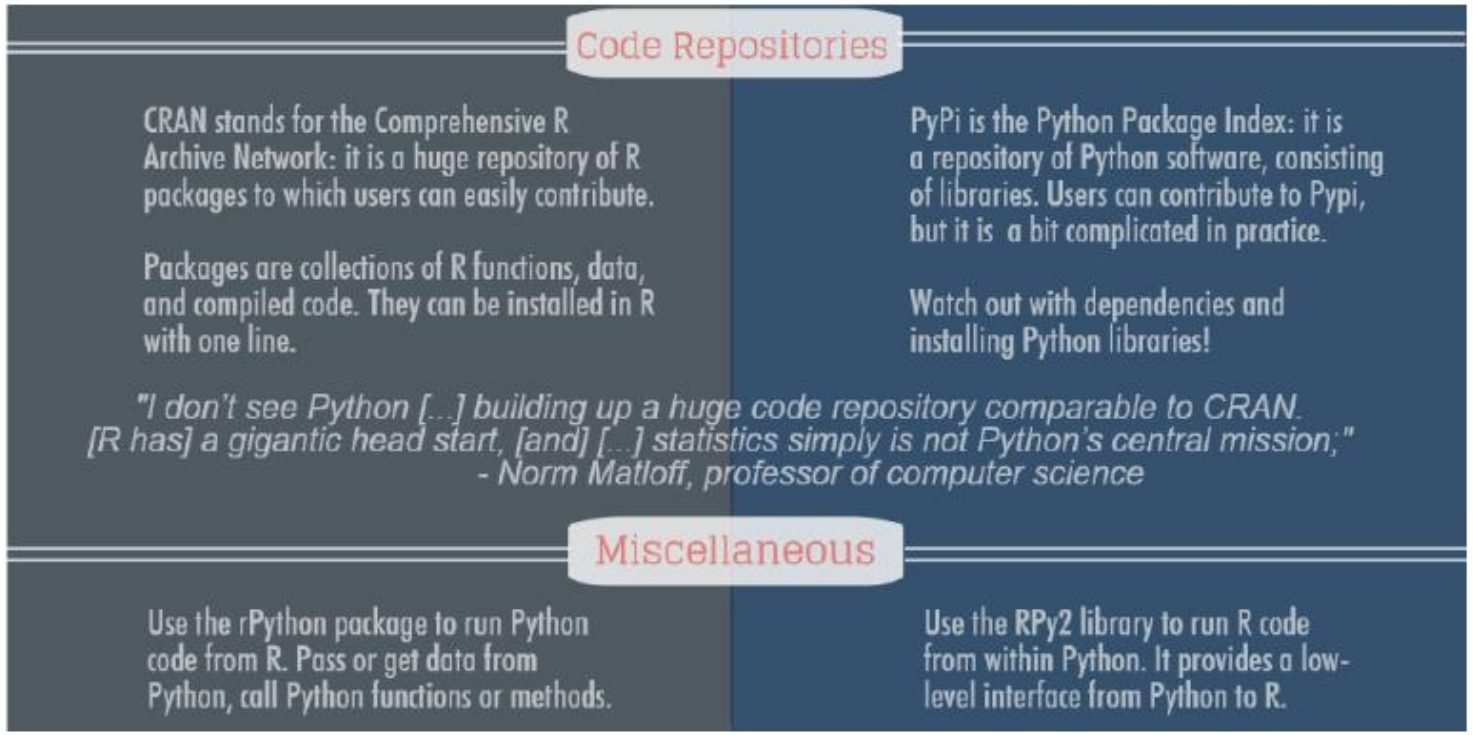


En estadística R es más flexible, sin embargo Python se puede usar en otras áreas, como desarrollo web.

La curva de aprendizaje es algo más complicada en R que en Python. Sobre todo para esas personas que ya hayan programado en otros lenguajes.

fuente: datacamp.com

R vs Python : Repositorios y otros



Ambos tienen repositorios donde podemos descargar librerías o paquetes, sin embargo el de R es mucho más extenso.

Podemos ejecutar código del otro lenguaje.

fuente: datacamp.com

R vs Python: Popularidad

https://www.tiobe.com/tiobe-index/

Jan 2021	Jan 2020	Change	Programming Language	Ratings	Change
1	2	↑	C	17.38%	+1.61%
2	1	↓	Java	11.96%	-4.93%
3	3		Python	11.72%	+2.01%
4	4		C++	7.56%	+1.99%
5	5		C#	3.95%	-1.40%
6	6		Visual Basic	3.84%	-1.44%
7	7		JavaScript	2.20%	-0.25%
8	8		PHP	1.99%	-0.41%
9	18	⇈	R	1.90%	+1.10%
10	23	⇈	Groovy	1.84%	+1.23%

¿Qué es R?

- Creado en 1993 por el Dpto. Estadística de Auckland
- Alternativa open source al lenguaje programación S
- Lenguaje de programación para el análisis estadístico
- Multiplataforma (Windows, Linux y Mac)
- Sintaxis muy simple e intuitiva
- Basado en librerías
- Comunidad muy activa (+ 7000 paquetes en CRAN)
- Mayoría de los objetos en RAM
- Uno de los mejores lenguajes para Data Science

Algo más que análisis estadísticos

- Análisis Estadísticos
- Data Mining y Limpieza de datos
- Estructurar conjuntos de datos
- Cambiar la forma de los datos
- Visualización de datos
- Análisis de Grafos
- Machine Learning
- Deep Learning
- Análisis de datos Interactivo
- Generación de Informes

Instalando R

Instalación de R

Seleccionamos nuestro sistema operativo para la instalación:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

Index of /bin/linux

Name	Last modified	Size	Description
Parent Directory		-	
debian/	2018-07-03 11:18	-	
redhat/	2014-07-27 21:12	-	
suse/	2012-02-16 15:09	-	
ubuntu/	2018-09-12 16:07	-	

R for Mac OS X

This directory contains binaries for a base distribution and packages to run on Mac OS X (release 10.6 and above). Mac OS 8.6 to 9.2 (and Mac OS X 10.1) are no longer supported but you can find the last supported release of R for these systems (which is R 1.7.1) here. Releases for old Mac OS X systems (through Mac OS X 10.5) and PowerPC Macs can be found in the old directory.

Note: CRAN does not have Mac OS X systems and cannot check these binaries for viruses. Although we take precautions when assembling binaries, please use the normal precautions with downloaded executables.

As of 2016/03/01 package binaries for R versions older than 2.12.0 are only available from the CRAN archive so users of such versions should adjust the CRAN mirror setting accordingly.

R 3.5.1 "Feather Spray" released on 2018/07/05

Important: since R 3.4.0 release we are now providing binaries for OS X 10.11 (El Capitan) and higher using non-Apple toolkit to provide support for OpenMP and C++17 standard features. To compile packages you may have to download tools from the tools directory and read the corresponding note below.

Please check the MD5 checksum of the downloaded image to ensure that it has not been tampered with or corrupted during the mirroring process. For example type
`md5 R-3.5.1.pkg`
in the *Terminal* application to print the MD5 checksum for the R-3.5.1.pkg image. On Mac OS X 10.7 and later you can also validate the signature using `pkgutil --check-signature R-3.5.1.pkg`

Lastest release:

R-3.5.1.pkg
MD5-hash: 58eaff65fbd024f267ef1e521e17e7f8
SHA1-hash: 76c01bfa62a6896d5f4a4511e25d17276d149621
(ca. 74MB)

R 3.5.1 binary for OS X 10.11 (El Capitan) and higher, signed package. Contains R 3.5.1 framework, R.app GUI 1.70 in 64-bit for Intel Macs, Tcl/Tk 8.6.6 X11 libraries and Texinfo 5.2. The latter two components are optional and can be ommitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from sources.

R for Windows

Subdirectories:

base	Binaries for base distribution. This is what you want to **install R for the first time**.
contrib	Binaries of contributed CRAN packages (for R >= 2.13.x; managed by Uwe Ligges). There is also information on third party software available for CRAN Windows services and corresponding environment and make variables.
old contrib	Binaries of contributed CRAN packages for outdated versions of R (for R < 2.13.x; managed by Uwe Ligges).
Rtools	Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

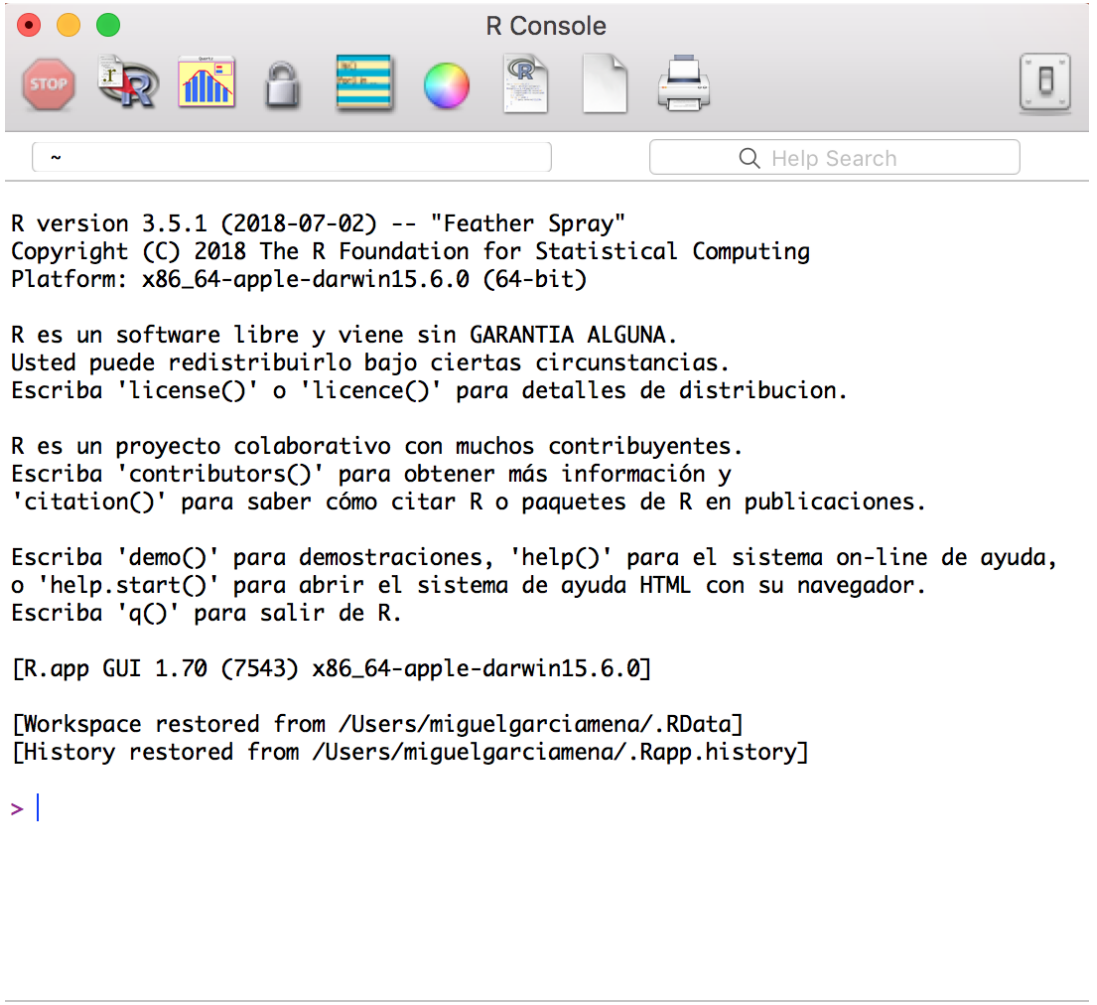
Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

Instalación de R

Una vez instalado ya tenemos acceso a la consola básica de R .



Instalación de RStudio

Una vez hemos finalizado la instalación de R, vamos a instalar RStudio, la UI (Interfaz de Usuario) más popular para trabajar con R.

Para ello vamos a acceder a su web:

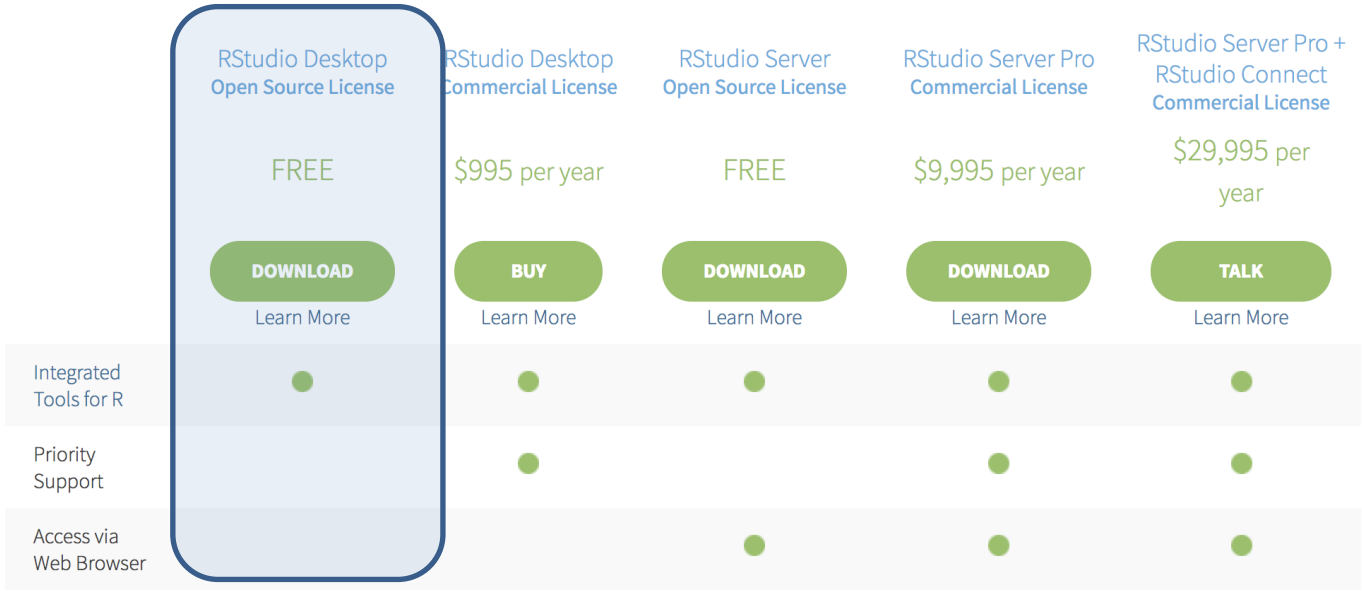
https://www.rstudio.com/

A la zona de descargas

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

Instalación de RStudio

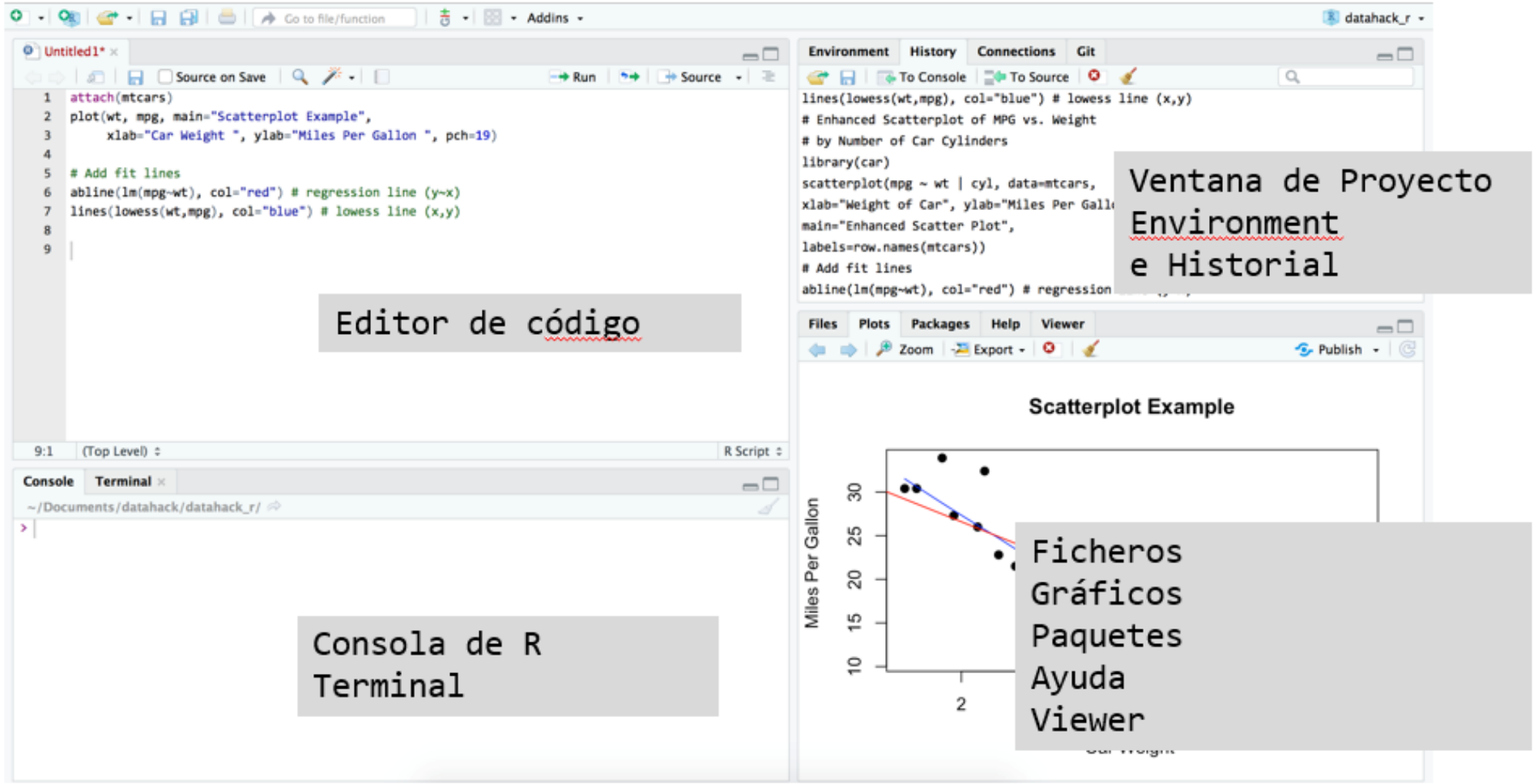
Seleccionaremos la versión RStudio Desktop
y la versión en base a nuestro sistema operativo



Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.1.456 - Windows Vista/7/8/10	85.8 MB	2018-07-19	`24ca3fe0dad8187aabd4bfbb9dc2b5ad`
RStudio 1.1.456 - Mac OS X 10.6+ (64-bit)	74.5 MB	2018-07-19	`4fc4f4f70845b142bf96dc1a5b1dc556`
RStudio 1.1.456 - Ubuntu 12.04-15.10/Debian 8 (32-bit)	89.3 MB	2018-07-19	`3493f9d5839e3a3d697f40b7bb1ce961`
RStudio 1.1.456 - Ubuntu 12.04-15.10/Debian 8 (64-bit)	97.4 MB	2018-07-19	`863ae806120358fa0146e4d14cd75be4`
RStudio 1.1.456 - Ubuntu 16.04+/Debian 9+ (64-bit)	64.9 MB	2018-07-19	`d96e63548c2add890bac633bdb883f32`
RStudio 1.1.456 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	88.1 MB	2018-07-19	`1df56c7cd80e2634f8a9fdd11ca1fb2d`
RStudio 1.1.456 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	90.6 MB	2018-07-19	`5e77094a88fdbdddddb0d35708752462`

Estructura Básica de RStudio



EJERCICIO 1

INSTALAR R Y RSTUDIO

Interfaz de R

- **Scripts**: Ficheros que contienen código R
  - Ficheros con extensión .R
  - Se cargan con la **source()**
  - Desde la shell del sistema con **Rscript**

```
> source("fmedia.R")
> |
```

```
Rscript Documents/datahack/IntroR/ejercicios/suma2.R
```

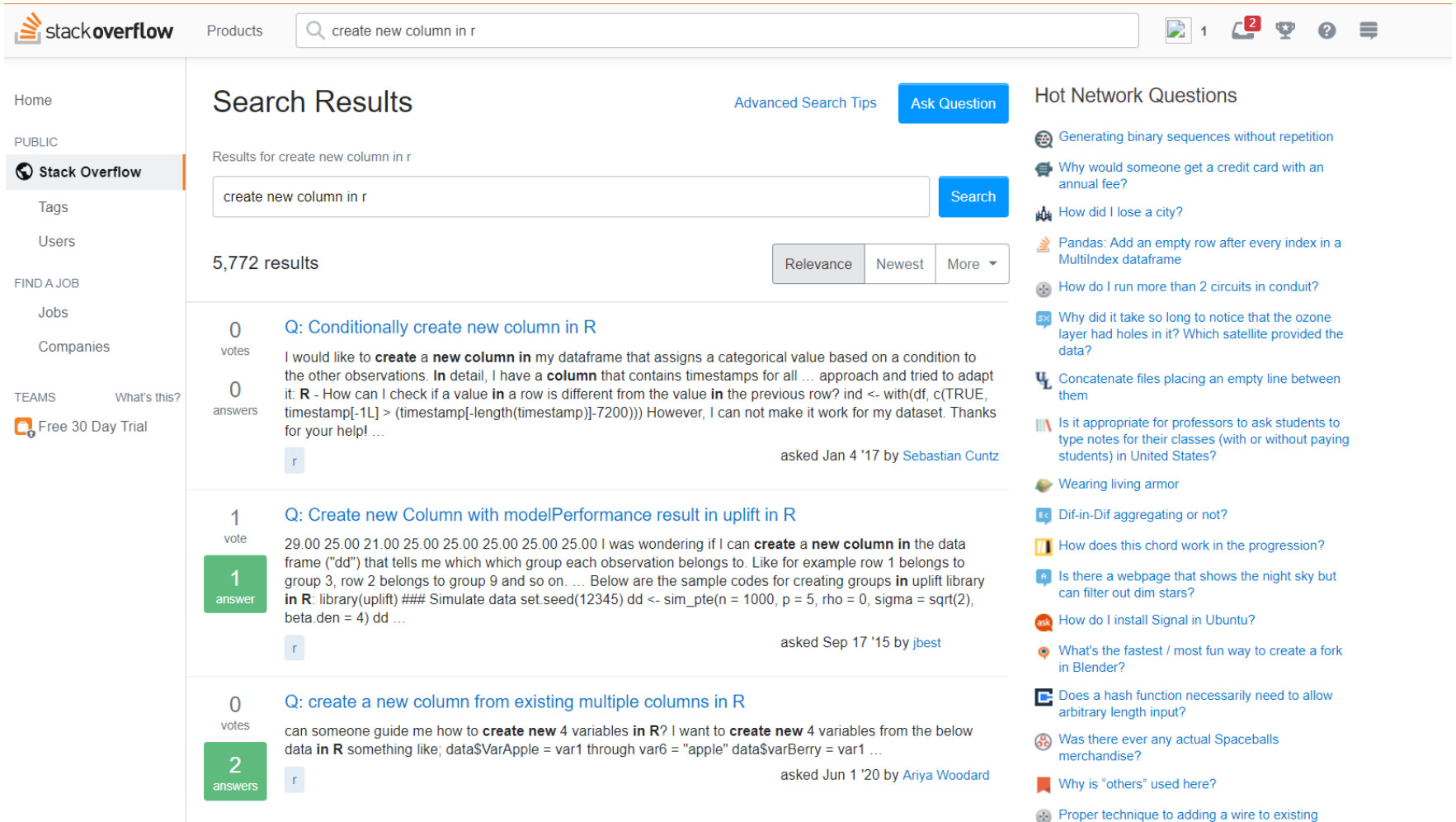
- **Comandos** interactivos que se introducen directamente en la consola
  - Buenos para mirar el aspecto que tienen los datos
  - Probar cosas
  - Mostrar gráficos

¿Cómo obtener ayuda?

- **help.start**()
  - Ayuda general
- **help**(mean)
  - Ayuda específica a una función
  - ?mean
- **help.search**("mean")
  - Encontrar una función en cualquier página de la ayuda
  - ??mean
- **example**(mean)
  - Mostrar un ejemplo de uso de una función

¿Cómo obtener ayuda?

http://stackoverflow.com



Índice

R como una calculadora

- La consola de R funciona como una calculadora
- R tiene los siguientes operadores aritméticos:

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

sqrt() -> *raíz cuadrada*

abs() -> *valor absoluto*

Operadores lógicos

- R tiene los siguientes operadores lógicos

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

```
> 5 > 4
[1] TRUE
>
> 6 >= 6
[1] TRUE
>
> 2 < 1
[1] FALSE
>
> 5 != 5
[1] FALSE
>
> ! T
[1] FALSE
>
> (2 == 2) & ( 4 > 2)
[1] TRUE
>
```

Variables

- Utilizar un nombre para almacenar un valor
- Se utiliza el operador de asignación **<- (Atajo** Alt + -)
- También se puede utilizar el símbolo igual **=**

```
> x <- 1
> |
```

- R es case sensitive

Tipos de datos

DECIMALES
4,5

ENTEROS
4

COMPLEJOS
(5 + 3i)

BOOLEANOS
TRUE O FALSE

TEXTO
“esto es un texto”

¿Cómo saber el tipo de variable?

La función **class()** informa del tipo de dato de una variable

```
> a <- 4.5
> b <- 4L
> c <- (5 + 3i)
> d <- TRUE
> e <- "VALOR"
```

```
> print(a)
[1] 4.5
> print(b)
[1] 4
> print(c)
[1] 5+3i
> print(d)
[1] TRUE
> print(c)
[1] 5+3i
```

```
> class(a)
[1] "numeric"
> class(b)
[1] "integer"
> class(c)
[1] "complex"
> class(d)
[1] "logical"
> class(e)
[1] "character"
```

Conversión de tipos

Los datos se pueden cambiar de tipo con **as.*()**

```
> x <- 0:6
> x
[1] 0 1 2 3 4 5 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
> as.character(x)
[1] "0" "1" "2" "3" "4" "5" "6"
```

EJERCICIO 2 – Operaciones Básicas

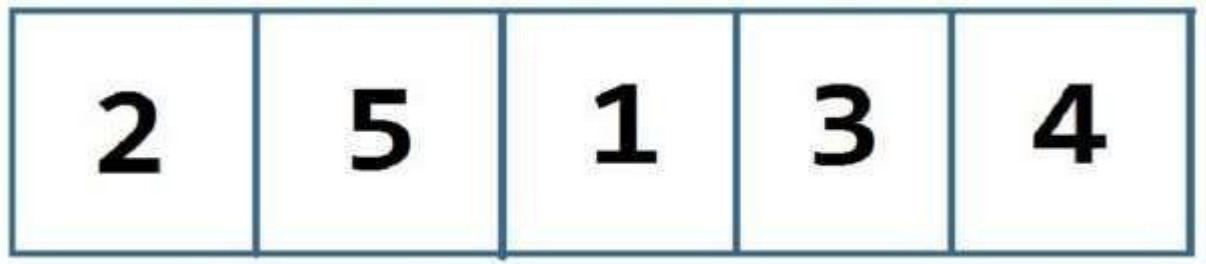
1. Crea dos variables, x e y, y asígnales un valor numérico
2. Crea una tercera variable (z),suma de las dos anteriores
3. Consulta su valor y tipo de dato
4. Crea una cuarta variable (a), resultado de dividir z entre 2
5. Consulta su valor y tipo de dato
6. Conviértela a entera y comprueba que lo ha hecho bien

Índice

Vectores

La estructura de datos básica de R es el **vector**

Un vector sólo puede contener datos del mismo tipo



Crear un Vector

El **operador ":"** se utiliza para crear vectores con secuencias de números

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
>
> x <- 1:100
> x
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
 [17]  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32
 [33]  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48
 [49]  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64
 [65]  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
 [81]  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96
 [97]  97  98  99 100
>
```

Crear un Vector

La función **c()** sirve para crear un vector con una lista de elementos

```
> x <- 1:5
>
> x
[1] 1 2 3 4 5
>
> x <- c(1, 2, 3, 4, 5)
> x
[1] 1 2 3 4 5
```

Crear un Vector

- La función **seq()** se utiliza para crear secuencias de números
- Pero si necesitamos un vector con una repetición de valores utilizamos **rep()**

```
> seq(from = 10, to = 20, by = 2)
[1] 10 12 14 16 18 20
>
> rep("A", 100)
  [1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
 [17] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
 [33] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
 [49] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
 [65] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
 [81] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
 [97] "A" "A" "A" "A"
>
```

Longitud de un vector

La función **length()** permite contar los elementos de un vector

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
>
> length(x)
[1] 10
>
```

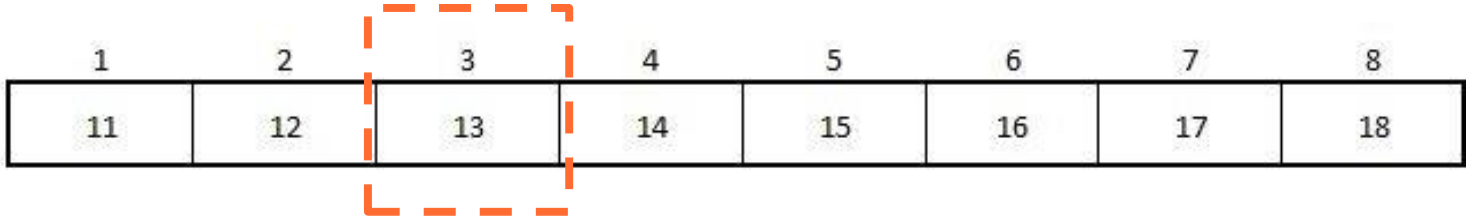
Operar con un vector

- En R las **operaciones son vectorizadas**
- Si se realiza una operación sobre un vector, se aplica a todos los elementos

```
> x
[1] 1 2 3 4 5
>
> x + 1
[1] 2 3 4 5 6
>
> x * 2
[1]  2  4  6  8 10
>
> x == 4
[1] FALSE FALSE FALSE  TRUE FALSE
>
> x >= 4
[1] FALSE FALSE FALSE  TRUE  TRUE
>
```

Seleccionando elementos en un vector

Para seleccionar un elemento del vector a través de su índice se utilizan los símbolos **[** y **]**



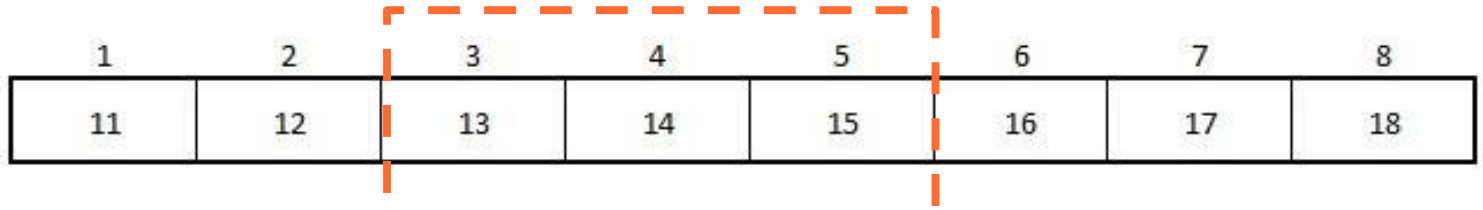
Seleccionando elementos en un vector

Para seleccionar un elemento del vector a través de su índice se utilizan los símbolos **[** y **]**

```
> x <- 11:18
>
> x
[1] 11 12 13 14 15 16 17 18
>
> x[3]
[1] 13
```

Seleccionando elementos en un vector

Es posible seleccionar más de un elemento…



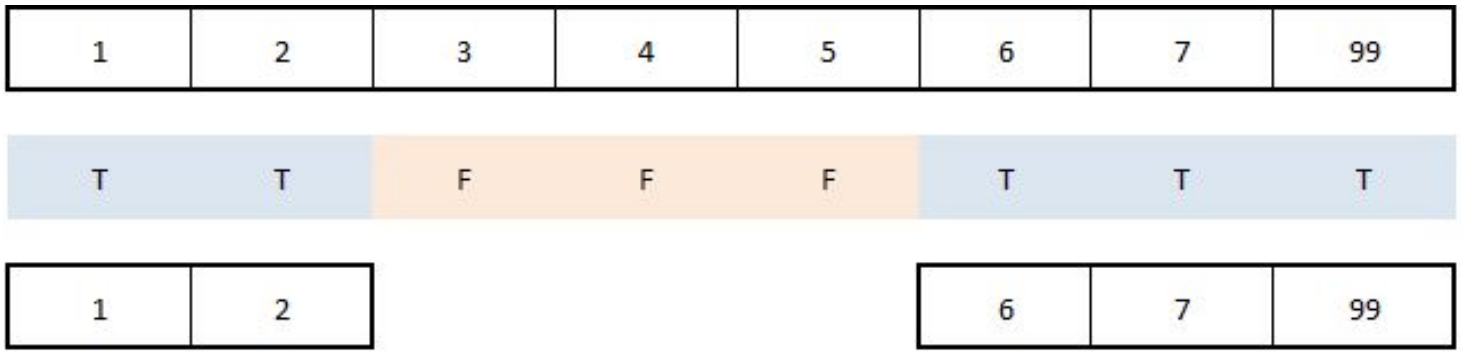
Seleccionando elementos en un vector

Es posible seleccionar más de un elemento…

```
> x <- c(11:18)
> x
[1] 11 12 13 14 15 16 17 18
>
> x [ 3:5 ]
[1] 13 14 15
>
```

Seleccionando elementos en un vector

En R se puede seleccionar elementos de un vector a través de otro vector de valores lógicos




```
> x <- c(1, 2, 3, 4, 5, 5, 6, 7, 99)
>
> y <- c(T, T, F, F, F, T, T, T, T)
>
> x[y]
[1]  1  2  5  6  7 99
>
```

Reemplazando elementos en un vector

Para reemplazar una posición concreta de un vector se utiliza el operador de asignación (**<-**) junto con el de selección **[ ]**

```
> x <- 1:5
> x
[1] 1 2 3 4 5
>
> x[1] <- 99
> x
[1] 99  2  3  4  5
>
```

Vectores con nombres

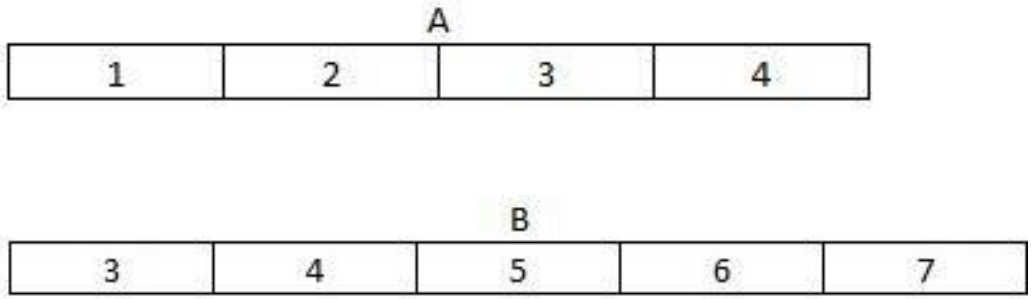
En R es posible asignar un nombre a cada elemento del vector y seleccionarlos a través de su nombre

```
> x <- 1:3
> x
[1] 1 2 3
> names(x) <- c("Uno", "Dos", "Tres")
> x
 Uno  Dos Tres
   1    2    3
> names(x)
[1] "Uno"  "Dos"  "Tres"
> x["Uno"]
Uno
  1
> unname(x)
[1] 1 2 3
> 
```

Operaciones en conjuntos

R tiene una serie de operaciones aplicables a conjuntos de vectores

```
> a <- 1:4
> b <- 3:7
> union(a,b)
[1] 1 2 3 4 5 6 7
> intersect(a,b)
[1] 3 4
> setdiff(a,b)
[1] 1 2
> setdiff(b,a)
[1] 5 6 7
> 2 %in% a
[1] TRUE
```



Ordenando un vector

La función **sort()** se utiliza para ordenar los elementos de un vector

```
> x <- c(3, 1, 3, 4, 5, 2, 3, 2, 1, 3, 4, 5)
>
> x
 [1] 3 1 3 4 5 2 3 2 1 3 4 5
>
> sort(x)
 [1] 1 1 2 2 3 3 3 3 4 4 5 5
>
> sort(x, decreasing = T)
 [1] 5 5 4 4 3 3 3 3 2 2 1 1
>
```

Contando elementos en un vector

La función **table()** se utiliza para contar los elementos de un vector

```
> x <- c(3, 1, 3, 4, 5, 2, 3, 2, 1, 3, 4, 5)
> x
 [1] 3 1 3 4 5 2 3 2 1 3 4 5
>
> sort(x)
 [1] 1 1 2 2 3 3 3 3 4 4 5 5
>
> table(x)
x
1 2 3 4 5
2 2 4 2 2
>
```

Números especiales

- **Inf** representa infinito / **-Inf** representa menos infinito
- **NaN** representa un valor indefinido (Not a Number)
- **NA** representa un valor inexistente

```
> 1 / 0
[1] Inf
> 0 / 0
[1] NaN
> c(1, NA, 2)
[1]  1 NA  2
>
```

Números especiales

La función **is.na()** se utiliza para ver los datos que son **NA** en un vector

De la misma forma que se puede utilizar **is.nan()** para los datos **NaN**

```
>
> x <- c(1, NA, 0/0)
> x
[1]   1  NA NaN
> is.na(x)
[1] FALSE  TRUE  TRUE
> is.nan(x)
[1] FALSE FALSE  TRUE
>
```

Eliminado valores NA de un vector

Observa cómo se puede utilizar la función **is.na()** para eliminar valores NA de un vector

```
> x <- c(1, 2, NA, 4, NA, 5)
> x
[1]  1  2 NA  4 NA  5
> bad <- is.na(x)
> bad
[1] FALSE FALSE  TRUE FALSE  TRUE FALSE
> x[!bad]
[1] 1 2 4 5
>
```

**Recuerda: selección de los elementos vector, haciendo uso de un vector lógico*

EJERCICIO 3 – Vectores

1. Crea un vector y asígnale los números del 1 al 10
2. Súmale 2
3. Comprueba el resultado
4. Crear otro vector con los números pares del 1 al 30
5. Comprueba su tamaño
6. Selecciona el primer y último elemento, a la vez
7. Reemplaza el primer elemento con el número 99
8. Crea un vector con los números coincidentes entre el primer y segundo vector, y ordénalos de forma decreciente

Matrices

Las matrices son un conjunto de vectores del mismo tamaño

Su dimensión viene determinada por el número de filas y columnas

$$\boldsymbol{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & & & A_{2n} \\ \vdots & & & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix}$$

```
> m <- matrix(1:6, nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
> class(m)
[1] "matrix"
>
> dim(m)
[1] 2 3
```

Seleccionando elementos en una matriz

Para seleccionar elementos en una matriz, hay que especificar la fila y la columna de los mismos: **matriz[fila, columna]**

```
> m <- matrix(11:16,2,3)
>
> m
     [,1] [,2] [,3]
[1,]   11   13   15
[2,]   12   14   16
>
> m[1,2]
[1] 13
>
```

```
> m[1,2:3]
[1] 13 15
>
> m[1:2,3]
[1] 15 16
>
> m[2]
[1] 12
>
> m[-2]
[1] 11 13 14 15 16
>
```

Seleccionando elementos en una matriz

Es posible realizar cálculos en matrices

```
> mat <- matrix(c(1,2,3,4), nrow = 2, ncol = 2, byrow = TRUE)
> mat
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> class(mat)
[1] "matrix"
> mat + mat
     [,1] [,2]
[1,]    2    4
[2,]    6    8
> mat * mat
     [,1] [,2]
[1,]    1    4
[2,]    9   16
> mat %*% mat # Matrix multiplication
     [,1] [,2]
[1,]    7   10
[2,]   15   22
```

El parámetro **byrow** = TRUE, indica que la matriz se llena por filas, por defecto es FALSE, es decir, que se va rellenando por columnas

Dos matrices se pueden **multiplicar** cuando el número de columnas de la primera es igual al número de filas de la segunda (2x3 - 3x2)

Cálculos en matrices

Se utiliza **t()** para transponer una matriz

```
> m=matrix(1:12,3,4)
>
> m
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
>
> t(m)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

Uniendo varios vectores en una matriz

Las funciones rbind() y cbind() permiten unir varios vectores para formar una matriz

```
> cbind(1:10, 101:110)
      [,1] [,2]
 [1,]    1  101
 [2,]    2  102
 [3,]    3  103
 [4,]    4  104
 [5,]    5  105
 [6,]    6  106
 [7,]    7  107
 [8,]    8  108
 [9,]    9  109
[10,]   10  110
>
```

```
> rbind(1:5, 101:105, rep(100, 5))
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]  101  102  103  104  105
[3,]  100  100  100  100  100
>
```

EJERCICIO 4 – Matrices

1. Crea tres vectores numéricos de cinco elementos
2. Úsalos para crear una matriz
3. Comprueba su tipo y dimensiones
4. Súmale uno
5. Comprueba el resultado
6. Selecciona uno o varios de sus elementos

Listas

Las listas es una clase especial de vector que pueden contener elementos de distinto tipo y tamaño

```
> x <- list(1, "a", TRUE, 1 + 4i)
> x
[[1]]
[1] 1

[[2]]
[1] "a"

[[3]]
[1] TRUE

[[4]]
[1] 1+4i

> class(x)
[1] "list"
```

Seleccionando elementos en una lista

Para seleccionar elementos en una lista se especifica un elemento mediante **[[n]]**

```
> l <- list(1:4, c("a","b"))
> l
[[1]]
[1] 1 2 3 4

[[2]]
[1] "a" "b"

> l[[1]]
[1] 1 2 3 4
> l[[2]]
[1] "a" "b"
> l[[c(1,3)]]
[1] 3
> l[[c(2,1)]]
[1] "a"
>
```

Listas con nombre

Como en el caso de los vectores, es posible dar un nombre a cada elemento de la lista

Se usa el símbolo $ para acceder al elemento a través del nombre

```
> l <- list("Numeros" = 1:3, "Letras" = c("a","b","c"))
> l
$Numeros
[1] 1 2 3

$Letras
[1] "a" "b" "c"

> |
```

Convirtiendo listas en vectores

Una lista se puede convertir en un vector mediante la función **unlist()**

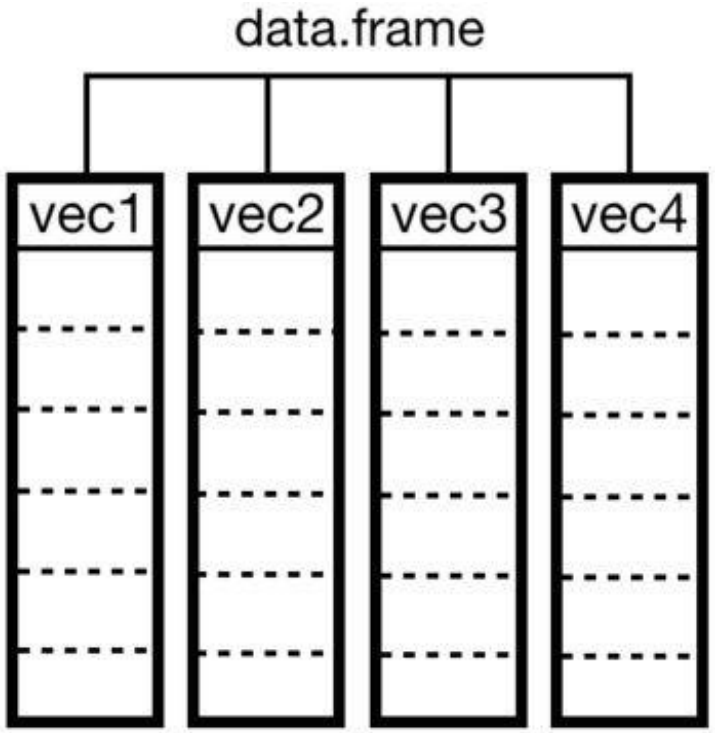
```
> l <- list(1:3, 5:8)
> l
[[1]]
[1] 1 2 3

[[2]]
[1] 5 6 7 8

> unlist(l)
[1] 1 2 3 5 6 7 8
```

EJERCICIO 5 – Listas

1. Crea una lista con valores numéricos, letras y lógicos
2. Comprueba su tipo
3. Da un nombre a cada uno de los elementos de las lista, por ejemplo: números, letras y lógicos
4. Haz una selección por nombre de elementos
5. Quítale los nombres a la lista
6. Convierte la lista en un vector y comprueba lo que ha pasado

DataFrames

Es una lista de vectores de igual tamaño y tipos distintos, que R visualiza como una tabla.



DataFrames

Se crean con la función **data.frame()**

Cada argumento se convierte es una columna

```
> Nombre <- c("Lucía","Antonio","Juan","María","Pedro")
> Edad <- c(23,34,56,65,22)
> Suscrito <- c(T,F,F,T,F)
> df <- data.frame(Nombre,Edad,Suscrito)
>
```

Obtener columnas de un DataFrame

El operador **$** se utiliza para obtener todos los valores de una columna

```
> df$Nombre
[1] Lucía   Antonio Juan    María   Pedro
Levels: Antonio Juan Lucía María Pedro
> df$Edad
[1] 23 34 56 65 22
> df$Suscrito
[1]  TRUE FALSE FALSE  TRUE FALSE
>
```

¿Qué devolverá la sentencia > *df$Nombre[2]* ?

Crear columnas de un DataFrame

Este operador se utiliza también para crear columnas nuevas …

```
> df$id <- 1:5
> df
   Nombre Edad Suscrito id
1   Lucía   23     TRUE  1
2 Antonio   34    FALSE  2
3    Juan   56    FALSE  3
4   María   65     TRUE  4
5   Pedro   22    FALSE  5
> 
```

Borrar columnas de un DataFrame

.. o borrarlas

```
> df$id <- NULL
> df
   Nombre Edad Suscrito
1   Lucía   23     TRUE
2 Antonio   34    FALSE
3    Juan   56    FALSE
4   María   65     TRUE
5   Pedro   22    FALSE
>
```

Para borrar un dataframe se utiliza el comando **rm()**

rm(nombre_del_dataframe)

Inserta filas en un DataFrame

Como insertar una fila en un dataframe

```
> df
   Nombre Edad Suscrito
1   Lucía   23     TRUE
2 Antonio   34    FALSE
3    Juan   56    FALSE
4   María   65     TRUE
5   Pedro   22    FALSE
> df_insertar <- data.frame(Nombre="Pepe",Edad=30,Suscrito=T)
> df <- rbind(df,df_insertar)
> df
   Nombre Edad Suscrito
1   Lucía   23     TRUE
2 Antonio   34    FALSE
3    Juan   56    FALSE
4   María   65     TRUE
5   Pedro   22    FALSE
6    Pepe   30     TRUE
>
```

Nombres de un DataFrame

La función **names()** devuelve el nombre de las columnas …
… y la función **row.names()** el nombre de las filas

```
> df
   Nombre Edad Suscrito
1   Lucía   23     TRUE
2 Antonio   34    FALSE
3    Juan   56    FALSE
4   María   65     TRUE
5   Pedro   22    FALSE
6    Pepe   30     TRUE
> names(df)
[1] "Nombre"   "Edad"     "Suscrito"
> row.names(df)
[1] "1" "2" "3" "4" "5" "6"
> 
```

Seleccionando columnas

En R es posible seleccionar columnas a través de su nombre o su número de orden

```
> df["Nombre"]
   Nombre
1   Lucía
2 Antonio
3    Juan
4   María
5   Pedro
6    Pepe
> df[1]
   Nombre
1   Lucía
2 Antonio
3    Juan
4   María
5   Pedro
6    Pepe
>
```

```
> df[c("Nombre","Suscrito")]
   Nombre Suscrito
1   Lucía     TRUE
2 Antonio    FALSE
3    Juan    FALSE
4   María     TRUE
5   Pedro    FALSE
6    Pepe     TRUE
> df[c(1,3)]
   Nombre Suscrito
1   Lucía     TRUE
2 Antonio    FALSE
3    Juan    FALSE
4   María     TRUE
5   Pedro    FALSE
6    Pepe     TRUE
>
```

Filtrando filas en un DataFrame

Y también las filas

```
> df[1:3,]
   Nombre Edad Suscrito
1   Lucía   23      TRUE
2 Antonio   34     FALSE
3    Juan   56     FALSE
> df[1:3,c("Nombre","Edad")]
   Nombre Edad
1   Lucía   23
2 Antonio   34
3    Juan   56
>
```

Filtrando filas en un DataFrame

También se puede utilizar valores lógicos para filtrar las filas de un DataFrame…

```
> df
   Nombre Edad Suscrito
1   Lucía   23     TRUE
2 Antonio   34    FALSE
3    Juan   56    FALSE
4   María   65     TRUE
5   Pedro   22    FALSE
6    Pepe   30     TRUE
> df[c(T,T,F,F,T,T),]
   Nombre Edad Suscrito
1   Lucía   23     TRUE
2 Antonio   34    FALSE
5   Pedro   22    FALSE
6    Pepe   30     TRUE
>
```

Filtrando filas con una condición

… por lo que se pueden utilizar condiciones que devuelvan vectores de valores lógicos

```
> df
   Nombre Edad Suscrito
1   Lucía   23     TRUE
2 Antonio   34    FALSE
3    Juan   56    FALSE
4   María   65     TRUE
5   Pedro   22    FALSE
6    Pepe   30     TRUE
> df[df$Edad>50,]
  Nombre Edad Suscrito
3   Juan   56    FALSE
4  María   65     TRUE
> df[df$Edad>50 & df$Suscrito,]
  Nombre Edad Suscrito
4  María   65     TRUE
> 
```

Filtrando filas y columnas

Es posible filtrar al mismo tiempo filas y columnas

```
> df[df$Edad>50, c("Nombre","Suscrito")]
  Nombre Suscrito
3   Juan    FALSE
4  María     TRUE
>
```

EJERCICIO 6 – DataFrames

1. Crea una dataframe con los siguientes valores:

Nombre = Lucía,Antonio, Juan, María y Pedro

Edad = 23, 34, 56, 65 y 22

Suscrito = T, F, F, T y F

1. Añade el campo id con los números del 1 al 5
2. Selecciona los registros cuya edad es mayor de 25 años
3. Selecciona los registros mayores de 25 años y que están suscritos

Manejo de datos - dplyr

Se usa para explorar y manipular datos

Trabaja muy bien con datos estructurados (tidyr)

Código muy fácil de aprender y manejar, similar al SQL

Muy rápido

Funciones más comunes:

- Select => Seleccionar conjunto de columnas
- Filter => Seleccionar conjunto de filas
- Arrange => Ordenar filas
- Mutate => Crear nuevas columnas
- Group_by => Para agrupamientos
- Summarize => Operaciones de agregación
- Join => Combinar datos de varias tablas

Select


```
head(msleep)
```

name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt
Cheetah	Acinonyx	carni	Carnivora	lc	12.1	NA	NA	11.9	NA	50.000
Owl monkey	Aotus	omni	Primates	<NA>	17.0	1.8	NA	7.0	0.01550	0.480
Mountain beaver	Aplodontia	herbi	Rodentia	nt	14.4	2.4	NA	9.6	NA	1.350
Greater short-tailed shrew	Blarina	omni	Soricomorpha	lc	14.9	2.3	0.1333333	9.1	0.00029	0.019
Cow	Bos	herbi	Artiodactyla	domesticated	4.0	0.7	0.6666667	20.0	0.42300	600.000
Three-toed sloth	Bradypus	herbi	Pilosa	<NA>	14.4	2.2	0.7666667	9.6	NA	3.850

```
head(select(msleep, name, sleep_total))
```

name	sleep_total
Cheetah	12.1
Owl monkey	17.0
Mountain beaver	14.4
Greater short-tailed shrew	14.9
Cow	4.0
Three-toed sloth	14.4

```
head(select(msleep, -name))
```

genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt
Acinonyx	carni	Carnivora	lc	12.1	NA	NA	11.9	NA	50.000
Aotus	omni	Primates	<NA>	17.0	1.8	NA	7.0	0.01550	0.480
Aplodontia	herbi	Rodentia	nt	14.4	2.4	NA	9.6	NA	1.350
Blarina	omni	Soricomorpha	lc	14.9	2.3	0.1333333	9.1	0.00029	0.019
Bos	herbi	Artiodactyla	domesticated	4.0	0.7	0.6666667	20.0	0.42300	600.000
Bradypus	herbi	Pilosa	<NA>	14.4	2.2	0.7666667	9.6	NA	3.850

Filter


```
head(msleep)
```

name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt
Cheetah	Acinonyx	carni	Carnivora	lc	12.1	NA	NA	11.9	NA	50.000
Owl monkey	Aotus	omni	Primates	<NA>	17.0	1.8	NA	7.0	0.01550	0.480
Mountain beaver	Aplodontia	herbi	Rodentia	nt	14.4	2.4	NA	9.6	NA	1.350
Greater short-tailed shrew	Blarina	omni	Soricomorpha	lc	14.9	2.3	0.1333333	9.1	0.00029	0.019
Cow	Bos	herbi	Artiodactyla	domesticated	4.0	0.7	0.6666667	20.0	0.42300	600.000
Three-toed sloth	Bradypus	herbi	Pilosa	<NA>	14.4	2.2	0.7666667	9.6	NA	3.850

```
head(filter(msleep, sleep_total >= 16))
```

name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt
Owl monkey	Aotus	omni	Primates	<NA>	17.0	1.8	NA	7.0	0.01550	0.480
Long-nosed armadillo	Dasypus	carni	Cingulata	lc	17.4	3.1	0.3833333	6.6	0.01080	3.500
North American Opossum	Didelphis	omni	Didelphimorphia	lc	18.0	4.9	0.3333333	6.0	0.00630	1.700
Big brown bat	Eptesicus	insecti	Chiroptera	lc	19.7	3.9	0.1166667	4.3	0.00030	0.023
Thick-tailed opposum	Lutreolina	carni	Didelphimorphia	lc	19.4	6.6	NA	4.6	NA	0.370
Little brown bat	Myotis	insecti	Chiroptera	<NA>	19.9	2.0	0.2000000	4.1	0.00025	0.010

```
head(filter(msleep, sleep_total >= 16, bodywt >= 1))
```

name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt
Long-nosed armadillo	Dasypus	carni	Cingulata	lc	17.4	3.1	0.3833333	6.6	0.0108	3.5
North American Opossum	Didelphis	omni	Didelphimorphia	lc	18.0	4.9	0.3333333	6.0	0.0063	1.7
Giant armadillo	Priodontes	insecti	Cingulata	en	18.1	6.1	NA	5.9	0.0810	60.0

Arrange


```
head(msleep)
```

name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt
Cheetah	Acinonyx	carni	Carnivora	lc	12.1	NA	NA	11.9	NA	50.000
Owl monkey	Aotus	omni	Primates	<NA>	17.0	1.8	NA	7.0	0.01550	0.480
Mountain beaver	Aplodontia	herbi	Rodentia	nt	14.4	2.4	NA	9.6	NA	1.350
Greater short-tailed shrew	Blarina	omni	Soricomorpha	lc	14.9	2.3	0.1333333	9.1	0.00029	0.019
Cow	Bos	herbi	Artiodactyla	domesticated	4.0	0.7	0.6666667	20.0	0.42300	600.000
Three-toed sloth	Bradypus	herbi	Pilosa	<NA>	14.4	2.2	0.7666667	9.6	NA	3.850

```
head(arrange(msleep, order))
```

name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt
Tenrec	Tenrec	omni	Afrosoricida	<NA>	15.6	2.3	NA	8.4	0.0026	0.900
Cow	Bos	herbi	Artiodactyla	domesticated	4.0	0.7	0.6666667	20.0	0.4230	600.000
Roe deer	Capreolus	herbi	Artiodactyla	lc	3.0	NA	NA	21.0	0.0982	14.800
Goat	Capri	herbi	Artiodactyla	lc	5.3	0.6	NA	18.7	0.1150	33.500
Giraffe	Giraffa	herbi	Artiodactyla	cd	1.9	0.4	NA	22.1	NA	899.995
Sheep	Ovis	herbi	Artiodactyla	domesticated	3.8	0.6	NA	20.2	0.1750	55.500

```
head(arrange(msleep, order, awake))
```

name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt
Tenrec	Tenrec	omni	Afrosoricida	<NA>	15.6	2.3	NA	8.4	0.0026	0.90
Pig	Sus	omni	Artiodactyla	domesticated	9.1	2.4	0.5000000	14.9	0.1800	86.25
Goat	Capri	herbi	Artiodactyla	lc	5.3	0.6	NA	18.7	0.1150	33.50
Cow	Bos	herbi	Artiodactyla	domesticated	4.0	0.7	0.6666667	20.0	0.4230	600.00
Sheep	Ovis	herbi	Artiodactyla	domesticated	3.8	0.6	NA	20.2	0.1750	55.50
Roe deer	Capreolus	herbi	Artiodactyla	lc	3.0	NA	NA	21.0	0.0982	14.80

Mutate


```
head(msleep)
```

name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt
Cheetah	Acinonyx	carni	Carnivora	lc	12.1	NA	NA	11.9	NA	50.000
Owl monkey	Aotus	omni	Primates	<NA>	17.0	1.8	NA	7.0	0.01550	0.480
Mountain beaver	Aplodontia	herbi	Rodentia	nt	14.4	2.4	NA	9.6	NA	1.350
Greater short-tailed shrew	Blarina	omni	Soricomorpha	lc	14.9	2.3	0.1333333	9.1	0.00029	0.019
Cow	Bos	herbi	Artiodactyla	domesticated	4.0	0.7	0.6666667	20.0	0.42300	600.000
Three-toed sloth	Bradypus	herbi	Pilosa	<NA>	14.4	2.2	0.7666667	9.6	NA	3.850

```
head(mutate(msleep, rem_proportion = sleep_rem / sleep_total))
```

name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt	rem_proportion
Cheetah	Acinonyx	carni	Carnivora	lc	12.1	NA	NA	11.9	NA	50.000	NA
Owl monkey	Aotus	omni	Primates	<NA>	17.0	1.8	NA	7.0	0.01550	0.480	0.1058824
Mountain beaver	Aplodontia	herbi	Rodentia	nt	14.4	2.4	NA	9.6	NA	1.350	0.1666667
Greater short-tailed shrew	Blarina	omni	Soricomorpha	lc	14.9	2.3	0.1333333	9.1	0.00029	0.019	0.1543624
Cow	Bos	herbi	Artiodactyla	domesticated	4.0	0.7	0.6666667	20.0	0.42300	600.000	0.1750000
Three-toed sloth	Bradypus	herbi	Pilosa	<NA>	14.4	2.2	0.7666667	9.6	NA	3.850	0.1527778

```
head(mutate(msleep, rem_proportion = sleep_rem / sleep_total, bodywt_grams = bodywt * 1000))
```

name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt	rem_proportion
Cheetah	Acinonyx	carni	Carnivora	lc	12.1	NA	NA	11.9	NA	50.000	NA
Owl monkey	Aotus	omni	Primates	<NA>	17.0	1.8	NA	7.0	0.01550	0.480	0.1058824
Mountain beaver	Aplodontia	herbi	Rodentia	nt	14.4	2.4	NA	9.6	NA	1.350	0.1666667
Greater short-tailed shrew	Blarina	omni	Soricomorpha	lc	14.9	2.3	0.1333333	9.1	0.00029	0.019	0.1543624
Cow	Bos	herbi	Artiodactyla	domesticated	4.0	0.7	0.6666667	20.0	0.42300	600.000	0.1750000
Three-toed sloth	Bradypus	herbi	Pilosa	<NA>	14.4	2.2	0.7666667	9.6	NA	3.850	0.1527778

bodywt_grams
50000
480
1350
19
600000
3850

Summarise & Group By


```
summarise(msleep, avg_sleep = mean(sleep_total))
avg_sleep
 10.43373

summarise(msleep, avg_sleep = mean(sleep_total), min_sleep = min(sleep_total),  max_sleep = max(sleep_total), total = n())
avg_sleep min_sleep max_sleep total
 10.43373       1.9      19.9    83

msleep %>%
    group_by(order) %>%
    summarise(avg_sleep = mean(sleep_total),
              min_sleep = min(sleep_total),
              max_sleep = max(sleep_total),
              total = n())
A tibble: 19 x 5
           order avg_sleep min_sleep max_sleep total
          <fctr>     <dbl>     <dbl>     <dbl> <int>
    Afrosoricida 15.600000      15.6      15.6     1
    Artiodactyla  4.516667       1.9       9.1     6
       Carnivora 10.116667       3.5      15.8    12
        Cetacea  4.500000       2.7       5.6     3
      Chiroptera 19.800000      19.7      19.9     2
       Cingulata 17.750000      17.4      18.1     2
 Didelphimorphia 18.700000      18.0      19.4     2
   Diprotodontia 12.400000      11.1      13.7     2
  Erinaceomorpha 10.200000      10.1      10.3     2
      Hyracoidea  5.666667       5.3       6.3     3
      Lagomorpha  8.400000       8.4       8.4     1
     Monotremata  8.600000       8.6       8.6     1
  Perissodactyla  3.466667       2.9       4.4     3
          Pilosa 14.400000      14.4      14.4     1
        Primates 10.500000       8.0      17.0    12
     Proboscidea  3.600000       3.3       3.9     2
        Rodentia 12.468182       7.0      16.6    22
      Scandentia  8.900000       8.9       8.9     1
    Soricomorpha 11.100000       8.4      14.9     5
```

El operador pipe

El operador pipe %>%

El operador pipe , %>% , se utiliza para insertar argumentos es una función. No es una función base de R , se incorpora con paquetes como dplyr y magrittr. El operador pipe coge los elementos de la izquierda (left-hand side,LHS) y los utiliza como primer argumento de la función de la derecha (RHS). Por ejemplo:

Hide

```
iris %>%
 group_by(Species) %>%
 summarise(avg = mean(Sepal.Width)) %>%
 arrange(avg)
```

Los pipes nos permiten leer el código de izquierda a derecha, no de dentro a fuera.

El operador pipe

Ejemplo de Hadley Wickham

Hide

```
library(nycflights13)
hourly_delay <- filter(
  summarise(
    group_by(
      filter(flights, !is.na(dep_delay)),
       hour),
    delay = mean(dep_delay),
    n = n()),
  n > 10
)

hourly_delay <- flights %>%
  filter(!is.na(dep_delay)) %>%
  group_by( hour) %>%
  summarise(
    delay = mean(dep_delay),
    n = n()
  ) %>%
  filter(n > 10)
```

EJERCICIO dplyr

Ejercicio

Carga el dataset iris en R (viene con el paquete base de R, usar el comando head(iris) para visualizarlo o simplemente introducir iris para ver todo el dataframe)

1. Elige las variables Petal.Width y Species
2. Selecciona un Sepal.Length mayor que 5.0
3. Selecciona Sepal.Width menor que 8.3 y Species igual a Virginica
4. Crea una nueva variable: Sepal = Sepal.Length / Sepal.Width
5. Agrupa por Species y calcula la media de Sepal.With,

Índice

Estadística descriptiva

length() devuelve la longitud de un vector
min() el valor mínimo
max() el valor máximo

```
> ages = c(25, 22, 18, 20, 22)
> ages
[1] 25 22 18 20 22
> min(ages)
[1] 18
> max(ages)
[1] 25
> length(ages)
[1] 5
```

Estadística descriptiva

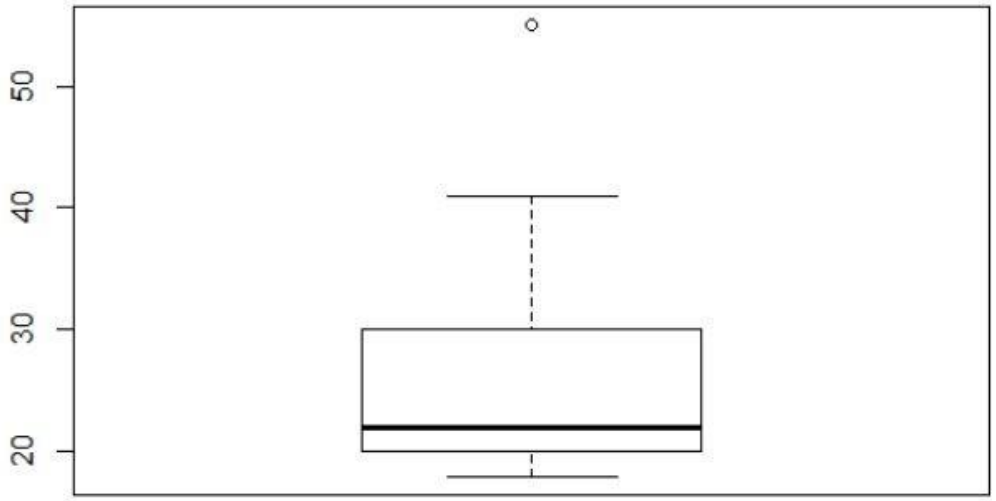
mean() devuelve el valor medio
median() la mediana
sd() la desviación típica
var() la varianza

```
> ages <- c(25, 22, 18, 20, 22)
> mean(ages)
[1] 21.4
> median(ages)
[1] 22
> sd(ages)
[1] 2.607681
> var(ages)
[1] 6.8
```

Estadística descriptiva

summary() nos muestra la distribución de datos

```
> ages <- c(19, 25, 22, 18, 20, 22, 30, 22, 55, 41)
> summary(ages)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  18.00   20.50   22.00   27.40   28.75   55.00
>
```



Distribuciones de probabilidad en R

R viene con una serie de distribuciones probabilísticas estándar

Permite generar números aleatorios según una determinada distribución

Las distribuciones sirven para modelizar fenómenos y son muy útiles en el Machine Learning

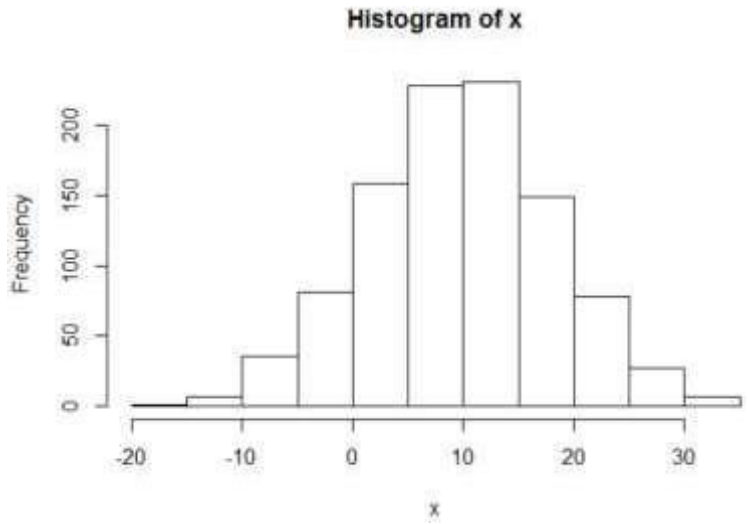
Utiliza la función **help(distributions)** para verlas

```
> help(Distributions)
>
```

Distribución normal

Para generar números aleatorios según la distribución normal se utiliza **rnorm()**

```
> x <- rnorm(1000, mean = 10, sd = 8)
> x[1:10]
 [1]  0.03286958  7.26519636 10.50557568  4.54998383  2.41240307
 [6] 14.75232535 -4.17608221 -7.72921943  5.31683759  5.43566948
>
```



Sampling

Para obtener un ejemplo de un vector de números se utiliza la función **sample()**

```
> set.seed(10)
>
> sample(1:10, 6, replace = F, prob=rep(0.1, 10))
[1] 7 4 6 9 2 5
>
> sample(1:10, 20, replace = T, prob = c(0.25,0.25,rep(0.1,8)))
 [1]  1  1  7  3  7  6  2  6  1  3  2  1  3  8 10  6  8  1  3  9
> 
```

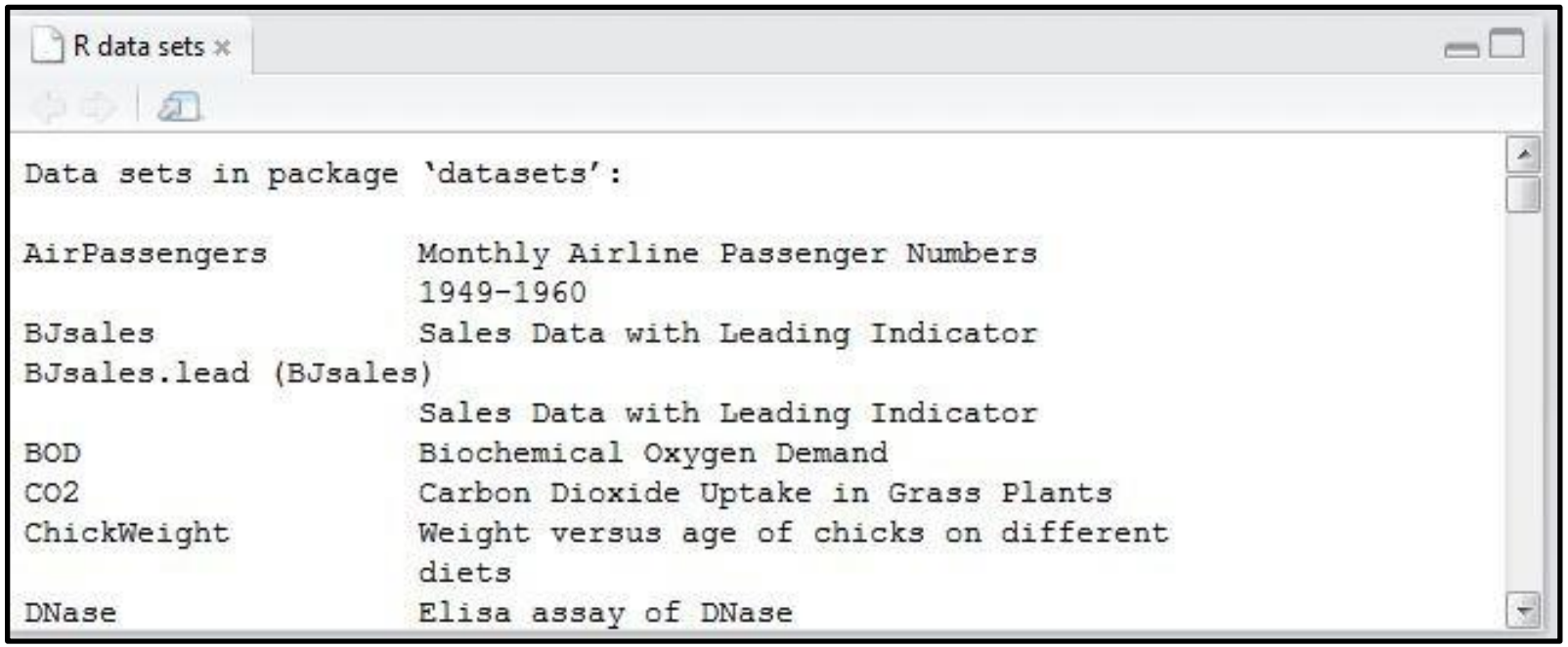
Creando DataFrames con valores aleatorios

En R es posible crear un DataFrame utilizando funciones aleatorias.

```
> data <- data.frame(categoria = rep(c("A", "B", "C"), 100),
+                    nota = sample(1:10, 100, replace = T),
+                    dato = rnorm(100, mean = 100, sd=20))
>
> data[1:10,]
   categoria nota      dato
1          A    6  91.98725
2          B    4  93.30887
3          C    5 127.35908
4          A    7 142.75534
5          B    1 110.11639
6          C    3 115.72685
7          A    3  81.95576
8          B    3 110.65794
9          C    7  87.08211
10         A    5 105.81975
>
```

Datasets de ejemplo

R viene con un conjunto de datasets de ejemplo con las que se puede experimentar



Datasets de ejemplo

Para listar los datasets se utiliza la función **data()**

Para cargarlo en memoria **data(DataSet)**

Con **rm(DataSet)** se descarga de memoria

```
> data()
> data ("airquality")
> rm ("airquality")
>
```

Índice

Explorando un DataFrame

Vamos a explorar el DataFrame "**airquality**"

Proviene de los datos de ejemplo de R

Con **?** Se obtiene una descripción completa

New York Air Quality Measurements

Description

Daily air quality measurements in New York, May to September 1973.

Usage

```
airquality
```

Format

A data frame with 154 observations on 6 variables.

Explorando un DataFrame

dim() devuelve las dimensiones del DataSets
ncol() el número de columnas y ..
nrow() el número de filas.

```
> ?airquality
> dim(airquality)
[1] 153   6
> ncol(airquality)
[1] 6
> nrow(airquality)
[1] 153
>
```

Explorando un DataFrame

names() devuelve el nombre de las columnas

```
> names(airquality)
[1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"
> 
```

Explorando un DataFrame

head() devuelve las primeras columnas …

```
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
>
```

Explorando un DataFrame

tail() devuelve las últimas columnas …

```
> tail(airquality)
    Ozone Solar.R Wind Temp Month Day
148    14      20 16.6   63     9  25
149    30     193  6.9   70     9  26
150    NA     145 13.2   77     9  27
151    14     191 14.3   75     9  28
152    18     131  8.0   76     9  29
153    20     223 11.5   68     9  30
> 
```

Explorando un DataFrame

summary() devuelve la distribución estadística de las columnas …

```
> summary(airquality)
     Ozone           Solar.R          Wind             Temp      
 Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00  
 1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00  
 Median : 31.50   Median :205.0   Median : 9.700   Median :79.00  
 Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88  
 3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00  
 Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00  
 NA's   :37       NA's   :7                                       
     Month            Day      
 Min.   :5.000   Min.   : 1.0  
 1st Qu.:6.000   1st Qu.: 8.0  
 Median :7.000   Median :16.0  
 Mean   :6.993   Mean   :15.8  
 3rd Qu.:8.000   3rd Qu.:23.0  
 Max.   :9.000   Max.   :31.0  
```

Explorando un DataFrame

str() imprime para cada columna, el tipo dato y una muestra de los valores que tiene

```
> str(airquality)
'data.frame':   153 obs. of  6 variables:
 $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
 $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
 $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
 $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
 $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
>
```

Explorando un DataFrame

La función **table()** se utiliza para que cuente los distintos valores de una columna

```
> table(airquality$Month)

 5  6  7  8  9
31 30 31 31 30
> table(airquality$Month,airquality$Day)

   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  5 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  6 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0
  7 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  8 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  9 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0
>
```

Índice

Gráficos en R – El sistema base

Sistema original de R, no es necesario instalar ningún paquete adicional

Se empieza con un lienzo vacío y se van añadiendo elementos gráficos

Es el más conveniente para análisis exploratorio de la información

```
> plot(Sepal.Length~Petal.Length, data=iris)
>
```

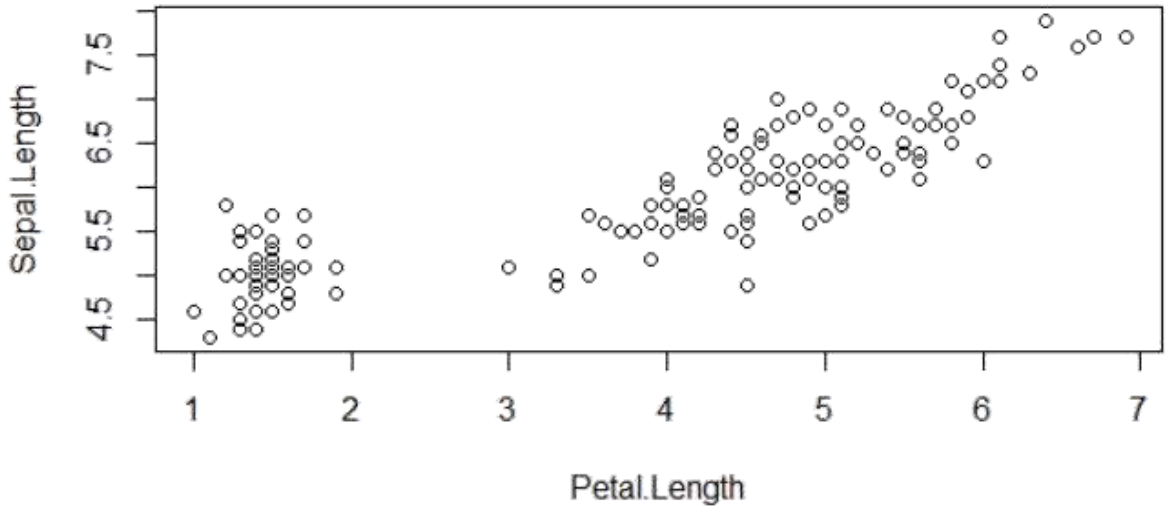


Gráfico de Dispersión o Scatter Plot

```
> plot(airquality$Temp,airquality$Ozone)
```

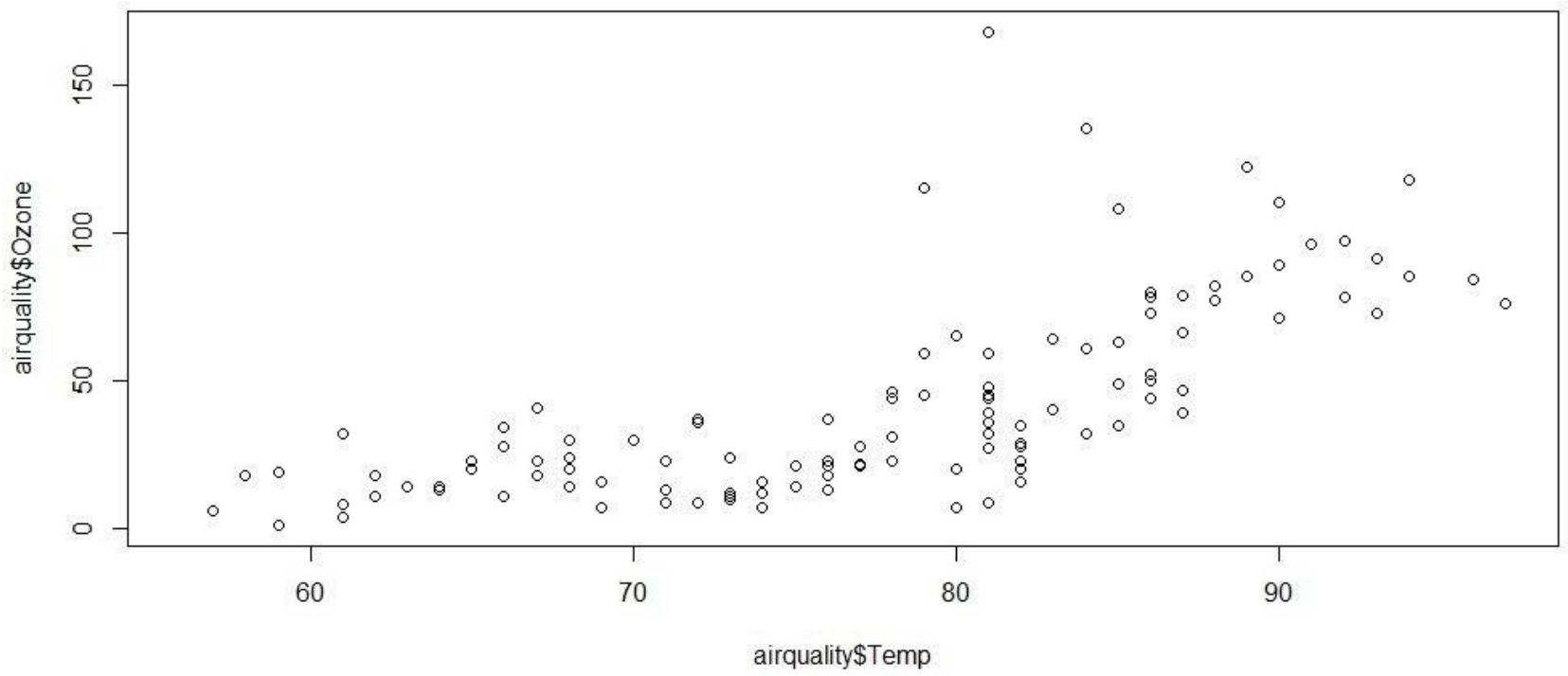
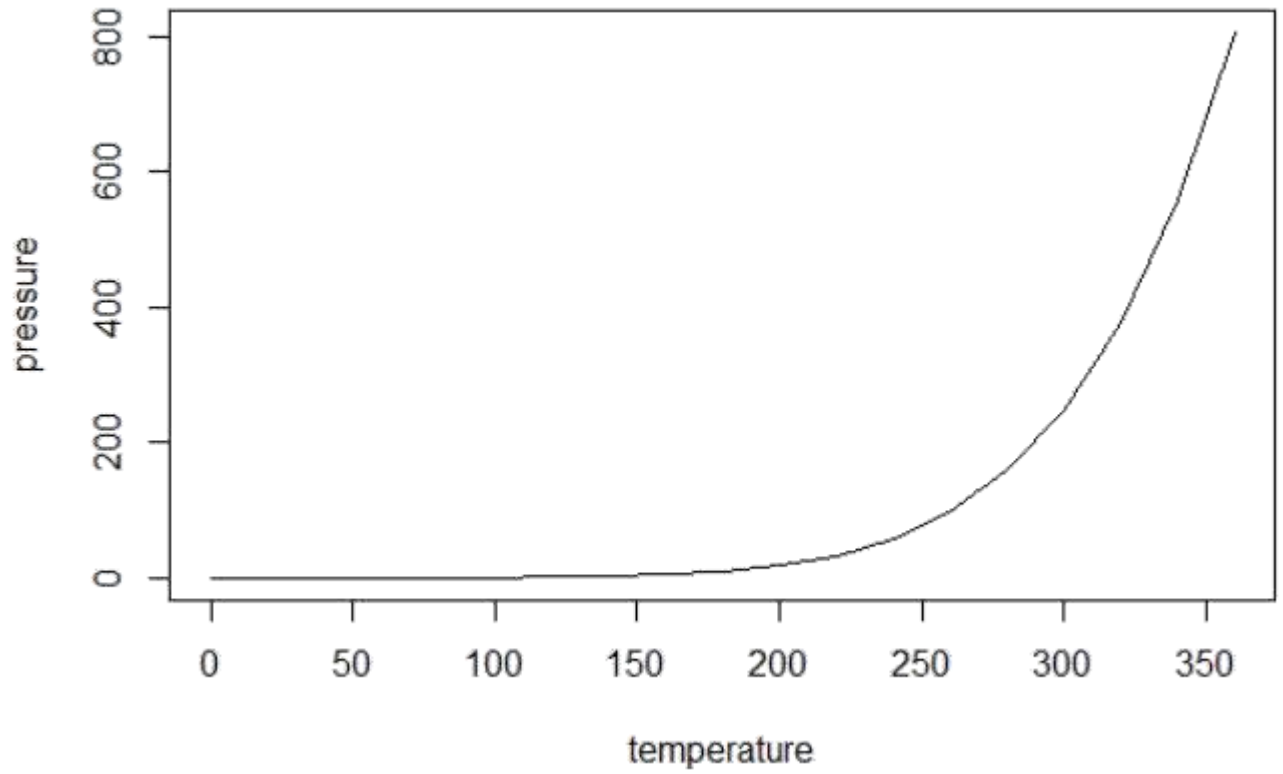


Gráfico de líneas

```
> ?pressure
> plot(pressure, type="l")
>
```



Líneas y puntos al mismo tiempo

```
> plot(pressure, type = "l")
> points(pressure$temperature, pressure$pressure)
> lines(pressure$temperature, pressure$pressure / 2, col = "red")
> points(pressure$temperature, pressure$pressure / 2, col = "red")
>
```

Pueden añadirse elementos a un gráfico

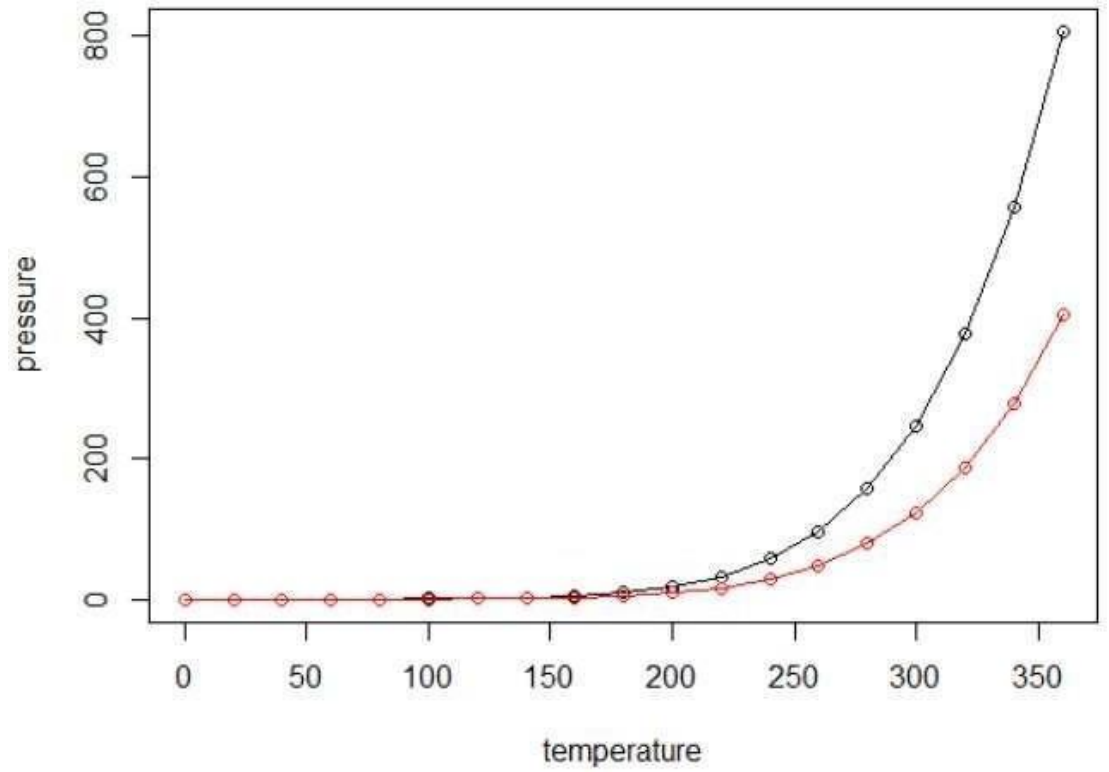
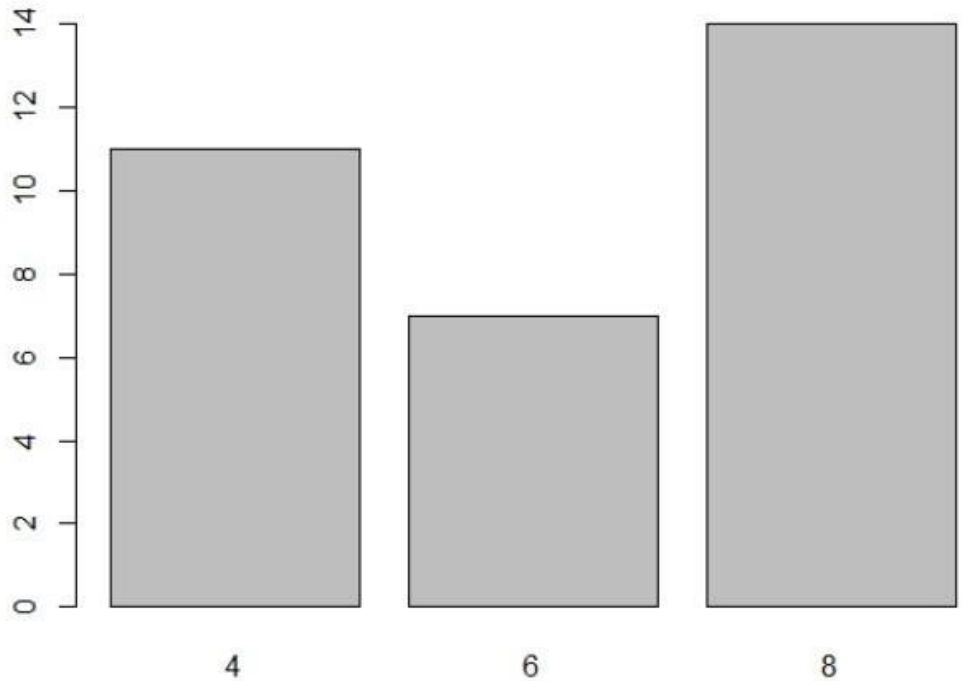


Diagrama de barras

```
> table(mtcars$cyl)

 4  6  8
11  7 14
>
> barplot(table(mtcars$cyl))
```

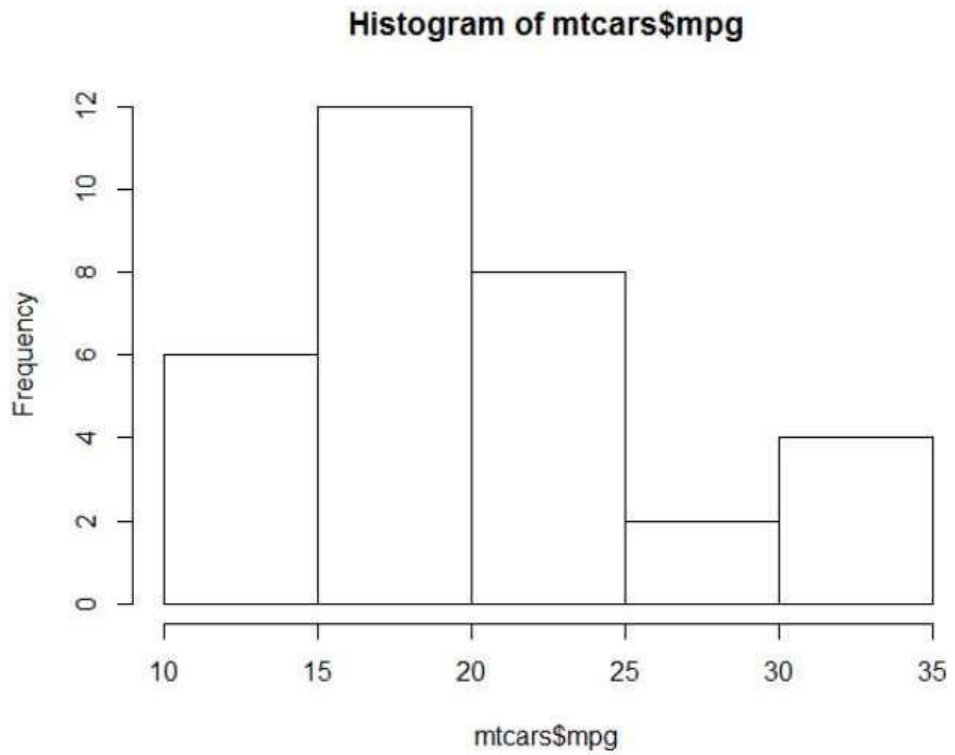
Para variables cualitativas



Histograma

```
> hist(mtcars$mpg)
> 
```

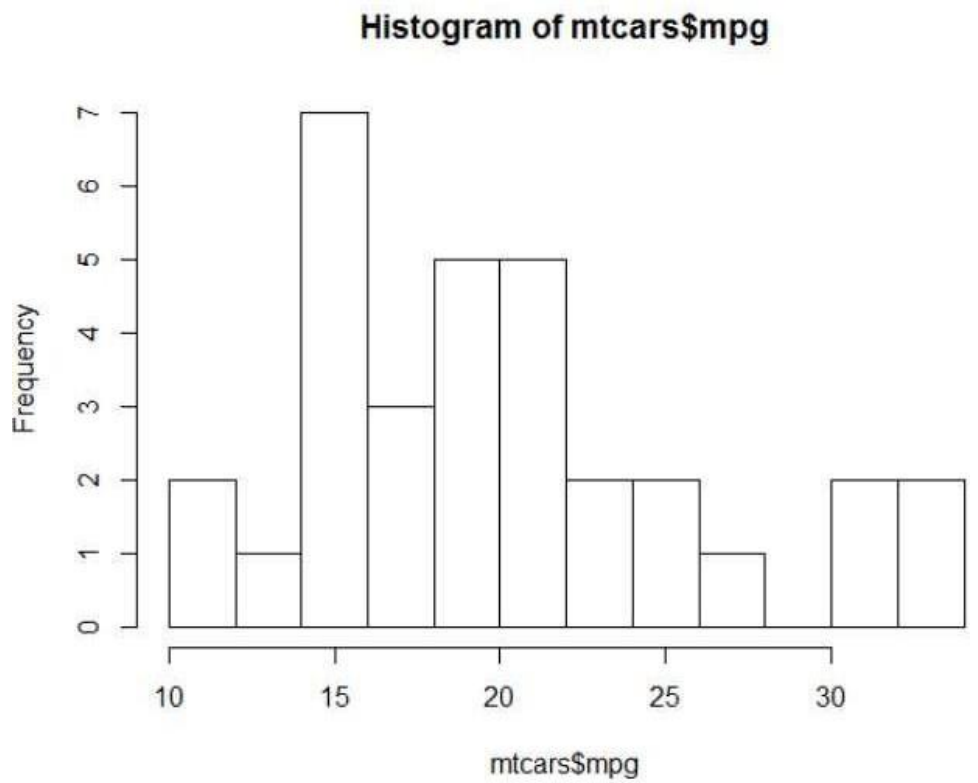
Ver la distribución de los datos



Histograma

```
> hist(mtcars$mpg, breaks = 10)
> |
```

Con la opción break, se establecen puntos de ruptura



Box Plot

```
> head(ToothGrowth)
   len supp dose
1  4.2   VC  0.5
2 11.5   VC  0.5
3  7.3   VC  0.5
4  5.8   VC  0.5
5  6.4   VC  0.5
6 10.0   VC  0.5
> boxplot(len ~ supp, data = ToothGrowth)
```

Muy útiles para ver como se distribuyen los datos y detectar valores atípicos

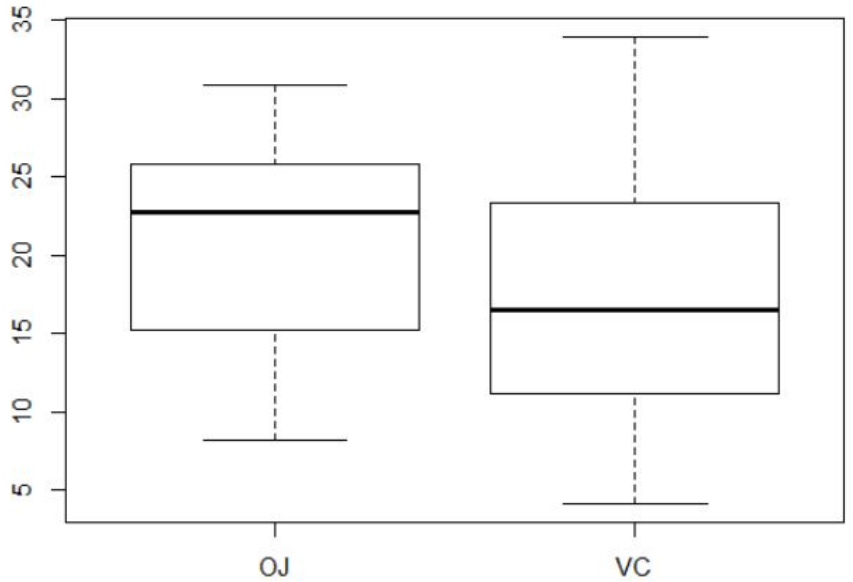
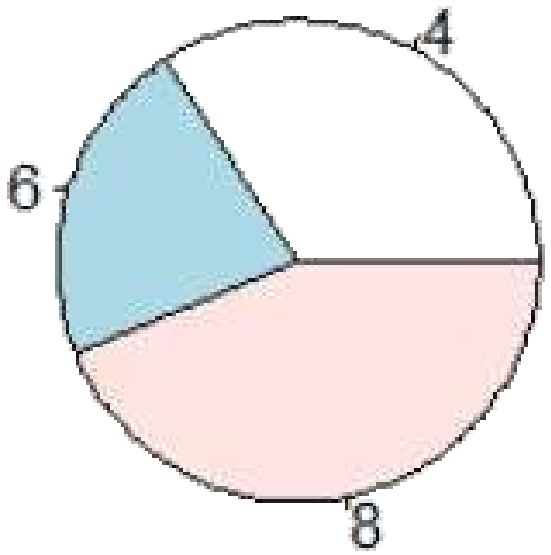


Gráfico de tarta

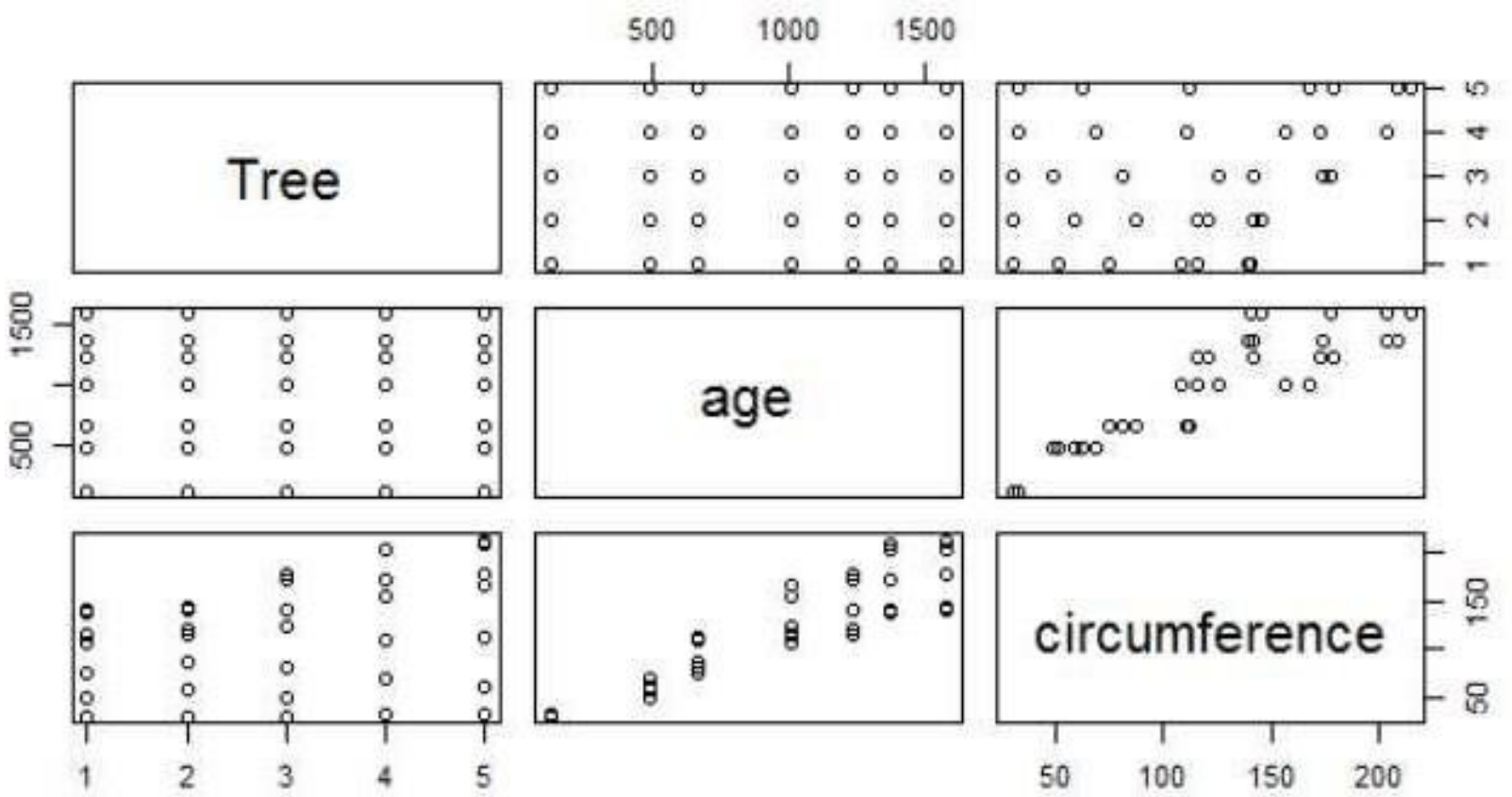
```
> cyltable<- table(mtcars$cyl)
> cyltable

 4  6  8
11  7 14
> pie(cyltable, labels = names(cyltable))
> 
```

Mostrar las proporciones de una serie de variable, deja de ser útil con más de diez valores



Todos con todos

```
> plot(Orange)
>
```



Guardar un gráfico

Por línea de comandos

```
> jpeg(filename="migrafico.jpeg")
> plot(Orange)
> dev.off()
RStudioGD
        2
>
```

```
> pdf(file="migrafico.pdf")
> plot(Orange)
> dev.off()
RStudioGD
        2
>
```

Se guarda en ubicación por defecto

Desde el propio RStudio, pestaña Plots y Export

EJERCICIO 7 – Analítica descriptiva

1. La **cantidad de zinc** (en mg/l) en 16 muestras de alimentos infantiles viene dada por:

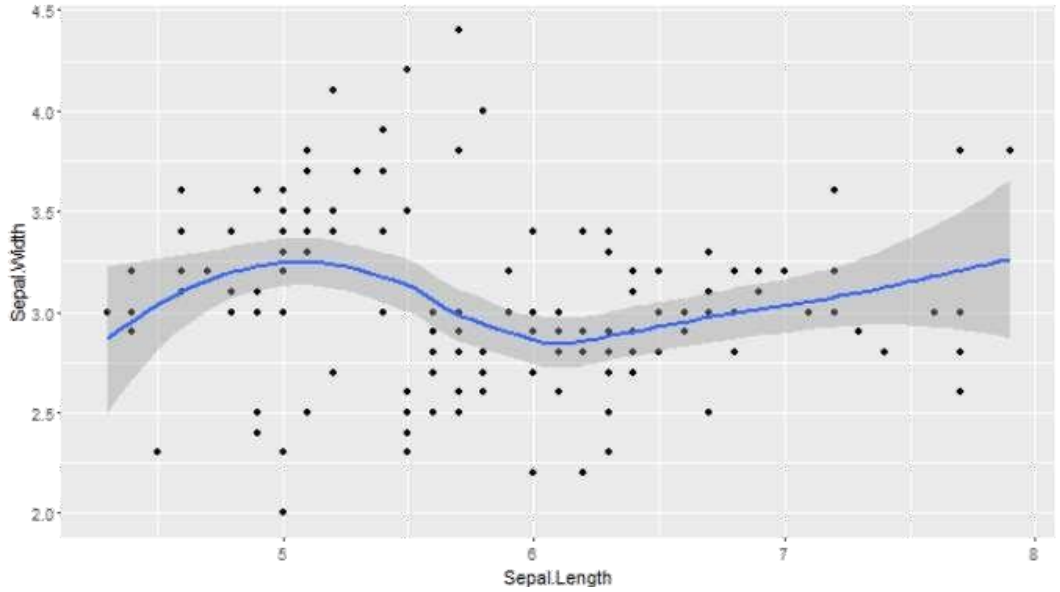
(3.0, 5.8, 5.6, 4.8, 5.1, 3.6, 5.5, 4.7, 5.7, 5.0, 5.9, 5.7, 4.4, 5.4, 4.2, 5.3)

2. Hallar media, desviación típica, mediana, cuartiles, y dibujar el box-plot e histograma.

Gráficos en R – El sistema Ggplot2

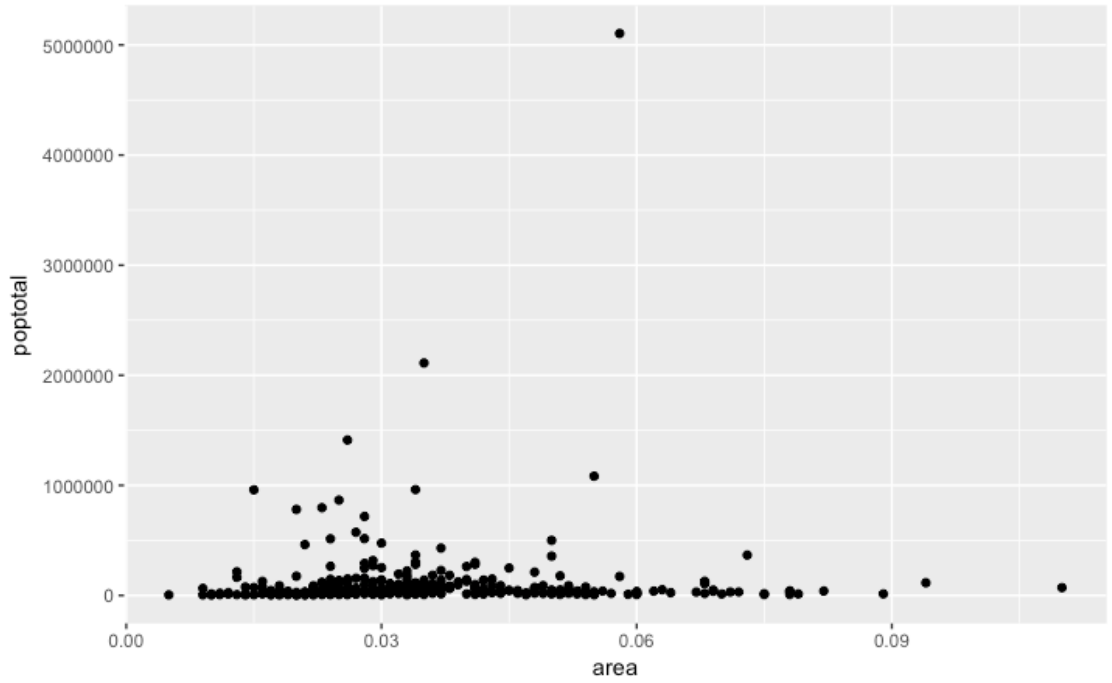
Basado en la gramática de los gráficos, proporciona un lenguaje para crear gráficas complejas de una forma más simple que los gráficos básicos

```
> ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width))  + geom_point()  + stat_smooth()
```



Ggplot

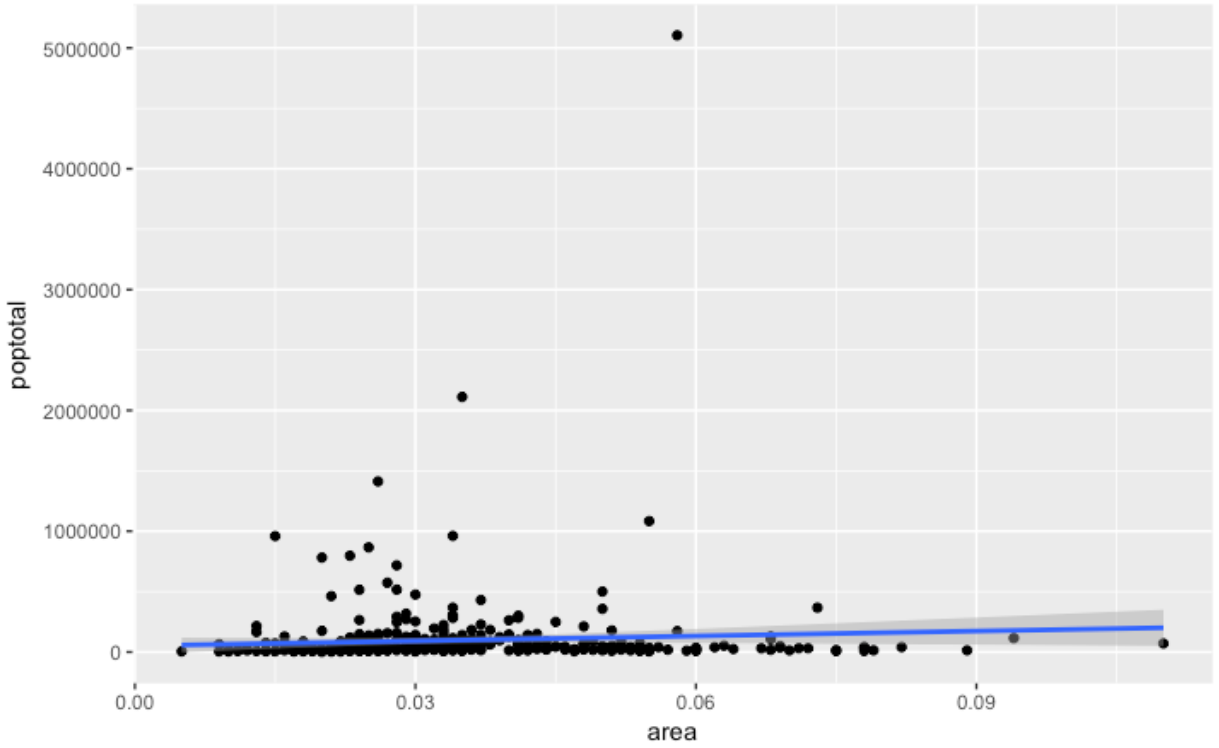
```
library(ggplot2)
ggplot(midwest, aes(x=area, y=poptotal)) + geom_point()
```



http://r-statistics.co/Complete-Ggplot2-Tutorial-Part1-With-R-Code.html

Ggplot

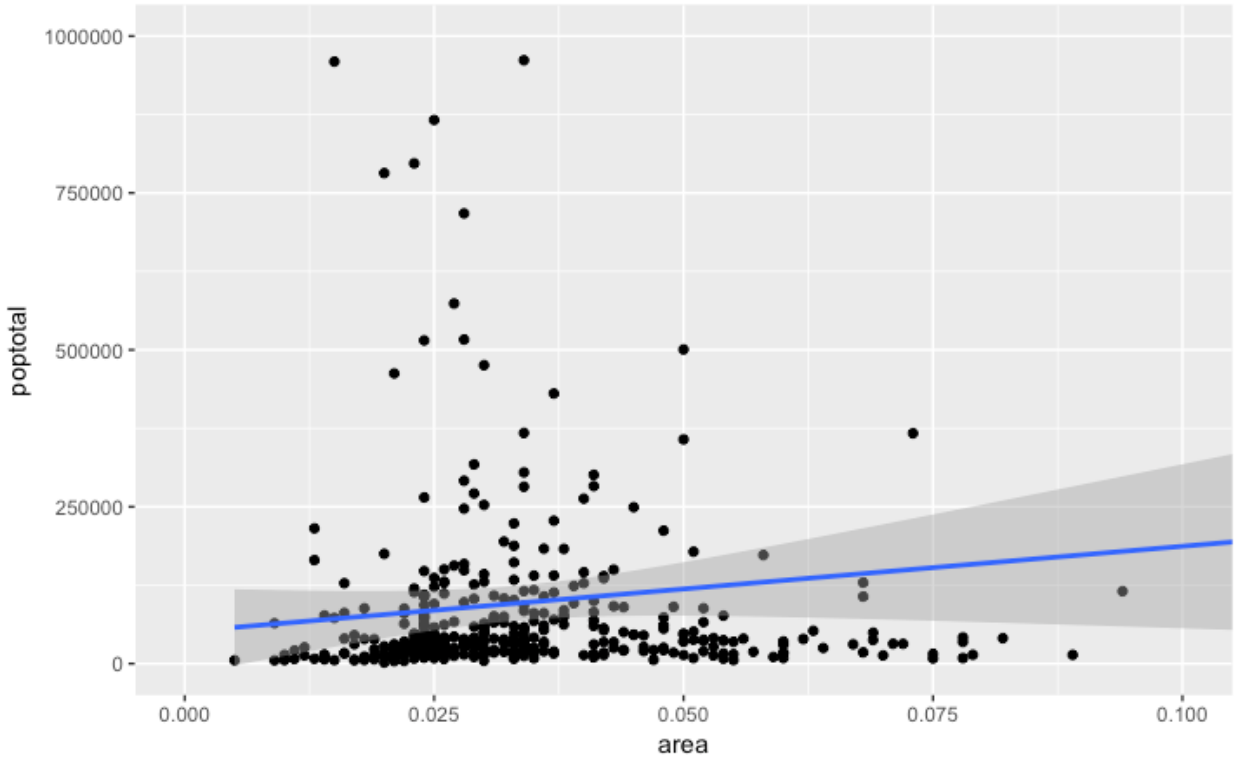
```
g <- ggplot(midwest, aes(x=area, y=poptotal)) + geom_point() +
geom_smooth(method="lm")  # set se=FALSE to turnoff confidence bands
plot(g)
```



http://r-statistics.co/Complete-Ggplot2-Tutorial-Part1-With-R-Code.html

Ggplot

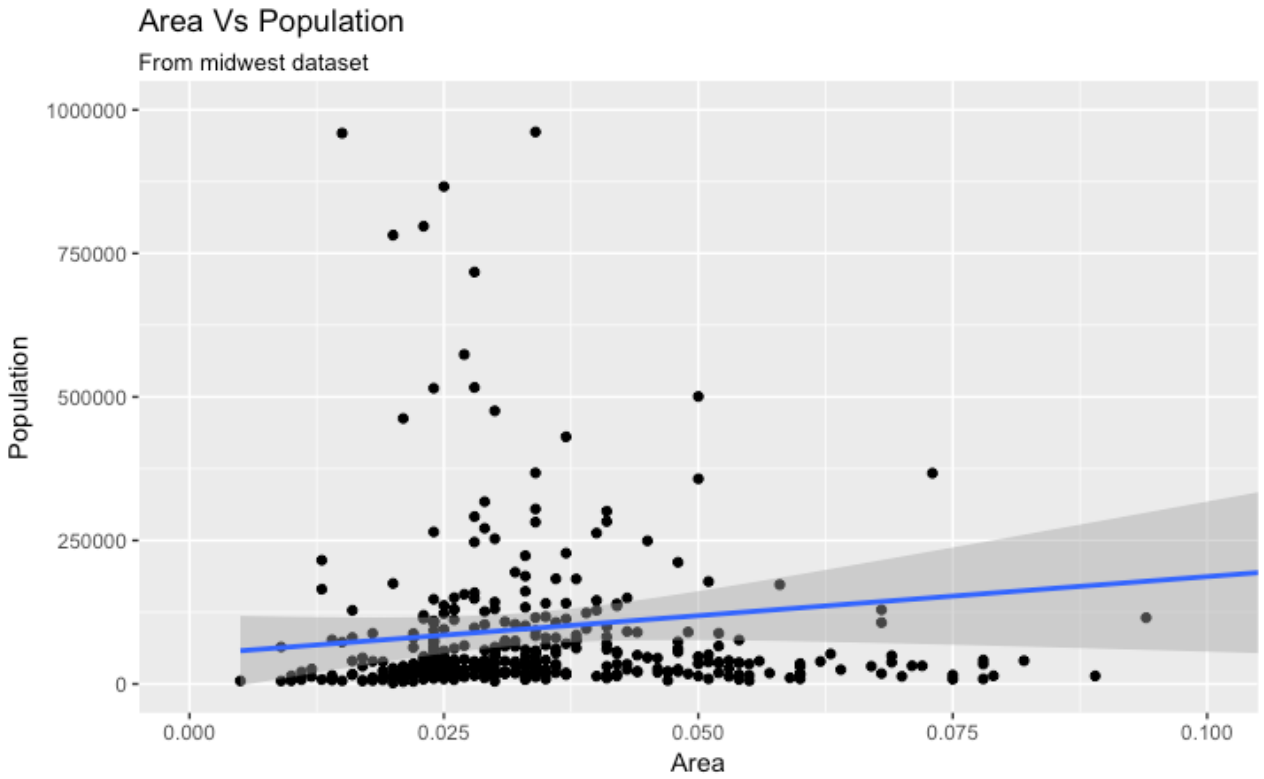
```
# Zoom in without deleting the points outside the limits.
# As a result, the line of best fit is the same as the original plot.
g1 <- g + coord_cartesian(xlim=c(0,0.1), ylim=c(0, 1000000))  # zooms in
plot(g1)
```



http://r-statistics.co/Complete-Ggplot2-Tutorial-Part1-With-R-Code.html

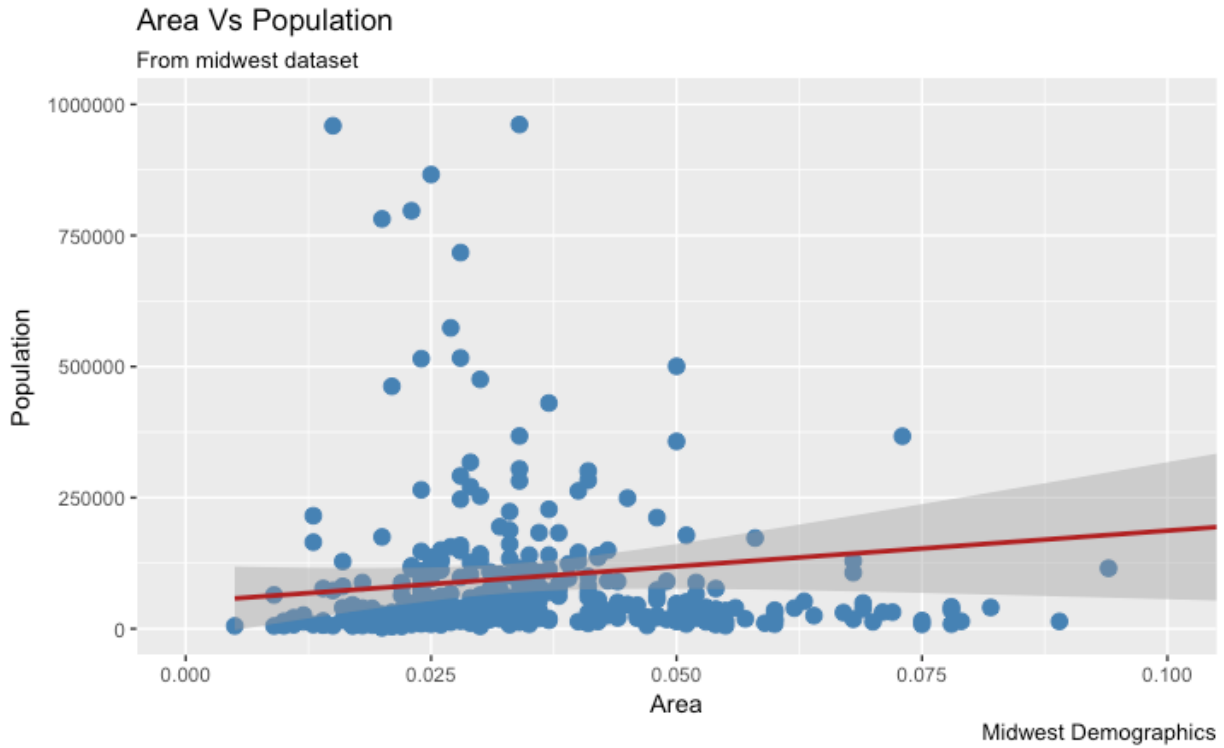
Ggplot

```
# Add Title and Labels
g1 + labs(title="Area Vs Population", subtitle="From midwest dataset", y="Population", x="Area", caption="Midwest Demographics")
# or
g1 + ggtitle("Area Vs Population", subtitle="From midwest dataset") + xlab("Area") + ylab("Population")
```



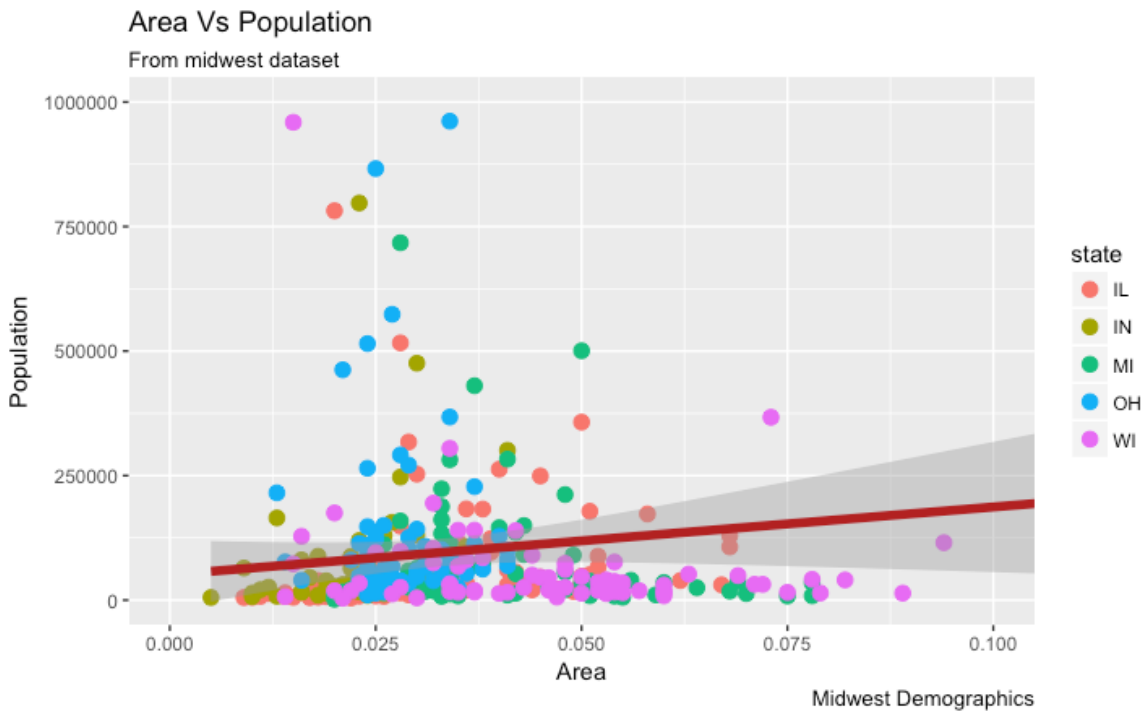
Ggplot

```
ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point(col="steelblue", size=3) +   # Set static color and size for points
  geom_smooth(method="lm", col="firebrick") +  # change the color of line
  coord_cartesian(xlim=c(0, 0.1), ylim=c(0, 1000000)) +
  labs(title="Area Vs Population", subtitle="From midwest dataset", y="Population", x="Area",
caption="Midwest Demographics")
```



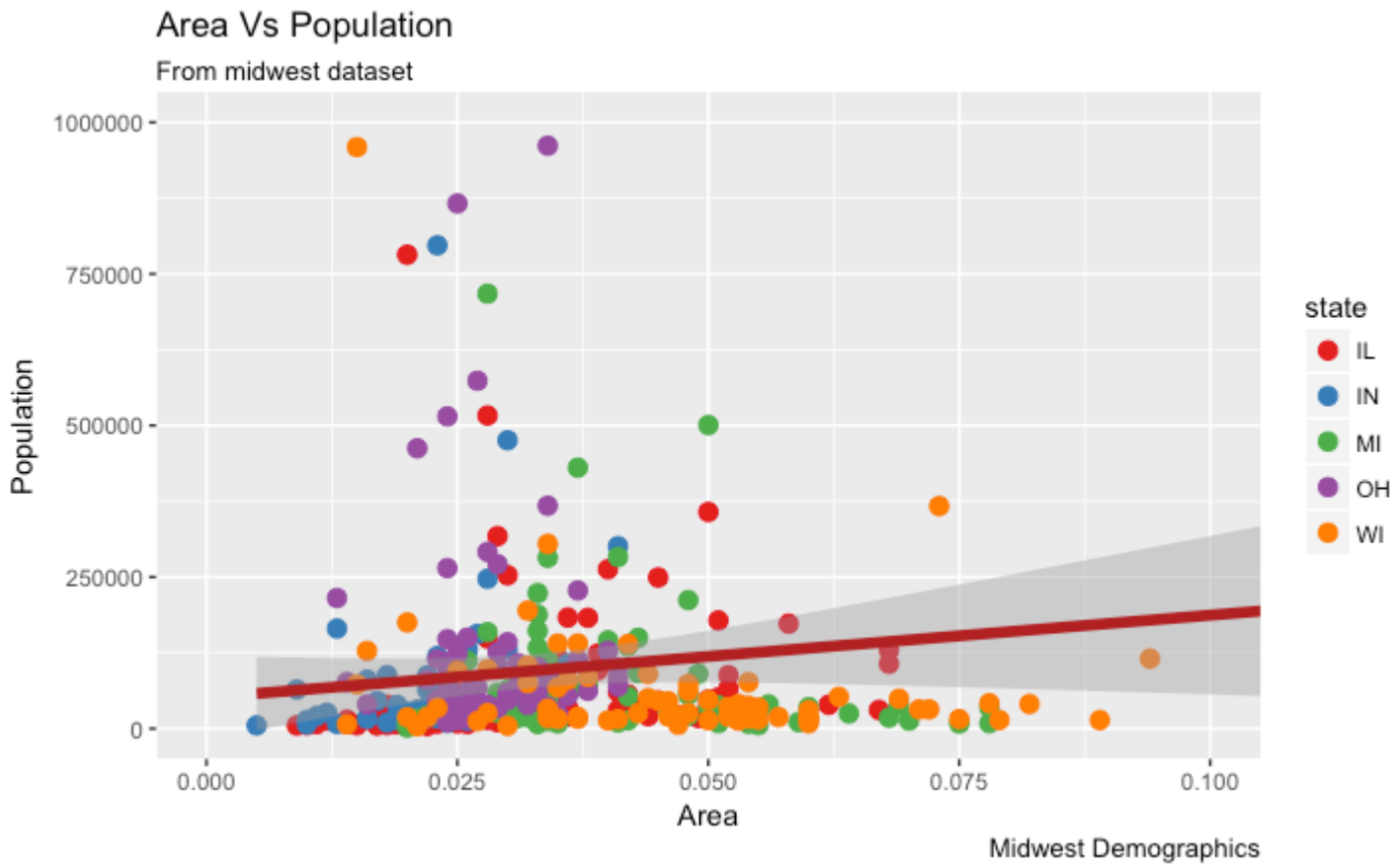
Ggplot

```
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point(aes(col=state), size=3) +  # Set color to vary based on state categories.
  geom_smooth(method="lm", col="firebrick", size=2) +
  coord_cartesian(xlim=c(0, 0.1), ylim=c(0, 1000000)) +
  labs(title="Area Vs Population", subtitle="From midwest dataset", y="Population", x="Area",
caption="Midwest Demographics")
plot(gg)
```



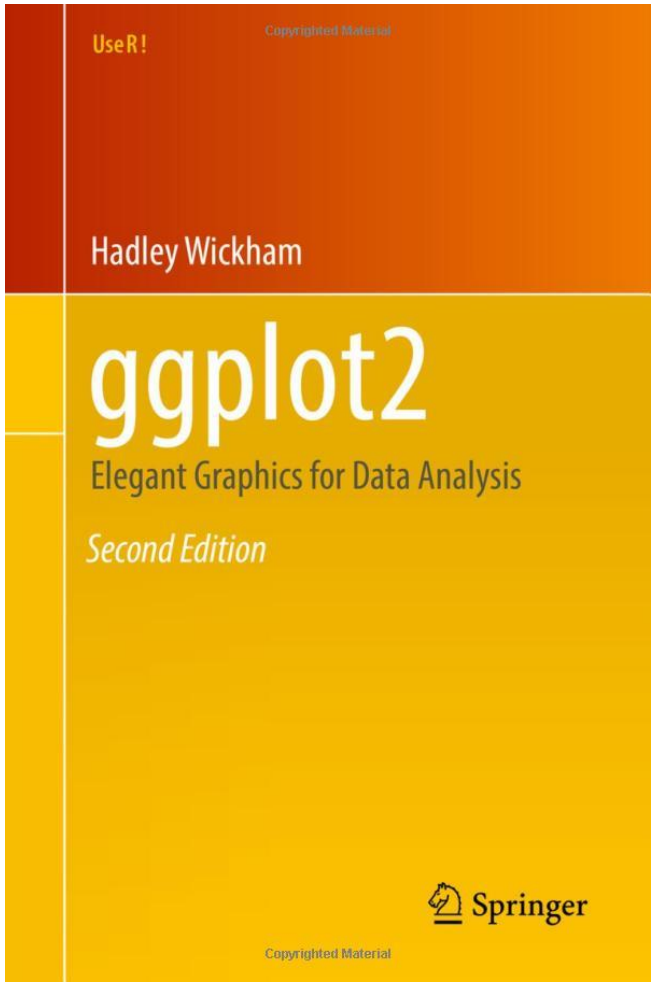
Ggplot

gg + scale_colour_brewer(palette = "Set1")



Libro ggplot

https://ggplot2-book.org/



Leaflet

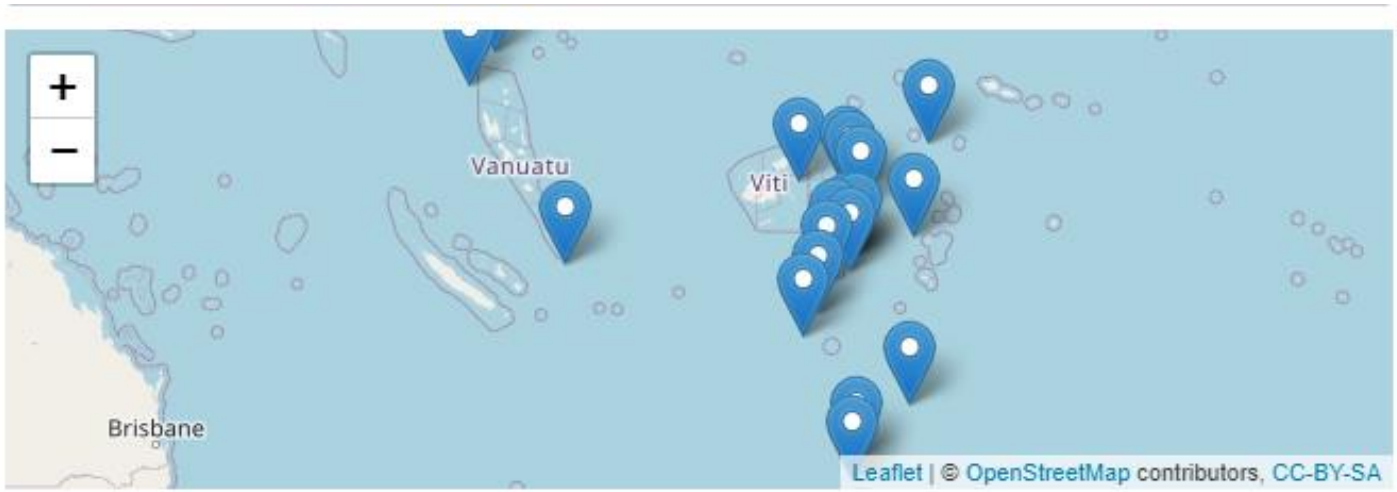
```
install.packages("leaflet")

library(leaflet)

m <- leaflet() %>%
  addTiles() %>%  # Add default OpenStreetMap map tiles
  addMarkers(lng=174.768, lat=-36.852, popup="The birthplace of R")
m  # Print the map
```

Leaflet - Tiles

```
m <- leaflet() %>% setView(lng = -71.0589, lat = 42.3601, zoom = 12)
m %>% addTiles()
m %>% addProviderTiles(providers$Stamen.Toner)
m %>% addProviderTiles(providers$Esri.NatGeoWorldMap)
```

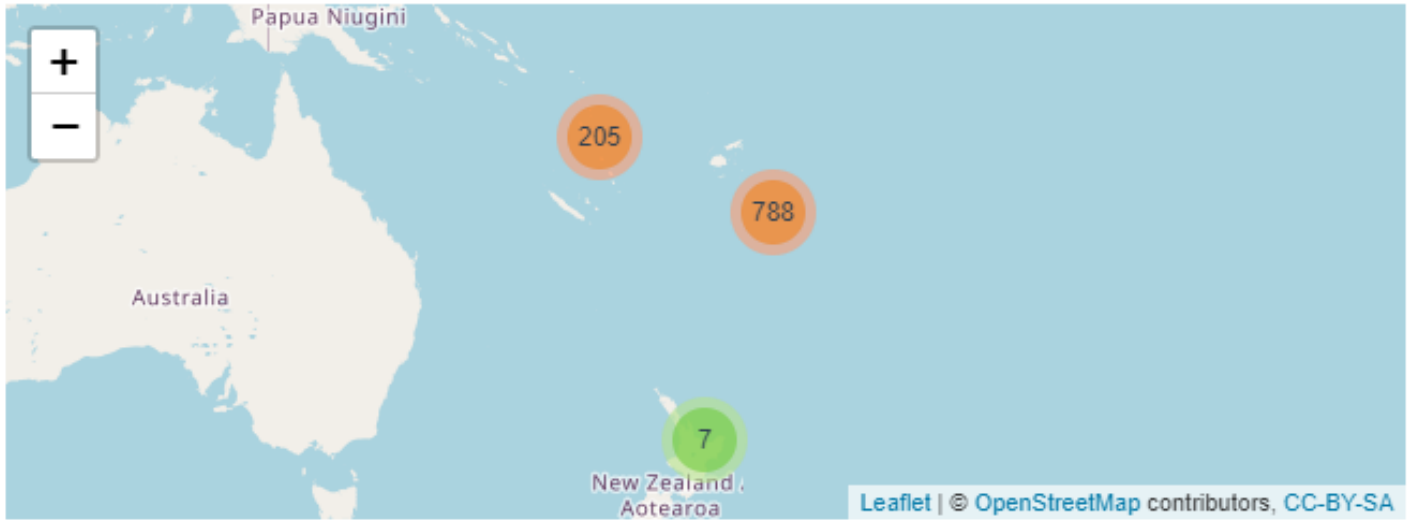
Leaflet - Markers

```
data(quakes)

# Show first 20 rows from the `quakes` dataset
leaflet(data = quakes[1:20,]) %>% addTiles() %>%
  addMarkers(~long, ~lat, popup = ~as.character(mag), label = ~as.character(mag))
```



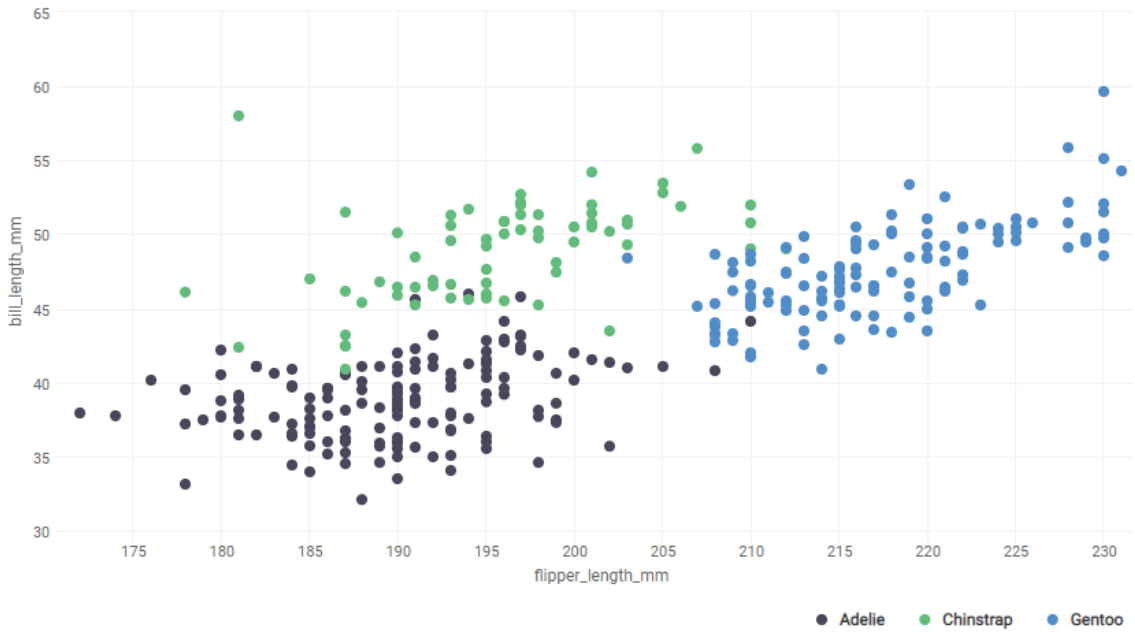
Leaflet – Marker Clusters

```
leaflet(quakes) %>% addTiles() %>% addMarkers(
  clusterOptions = markerClusterOptions()
)
```



Highcharts

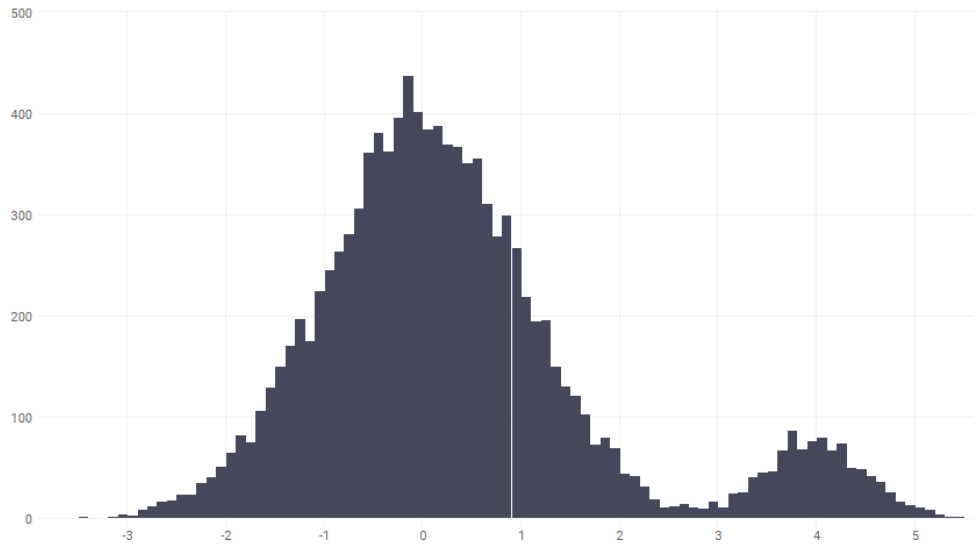
```
Install.packages(highcharter)
library(highcharter)
# remotes::install_github("allisonhorst/palmerpenguins")
data(penguins, package = "palmerpenguins")

hchart(penguins, "scatter", hcaes(x = flipper_length_mm, y = bill_length_mm, group = species))
```



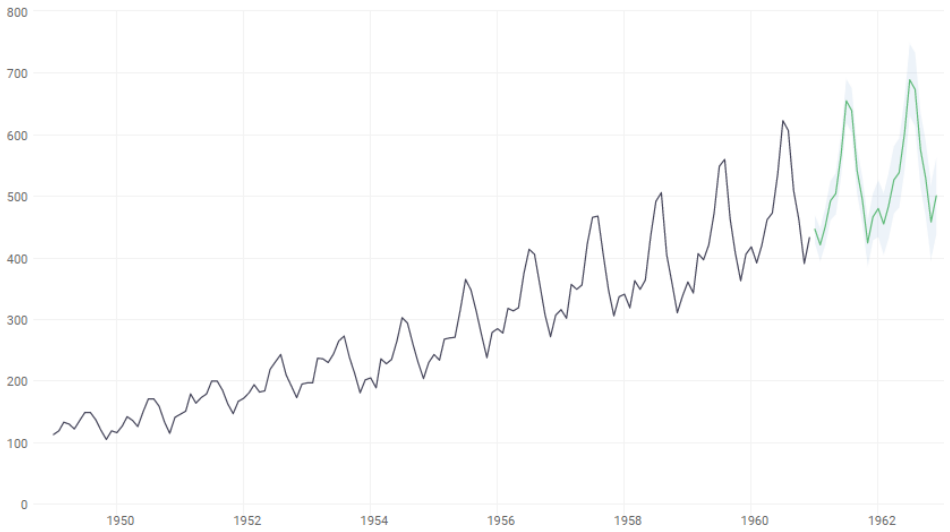
https://jkunst.com/highcharter/articles/highcharter.html

Highcharts

```
x <- c(rnorm(10000), rnorm(1000, 4, 0.5))

hchart(x, name = "data")
```




```
library(forecast)

airforecast <- forecast(auto.arima(AirPassengers), level = 95)

hchart(airforecast)
```



https://jkunst.com/highcharter/articles/highcharter.html

Highcharts

```
library(quantmod)

x <- getSymbols("GOOG", auto.assign = FALSE)
y <- getSymbols("AMZN", auto.assign = FALSE)

highchart(type = "stock") %>%
  hc_add_series(x) %>%
  hc_add_series(y, type = "ohlc")
```

https://jkunst.com/highcharter/articles/highcharter.html

EJERCICIO 7 – dplyr + gráficos

1. Instala la librería Titanic.
2. Haz un head del dataset titanic_train
3. Filtra el dataset para los viajeros que sobrevivieron y haz un conteo.
4. Crea un dataframe nuevo con el nombre y la edad de los pasajeros.
5. Filtra este dataframe para obtener todos los pasajeros que su apellido empieza por la letra G.
6. Del original, ¿cuál era la districución por sexo?
7. ¿Cuál era la edad con mayor frecuencia?
8. ¿Eran más jóvenes los supervivientes o los fallecidos?
9. Crea una nueva columna que nos diga si el pasajero era mayor o menor de edad.
10. ¿Quiénes fueron los pasajeros que pagaron más por el billete?
11. Haz un gráfico que represente sexo y clase.
12. Haz un gráfico para representar supervivientes por clases.
13. Haz un gráfico para representar supervivencia por precio del billete.
14. Haz un gráfico para representar supervivencia por sexo y clase.

Índice

Funciones en R

Todo lo que usas en R es una función

Las librerías permiten añadir nuevas funciones

R permite escribir nuevas funciones

Al escribir el código de una función sin paréntesis, se muestra el código de la función

```
> rep
function (x, ...)  .Primitive("rep")
> c
function (..., recursive = FALSE)  .Primitive("c")
> sum
function (..., na.rm = FALSE)  .Primitive("sum")
>
```

Funciones en R

Grupo de instrucciones que toma un **input** o datos de entrada, los procesa y devuelve un resultado
La sintaxis es la siguiente:

```
nombre_funcion <- function(input) {
  # Cuerpo de la funcion
  output <- input + 1
  return(output)
}
```

Se utiliza **return()** para establecer lo que devuelve

En el caso de no utilizar **return**, se devuelve el último valor asignado, dentro de la función

Funciones en R

```r
suma2 <- function(x, y)
{
    output <- x + y
    return(output)
}
```

```r
suma2bis <- function(x, y)
{
    x + y
}
```

Funciones en R

El valor que devuelve una función se puede almacenar en una variable

Dentro de una función es posible utilizar variables declaradas anteriormente

```
fmaximo <-function(v,f=max)
{
  output <- f(v)
  return(output)
}
```

```
> valores <- 1:34
> maximo <- fmaximo(valores)
> maximo
[1] 34
>
```

Estructuras de Control

Permiten el control de la ejecución de comandos

Con **?Control**, pueden verse las estructuras disponibles:

- If
- For
- While
- Repeat
- Break
- Next

Condicional if

Evalúa una condición y actúa en base a su resultado

```
fmultiplicacion <-function(v,m=2)
{
  if(v[1] > 5)
  {
    return(v*10)
  }
  else
  {
    return(v*m)
  }
}
```

Bucle for

Recorre un vector y ejecuta las instrucciones que se encuentran entre los corchetes:

```
> for (k in 1:8)
+ {
+     print(1:k)
+ }
[1] 1
[1] 1 2
[1] 1 2 3
[1] 1 2 3 4
[1] 1 2 3 4 5
[1] 1 2 3 4 5 6
[1] 1 2 3 4 5 6 7
[1] 1 2 3 4 5 6 7 8
>
```

Bucle repeat

Lanza un conjunto de comandos hasta que se encuentra con un **break**

```
> repeat
+ {
+     print(x)
+     if (x > 10) break
+     x <- x + 1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
[1] 11
>
```

Bucle while

Ejecuta un conjunto de comandos mientras que se cumpla una condición

```
> x <- 1
> while (x <= 10) {
+   print(x)
+   x <- x + 1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
>
```

Next

Permite parar la ejecución del bucle y pasar al siguiente ciclo

```
> for(i in 1:10)
+ {
+     if(i %in% 4:7)
+         next
+     print(i)
+ }
[1] 1
[1] 2
[1] 3
[1] 8
[1] 9
[1] 10
>
```

EJERCICIO 8 – Creando funciones

1. Crear una función que tome dos valores de entrada y devuelva el resultado de restar el segundo al primero

2. Crear una función que calcule la media de un vector

Índice

Librerias

- Son colecciones de funciones
- R viene con un conjunto estándar
- Muchos disponibles para su descarga e instalación
- Tienen que ser cargados para poder usarlos

Localizar un Paquete

Vista por tareas en CRAN

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

CRAN Task Views

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
Finance	Empirical Finance
Genetics	Statistical Genetics
Graphics	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
HighPerformanceComputing	High-Performance and Parallel Computing with R
MachineLearning	Machine Learning & Statistical Learning
MedicalImaging	Medical Image Analysis
MetaAnalysis	Meta-Analysis
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
NumericalMathematics	Numerical Mathematics
OfficialStatistics	Official Statistics & Survey Methodology
Optimization	Optimization and Mathematical Programming

Instalar un nuevo paquete

Con **library()** pueden verse los paquetes instalados
La función **install.packages()** se utiliza para instalar nuevos paquetes

```
> install.packages("ggplot2")
Warning in install.packages :
  downloaded length 227 != reported length 227
Installing package into 'C:/Users/se47351/Documents/R/win-library/3.1'
(as 'lib' is unspecified)
trying URL 'http://cran.rstudio.com/bin/windows/contrib/3.1/ggplot2_1.0.1.zip'
Content type 'application/zip' length 2676646 bytes (2.6 Mb)
opened URL
downloaded 2.6 Mb

package 'ggplot2' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
 C:\Users\se47351\AppData\Local\Temp\RtmpURskAn\downloaded_packages
```

Utilizar un paquete

- Es necesario utilizar la función **library()** antes de utilizar un paquete que no esté dentro de los paquetes base de R
- De esta forma se cargan en memoria todas las funciones y datos que contiene la librería

```
>
> library(ggplot2)
Warning message:
package 'ggplot2' was built under R version 3.1.3
>
```

Aprender R con R

{swirl}

Learn R, in R.

swirl teaches you R programming and data science interactively, at your own pace, and right in the R console!

- install.packages("swirl")
- library(swirl)

```
> library(swirl)

| Hi! Type swirl() when you are ready to begin.

> swirl()

| Welcome to swirl! Please sign in. If you've been here before, use the same name as you did then.
 If you are
| new, call yourself something unique.

What shall I call you? |
```

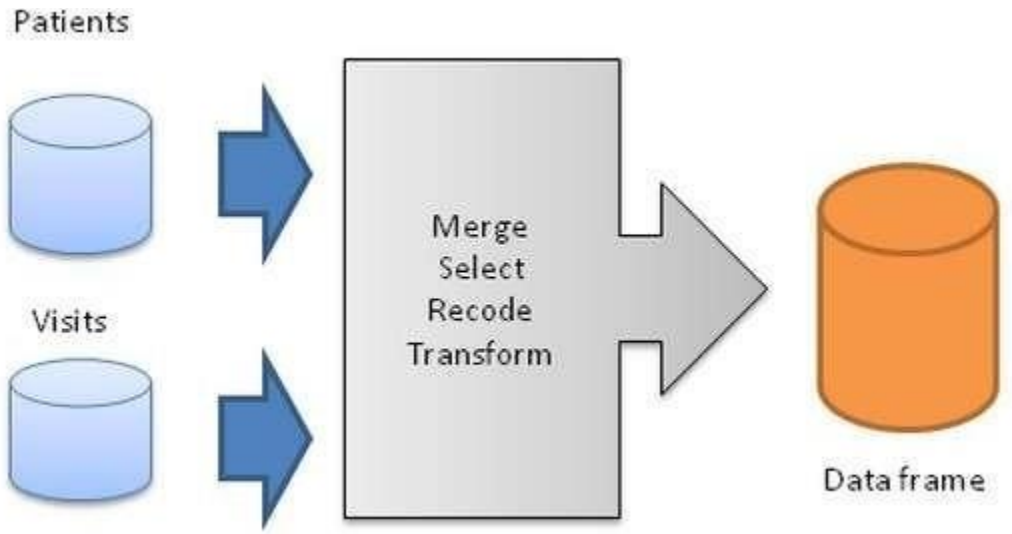
```
install_from_swirl("R Programming")
install_from_swirl("Getting and Cleaning Data")
install_from_swirl("Exploratory Data Analysis")
install_from_swirl("Open Intro")
install_from_swirl("Regression Models")
install_from_swirl("Statistical Inference")
```

Limpieza de datos

- Detección y localización de errores
- Corrección de errores
- Relleno de huecos
- Filas duplicadas
- Valores Imposibles
  - Fechas inconsistentes
  - Ventas negativas

Gestión de los datos

- Crear nuevas variables
- Cambiar de forma los datos
- Unir
- Ordenar
- Mezclar
- Agregar
- Filtrar



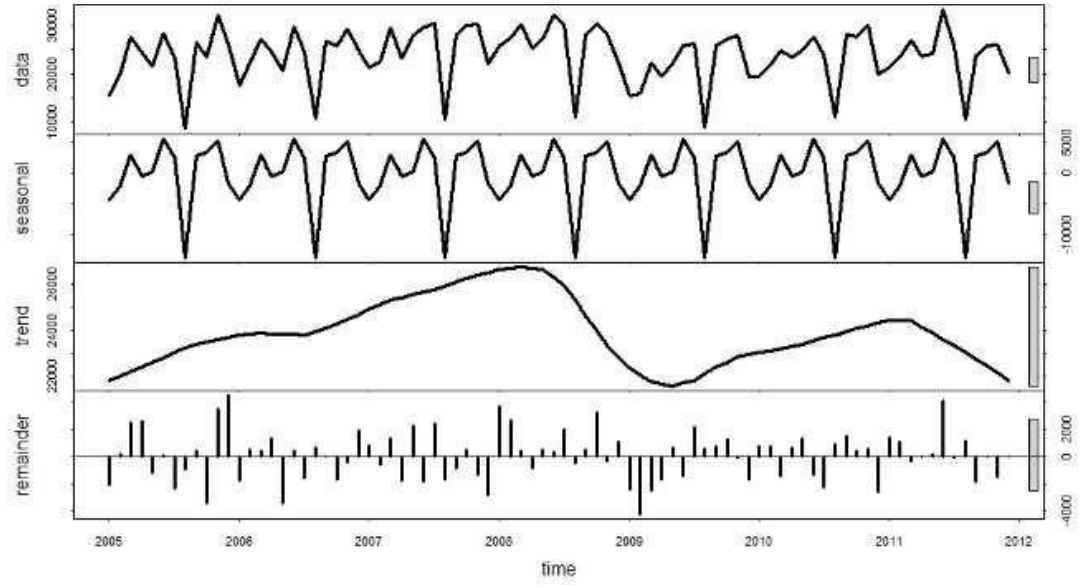
Manejo de fechas y horas en R

Tipos especiales para almacenar fechas y horas

Operaciones con fechas => **lubridate**

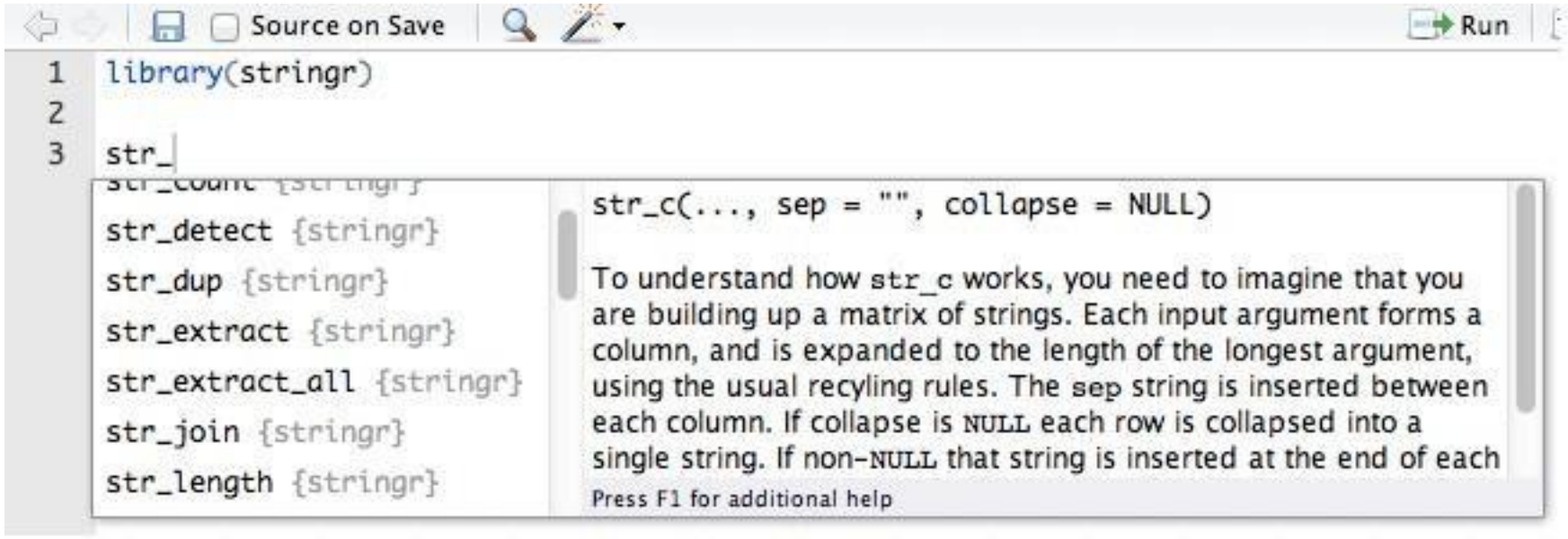
Análisis temporales en R

https://a-little-book-of-r-for-time-series.readthedocs.io/en/latest



Manejo de cadenas en R - stringr

Con R base contamos con un buen número de funciones para manipular cadenas, pero si son necesarias más, podemos usar el **paquete stringr**



Estructurar datos - tidyr

- Se utiliza para estructurar conjuntos de datos
- Paso previo a dplyr
- Sus principios son:
  - Cada variable debe tener su columna
  - Cada observación tiene que tener su fila
  - Cada tipo de unidad de observación (objeto) forma una tabla
- Si dataset no cumple estas condiciones => "Dato Sucio"
- Sus funciones más comunes son:
  - **Gather**: Cuando una variable está en más de una columna
  - **Separate**: Separar una columna en múltiples columnas
  - **Spread**: Cuando una observación esta en varias filas
  - **Unite**: Opuesto de separate

Gather

Toma múltiples columnas y las une en pares clave-valor.

```
df_gather
```

```
   Provincia Censo_1991 Censo_2001 Censo_2010
Buenos Aires   10934727   11460575   13596320
     Córdoba    1208554    1368301    1466823
     Rosario    1118905    1161188    1236089
```

Las columnas `Censo 1991`, `Censo 2001` y `Censo 2010`, son cada uno de los años en que se realizó el censo de población

```
df1 <- gather(data = df_gather, key = "censo", value = "poblacion", 2:4)
```

```
   Provincia      censo poblacion
Buenos Aires Censo_1991  10934727
     Córdoba Censo_1991   1208554
     Rosario Censo_1991   1118905
Buenos Aires Censo_2001  11460575
     Córdoba Censo_2001   1368301
     Rosario Censo_2001   1161188
Buenos Aires Censo_2010  13596320
     Córdoba Censo_2010   1466823
     Rosario Censo_2010   1236089
```

Lo que hemos hecho es decir a la función que tome el dataframe `df gather`, y una las columnas `2:4` (las que tienen los valores obtenidos en cada censo), metiendo el resultado en el par clave (*censo*) – valor (*población*).

Separate

Separar una columna en otras, dos o más

```
df_separate
```

```
     Pais Censo Tasa.de.poblacion.urbana
Argentina  1980  23198068 / 27949480
Argentina  1991  28832126 / 32615528
Argentina  2001  32380296 / 36260130
Argentina  2010  36907728 / 40117096
```

Se puede ver que la columna (`Tasa.de.poblacion.urbana`) contiene más de un valor. Para solucionar esto tenemos que emplear la función `separate`. Esta función lo que hace es dividir una columnas en múltiples columnas, tomando como separador algún símbolo.

```
df2 <- separate(data =  df_separate,
         col  =  Tasa.de.poblacion.urbana,
         into =  c("urbana", " total"),
         sep  =  "/")
```

```
     Pais Censo    urbana     total
Argentina  1980 23198068   27949480
Argentina  1991 28832126   32615528
Argentina  2001 32380296   36260130
Argentina  2010 36907728   40117096
```

Spread

Unificar observaciones que están en varias filas

```
df_spread
```

```
     Pais Censo   Tipo  Poblacion
Argentina  1980 urbana 23.198.068
Argentina  1980  total 27.949.480
Argentina  1991 urbana 28.832.127
Argentina  1991  total 32.615.528
Argentina  2001 urbana 32.380.296
Argentina  2001  total 36.260.130
Argentina  2010 urbana 36.907.728
Argentina  2010  total 40.117.096
```

En el siguiente ejemplo, vemos que una misma observación está en dos filas; una fila para la población total y otra para la población urbana. Para unificarlas, se usa la función `spread`.

```
df3 <- spread(data = df_spread, key = Tipo, value = Poblacion)
```

```
     Pais Censo      total     urbana
Argentina  1980 27.949.480 23.198.068
Argentina  1991 32.615.528 28.832.127
Argentina  2001 36.260.130 32.380.296
Argentina  2010 40.117.096 36.907.728
```

La columna que contiene los nombres de las variables a unificar, corresponde al argumento `key`, mientras que la columna que contiene el valor de la variable es `value`.

Unite

Función opuesta a `separate` que toma múltiples columnas y las une en una.

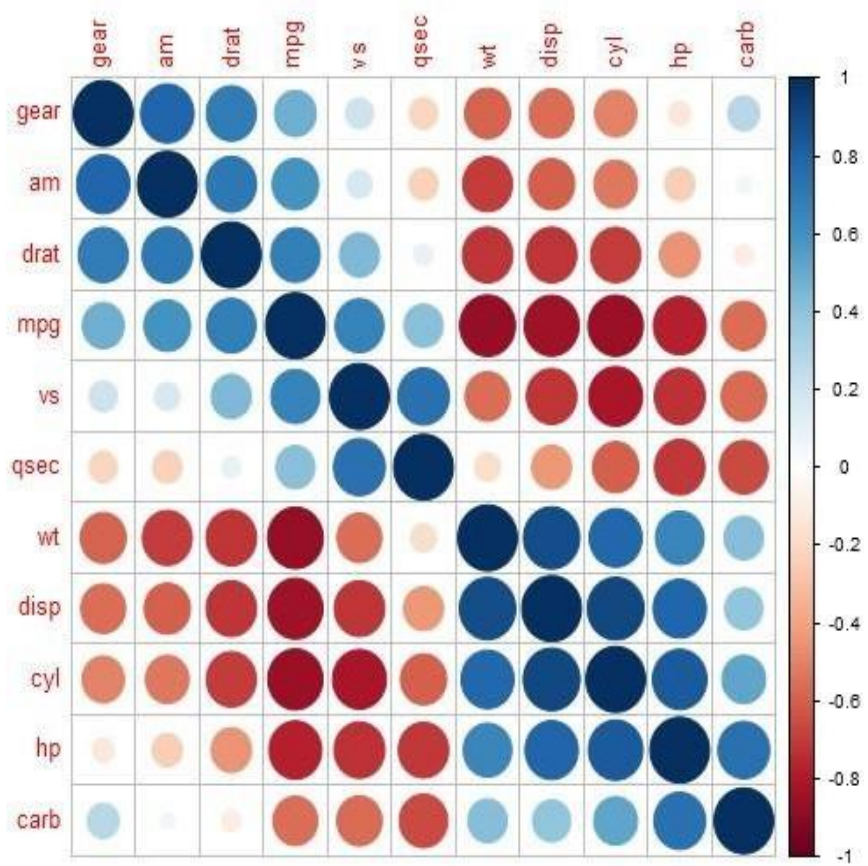
```
df_unite
```

```
ID dia mes anio
 1  25   9 2015
 2  30   7 2015
 3   7   3 2015
 4  15   8 2015
```

```
unite(data = df_unite, col = Fecha, sep = "-",dia,mes,anio)
```

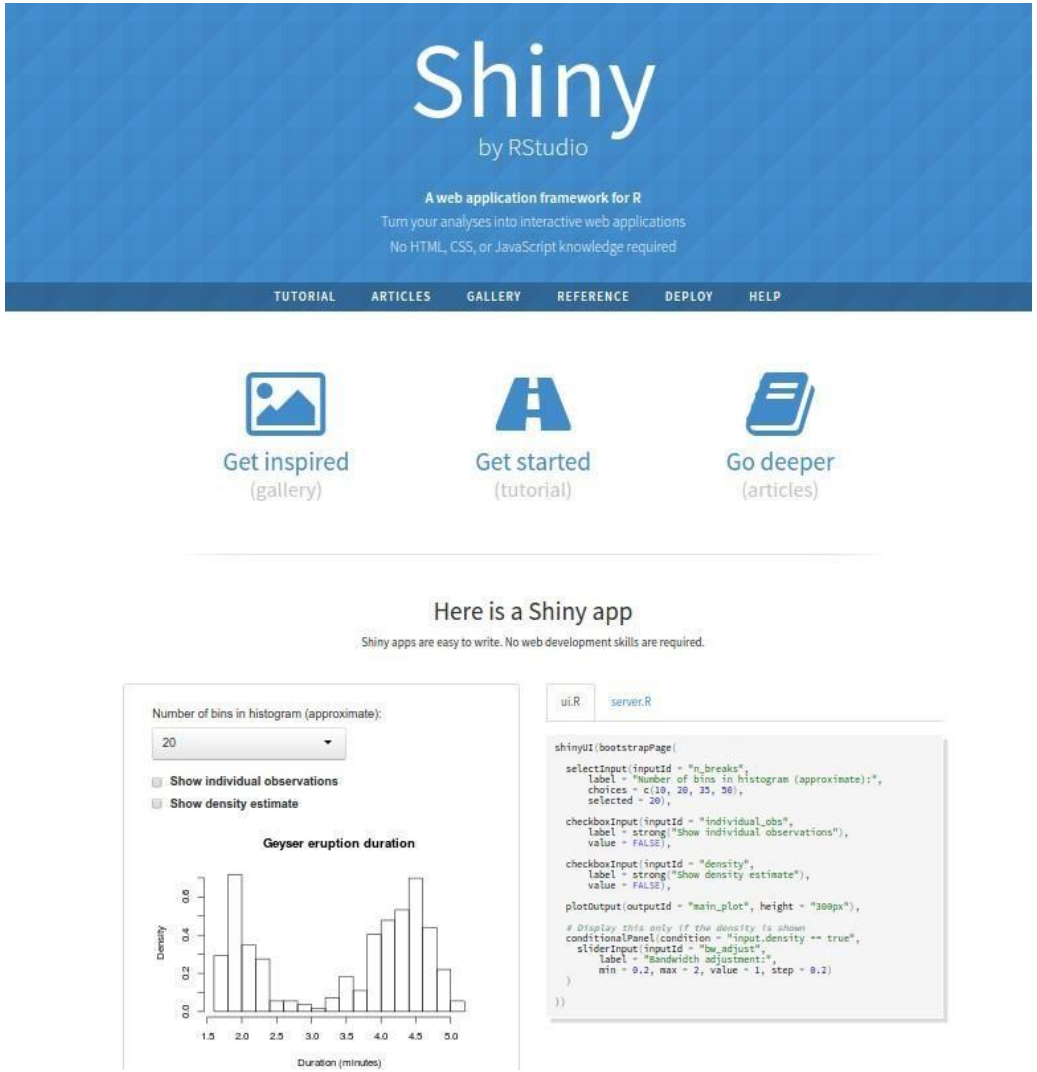
```
ID      Fecha
 1 25-9-2015
 2 30-7-2015
 3  7-3-2015
 4 15-8-2015
```

CORRPLOT – Matriz de Correlaciones



https://cran.r-project.org/web/packages/corrplot/vignettes/corrplot-intro.html

Shiny – Cuadros de Mando con R



Shiny - Aplicaciones web con R

- No hace falta saber HTML/Javascript
- Se combina bien con **ggvis**, versión interactiva de ggplot
- Los proyectos se suelen componer de dos archivos:
  - **server.R** => lógica para dibujar la visualización
  - **ui.R** => descripción del GUI
- En versiones recientes pueden unirse en uno solo llamado **app.R**
- Veamos un ejemplo:

1. Instalar el paquete: **install.packages("shiny")**
2. Cargar la librería: **library("shiny")**
3. Lanzar un ejemplo: **runExample("01_hello")**

Ejemplos de uso

CLUSTERIZAR IMÁGENES

NUBE DE PALABRAS

MAPAS DE VORONOY

BARPLOT ANIMADO

Gracias a todos/as por asistir.

datahack
BIG DATA FAMILY