



# Elementos básicos de R

Miguel David Alvarez Hernández

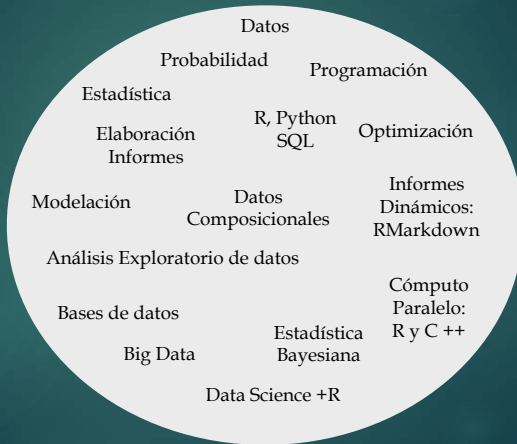
`mdalvarezh@gmail.com`

UAM-Lerma

Octubre 2022

# Y para qué necesitamos R ?

## El Arte del Análisis de Datos



# Resumen

R es un lenguaje y ambiente gratis para el cálculo y la graficación estadística que ofrece una amplia variedad de técnicas y gráficas estadísticas como: modelación lineal y no lineal en varios estratos, pruebas estadísticas, análisis de series de tiempo, técnicas multivariadas, etc. Iniciado como una implementación en "código abierto" (GNU) del language S, es sin lugar a duda, la herramienta de cómputo actualmente más usada en la investigación y desarrollo de la estadística a nivel mundial. El grupo principal de desarrollo de R está formado por estadísticos de primer nivel, incluyendo al autor original del lenguaje S, John Chambers de AT&T Labs, y los creadores de R, Robert Gentleman y Robert Ihaka.

Como ya se mencionó, R es "código abierto" disponible a todo el interesado, pero se tienen también versiones ya compiladas para las plataformas más comunes: MS Windows, Mac OS X, y varias versiones de Linux y Unix, lo que su instalación es inmediata. R es un software que tiene, entre otras, las siguientes características: a) No sólo es un software sino que es un lenguaje de programación característica que lo hace muy genérico para realizar prácticamente cualquier análisis estadístico. b) Contrario a lo que se pueda pensar es relativamente fácil de implementar y usar.

## Resumen

Además al ser un lenguaje orientado a objetos ofrece una gran flexibilidad para el análisis estadístico y desarrollo de técnicas aún no implementadas. Además existen poco más de 1000 paquetes desarrollados en R, desde aplicaciones Bayesianas, financieras, graficación de mapas, wavelets, microarreglos, etc.

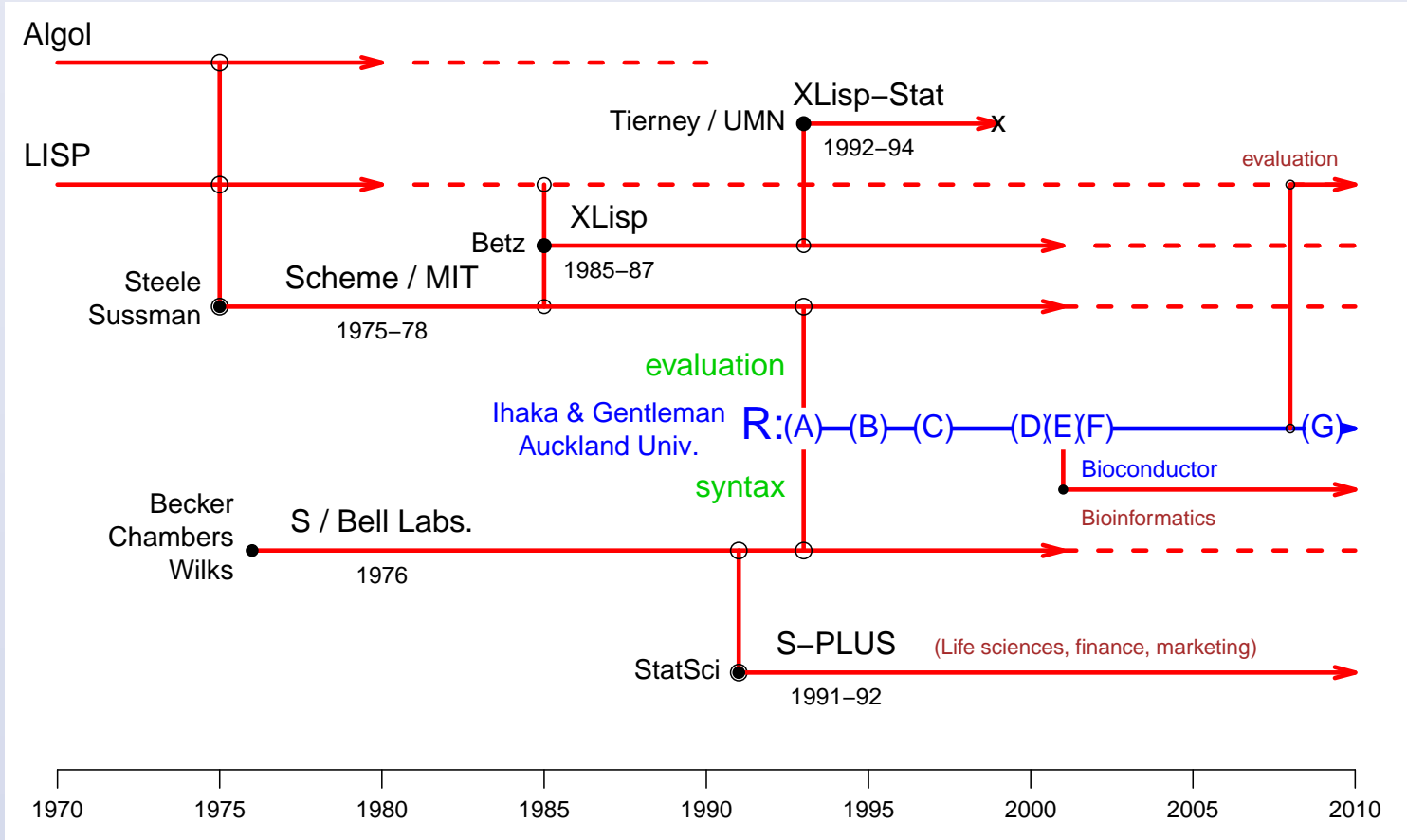
Este material pretende difundir y mostrar que R es una buena alternativa para realizar análisis estadístico y gráfico. Se ejemplifica su uso a través de diversas aplicaciones, por ejemplo, pruebas estadísticas, modelación, graficación, y programación de técnicas no incluidas en el lenguaje básico. Adicionalmente, el contenido de este trabajo es resultado de las discusiones con otros colegas y pretende ser en un futuro un documento que sirva como un curso de docencia.

# Instalación

- **Instalación:**

1. Google: R. (<http://www.r-project.org>) || CRAN
2. *Elegir alguna dirección URL* || <http://lib.stat.cmu.edu/R/CRAN/>
3. *Elegir algún sistema operativo* || Windows || base
4. Bajar el ejecutable

# Genealogía de R



## R: A Language for Data Analysis and Graphics

ROSS IHAKA and ROBERT GENTLEMAN

In this article we discuss our experience designing and implementing a statistical computing language. In developing this new language, we sought to combine what we felt were useful features from two existing computer languages. We feel that the new language provides advantages in the areas of portability, computational efficiency, memory management, and scoping.

**Key Words:** Computer language; Statistical computing.

---

Ross Ihaka is Senior Lecturer, and Robert Gentleman is Senior Lecturer, Department of Statistics, University of Auckland, Private Bag 92019, Auckland, New Zealand, e-mail: [ihaka@stat.auckland.ac.nz](mailto:ihaka@stat.auckland.ac.nz).

©1996 American Statistical Association, Institute of Mathematical Statistics,  
and Interface Foundation of North America  
*Journal of Computational and Graphical Statistics*, Volume 5, Number 3, Pages 299–314

# Estatutos de “*The R Foundation for Statistical Computing*”

## 1 Nombre, Lugar y Campo de Actividad

- a) La organización es llamada “*The R Foundation for Statistical Computing*”, abreviado “Fundación R”, que se usará en este documento.

## 2 Objetivos

### 1. Fundamentales

- (b) La “Fundación R” es una organización no lucrativa que trabaja por el interés público.

### 2. Los objetivos de la “Fundación R” son:

- (a) El avance del proyecto R del cálculo estadístico para proveer de un ambiente para el análisis de datos y graficación gratis y de código abierto.

## 3 Medios para cumplir los objetivos

### 1. Para cumplir estos objetivos la organización especialmente:

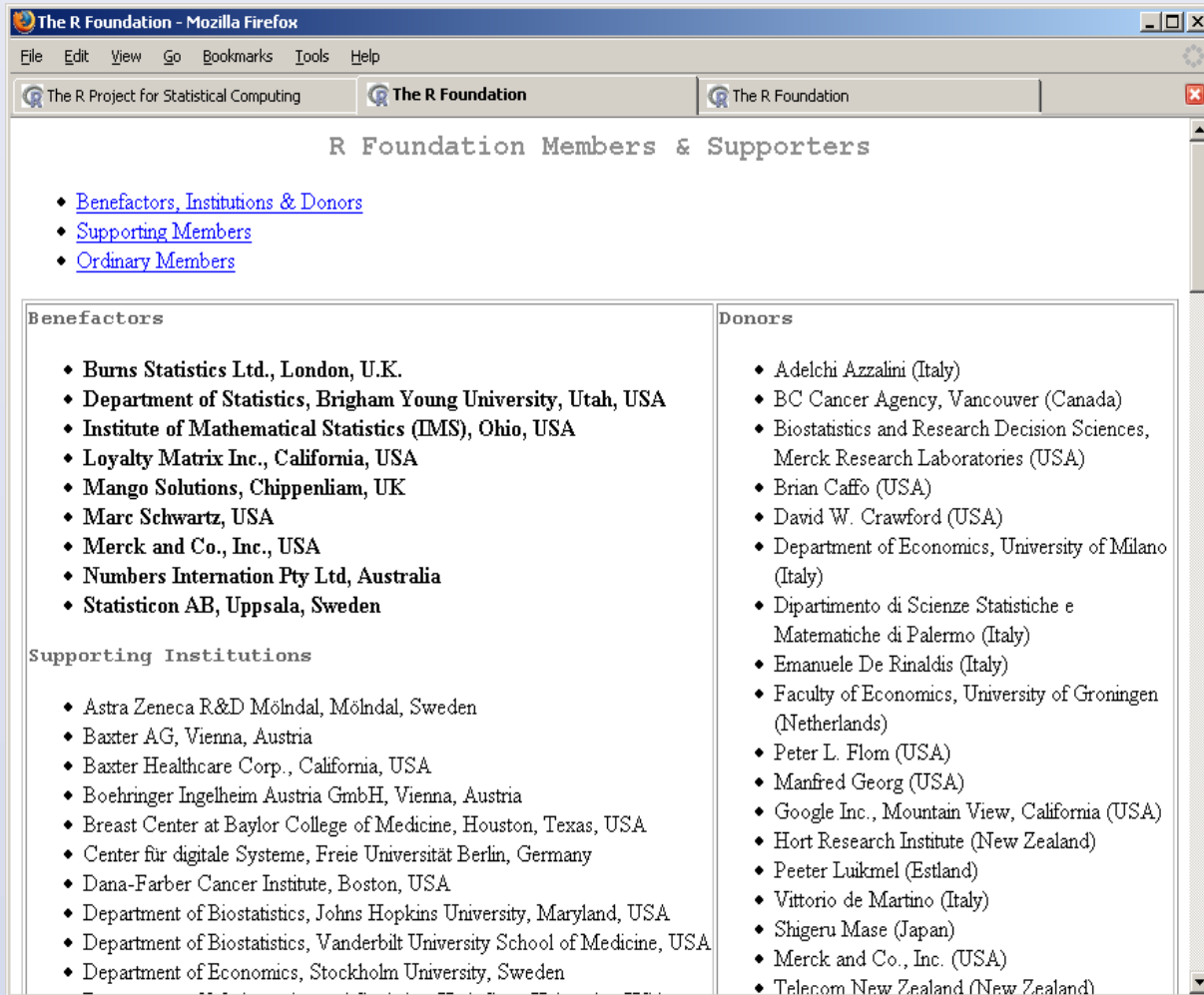
- (a) Apoyará en el desarrollo de R y proyectos de código abierto relacionados.



# Miembros y Benefactores de R



# Miembros y Benefactores de R



The R Foundation - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

The R Project for Statistical Computing The R Foundation The R Foundation

## R Foundation Members & Supporters

- [Benefactors, Institutions & Donors](#)
- [Supporting Members](#)
- [Ordinary Members](#)

Benefactors	Donors
<ul style="list-style-type: none"><li>• Burns Statistics Ltd., London, U.K.</li><li>• Department of Statistics, Brigham Young University, Utah, USA</li><li>• Institute of Mathematical Statistics (IMS), Ohio, USA</li><li>• Loyalty Matrix Inc., California, USA</li><li>• Mango Solutions, Chippenham, UK</li><li>• Marc Schwartz, USA</li><li>• Merck and Co., Inc., USA</li><li>• Numbers Internation Pty Ltd, Australia</li><li>• Statisticon AB, Uppsala, Sweden</li></ul>	<ul style="list-style-type: none"><li>• Adelchi Azzalini (Italy)</li><li>• BC Cancer Agency, Vancouver (Canada)</li><li>• Biostatistics and Research Decision Sciences, Merck Research Laboratories (USA)</li><li>• Brian Caffo (USA)</li><li>• David W. Crawford (USA)</li><li>• Department of Economics, University of Milano (Italy)</li><li>• Dipartimento di Scienze Statistiche e Matematiche di Palermo (Italy)</li><li>• Emanuele De Rinaldis (Italy)</li><li>• Faculty of Economics, University of Groningen (Netherlands)</li><li>• Peter L. Flom (USA)</li><li>• Manfred Georg (USA)</li><li>• Google Inc., Mountain View, California (USA)</li><li>• Hort Research Institute (New Zealand)</li><li>• Peeter Luikmel (Estonia)</li><li>• Vittorio de Martino (Italy)</li><li>• Shigeru Mase (Japan)</li><li>• Merck and Co., Inc. (USA)</li><li>• Telecom New Zealand (New Zealand)</li></ul>

Supporting Institutions

- Astra Zeneca R&D Mölndal, Mölndal, Sweden
- Baxter AG, Vienna, Austria
- Baxter Healthcare Corp., California, USA
- Boehringer Ingelheim Austria GmbH, Vienna, Austria
- Breast Center at Baylor College of Medicine, Houston, Texas, USA
- Center für digitale Systeme, Freie Universität Berlin, Germany
- Dana-Farber Cancer Institute, Boston, USA
- Department of Biostatistics, Johns Hopkins University, Maryland, USA
- Department of Biostatistics, Vanderbilt University School of Medicine, USA
- Department of Economics, Stockholm University, Sweden

# Introducción

- R ofrece una gran cantidad de funciones para realizar análisis gráfico y estadístico.
- ¿Cómo trabaja R?
- R es un lenguaje orientado a objetos. Es un intérprete no un compilador, esto significa que todos los comando escritos sobre la interface se ejecutan directamente sin que se requiera escribir un programa completo como en otros lenguajes como C, Fortran, Pascal, etc.
- Una vez que se abre R aparece el prompt de default “>”, lo que indica que R espera algún comando.

## Introducción

- El nombre de un objeto debe empezar con una letra (A-Z y a-z) y puede incluir dígitos y puntos.
- R discrimina para el nombre de los objetos letras mayúsculas de minúsculas, por lo que **x** y **X** nombrarán a diferentes objetos.
- En R para ejecutar una función, ésta siempre se debe escribir con paréntesis aunque no haya nada dentro de ellos. Por ejemplo:

`ls()`

desplegará el contenido del directorio de trabajo actual.

- Los argumentos de una función pueden ser en si objetos (datos, fórmulas, matrices, tablas, etc.)

more . . .



# Creando Objetos

- La forma de asignar objetos en R es a través del simbolo  $<-$ . Por ejemplo:

```
> x<- 56
```

```
> X<- 23
```

```
>x;X
```

```
[1] 56 [1] 23
```

```
>
```

```
> n<- sqrt(X)
```

```
> n
```

```
[1] 4.795832
```

```
> m<- 3+n
```

```
> m
```

```
[1] 7.795832
```

```
> m.aux<-10*n
```

```
> m.aux
```

## Creando Objetos

- Para borrar objetos de la memoria se usa la función `rm()`

```
> ls()
[1] "m"          "m.aux"      "n"          "x"          "X"
> rm(X)
> rm(n,x)
> ls()
[1] "m"          "m.aux"      "m.aux.2"
>
```

# Ayudas

- R ofrece ayuda en línea a través de la función `help()`.

```
>help(rm)
```

```
>?rm
```

- R hace también búsquedas inteligentes

```
>help.search("rose diagram")
```

Help files with alias or concept or title matching rose diagram  
using fuzzy matching:

<code>rose.diag(CircStats)</code>	Rose Diagram
<code>rose.diag(circular)</code>	Rose Diagram

Type `help(FOO, package = PKG)` to inspect entry `FOO(PKG) TITLE`.

- Adicionalmente, se pueden hacer búsquedas en el sitio web de R.

```
>RSiteSearch("rose diagram")
```



# Tipo de Objetos

- Todos los objetos tienen dos atributos intrínsecos: tipo (mode) y longitud (length)
- El tipo de objeto puede ser: numérico, caracter, complejo y lógico(FALSE/TRUE)

```
> x<- 1
> mode(x)
[1] "numeric"
> length(x)
[1] 1
> nombre.1<-"Aprobado" ; compara<-TRUE ; z<-1i
> mode(nombre.1) ; mode(compara) ; mode(z)
[1] "character"    [1] "logical"      [1] "complex"
```

# Generación de Datos

## Secuencias Regulares

- Algunas sucesiones se pueden generar de la siguiente manera:

```
> x<-1:15
> x
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
> y<-seq(1,5,0.5)
> y
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
> w<-seq(length=9,from=1,to=5)
> w
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

## Generación de Variables Aleatorias

- Generando observaciones de variable aleatorias

```
> x<-rnorm(1000)
> y<-rnorm(1000,5,3)
> u<-runif(2000,-1,1)
> z<-rexp(1000,2)
```

- No sólo se pueden generar observaciones de variable aleatorias. También se puede obtener la densidad, la probabilidad acumulada y los cuantiles de la correspondiente variable.

```
dnorm(x, mean=0, sd=1)
pnorm(q, mean=0, sd=1)
qnorm(p, mean=0, sd=1)
rnorm(n, mean=0, sd=1)
```

more . . .



# Manipulación de vectores y matrices

- La forma más común de crear vectores es con la función `c()`.

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

- Las operaciones aritméticas (+, -, \*, /, %, etc.) entre vectores se realizan elemento a elemento.

```
> X<-c(10,11,12,100,-5,-6)
> x*X
[1] 10 22 36 400 -25 -36
> X+1
```

## Manipulación de vectores y matrices

- Las matrices pueden ser creadas a partir de vectores o directamente usando la función `matrix()`.

```
> matrix(data=5,nrow=2,ncol=3)
```

	[,1]	[,2]	[,3]
[1,]	5	5	5
[2,]	5	5	5

```
> matrix(0,ncol=3,nrow=2)
```

	[,1]	[,2]	[,3]
[1,]	0	0	0
[2,]	0	0	0

```
> matrix(1:6,2,3)
```

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

```
> matrix(1:6,2,3,byrow=T)
```

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

## Manipulación de vectores y matrices

```
> x<-c(1,2,3,4,5,6)
```

```
> X<-c(10,11,12,100,-5,-6)
```

```
> M<-matrix(c(x,X), ncol=2)
```

```
> M
```

	[,1]	[,2]
[1,]	1	10
[2,]	2	11
[3,]	3	12
[4,]	4	100
[5,]	5	-5
[6,]	6	-6

## Manipulación de vectores y matrices

- También se pueden crear matrices a través de las funciones `rbind()` y `cbind()`, estas ligan o adjuntan matrices por renglón o columna, respectivamente.

```
> m1<-matrix(1,2,2)
```

```
> m2<-matrix(2,2,2)
```

```
> M1<-rbind(m1,m2)
```

```
> M1
```

	[,1]	[,2]
[1,]	1	1
[2,]	1	1
[3,]	2	2
[4,]	2	2

```
> M2<-cbind(m1,m2)
```

```
> M2
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	1	2	2
[2,]	1	1	2	2



## Manipulación de vectores y matrices

- La multiplicación de matrices, de dimensiones adecuadas, se realiza a través de la función `%*%`.
- La transpuesta de una matriz se obtiene con la función `t()`.

```
> t(M2)
```

	[,1]	[,2]
[1,]	1	1
[2,]	1	1
[3,]	2	2
[4,]	2	2

```
> M1%*%M2
```

	[,1]	[,2]	[,3]	[,4]
[1,]	2	2	4	4
[2,]	2	2	4	4
[3,]	4	4	8	8
[4,]	4	4	8	8

```
> M2%*%M1
```

	[,1]	[,2]
[1,]	10	10
[2,]	10	10

## Manipulación de vectores y matrices

- La función `diag()` puede ser usada para extraer o modificar la diagonal de una matriz o para construir una matriz diagonal.

```
M3<- t(M2)%*%M2
> M3
```

	[,1]	[,2]	[,3]	[,4]
[1,]	2	2	4	4
[2,]	2	2	4	4
[3,]	4	4	8	8
[4,]	4	4	8	8

```
> diag(M3)
[1] 2 2 8 8
```

```
> diag(4)
```

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

```
> diag(c(10,20,30))
```

	[,1]	[,2]	[,3]
[1,]	10	0	0
[2,]	0	20	0
[3,]	0	0	30

## Manipulación de vectores y matrices

- \* Al igual que con los vectores las operaciones aritméticas se realizan elemento a elemento.

```
> m1+m2
```

```
      [,1] [,2]  
[1,]     3     3  
[2,]     3     3
```

```
> m1*m2
```

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

```
> m1/m2
```

```
      [,1] [,2]  
[1,]  0.5  0.5  
[2,]  0.5  0.5
```

```
> m1<-matrix(1,2,2)
```

```
> m1
```

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

```
> m1<-matrix(2,2,2)
```

```
> m2
```

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

more . . .



# Lectura de Datos

- Para leer datos desde un archivo se pueden utilizar las funciones `read()`, `table()` o `scan()`. Esta última es más flexible y permite especificar el tipo de cada variable.

```
> mydata<-scan(file="./NMV.dat2", what=list("",0,0))
```

```
Read 15 records
```

```
> mydata
```

```
[[1]] [1] "18.61664" "19.43575" "20.20695" "21.84337" "21.34864"  
[[2]] [1] 20.48832 17.99986 21.38629 17.97225 22.99391 [[3]]  
[1]18.70510 20.18638 21.75702 22.66418 20.04545
```

- Se puede notar que la primera variables es de tipo caracter y las otras dos de tipo numérico

## Lectura de Datos

- Adicionalmente, la función `scan()` junto con la función `matrix()` puede ser usada para crear diferentes objetos como matrices, vectores, etc.

```
> datos.aux<-scan(file="./NMV.dat2")
```

```
Read 15 items
```

```
> datos.aux
```

```
[1] 18.61664 20.48832 18.70510 19.43575 17.99986 20.18638 20.20695  
21.38629 21.75702 21.84337 17.97225 22.66418 21.34864 22.99391  
20.0454
```

```
> datos<-matrix(datos.aux,ncol=3)
```

```
> datos
```

```
      [,1]      [,2]      [,3]  
[1,] 18.61664 20.18638 17.97225  
[2,] 20.48832 20.20695 22.66418  
[3,] 18.70510 21.38629 21.34864  
[4,] 19.43575 21.75702 22.99391  
[5,] 17.99986 21.84337 20.04545
```

## Lectura de Datos

- Leer tablas de datos en los formatos usuales de texto (separado por comas, ancho fijo) es fácil. En el siguiente ejemplo cambiamos directorio de trabajo, leemos de un archivo separado por comas con encabezado (nombres de las columnas) que esta en nuestro sistema e imprimimos la tabla:

```
> setwd("K:/Datos")  
> mi.tabla <- read.table("mis_datos2.csv", sep = ",",  
  header = TRUE)  
> print(mi.tabla)
```

	ind	var1	var2
1	1	rojo	0.095
2	2	rojo	3.221
3	3	verde	-2.342

## Lectura de Datos

- De manera similar podemos leer archivos de otros formatos usando la librería *foreign*). Por ejemplo, para leer un archivo dbf hacemos:

```
> library(foreign)
> setwd("K:/Datos")
> so2_df<-read.dbf("2008S02P.DBF")
#Sólo imprime 4 renglones y 3 columnas
> print(so2_df[1:4,1:3])
```

	FECHA	HORA	MSO2VAL
1	2008-01-01	1	0.010
2	2008-01-01	2	0.016
3	2008-01-01	3	0.013
4	2008-01-01	4	0.012



## Lectura de Datos

- Todo el archivo se lee a la memoria en los ejemplos anterior, así que no siempre podemos hacer esto con archivos muy grandes. En estos casos, podemos abrir conexiones y leer y procesar líneas secuencialmente (nótese también que con las primeras dos líneas bajamos el archivo del *internet movie database*):

```
> #url_base<-"ftp://ftp.sUNET.se/pub/tv+movies/imdb/
  ratings.list.gz"
> #download.file(url_base,"ratings.list.gz")
> ratings_gz<-gzfile("ratings.list.gz","r")  #Abrir conexión
> encabezado<-readLines(ratings_gz,27)        #Leer líneas 1-27
> lineas<-readLines(ratings_gz,4)             #Leer líneas 28-31
> close(ratings_gz)
> print(lineas)

[1] "New Distribution Votes Rank Title"
[2] "    0000000115  371002  9.1 Shawshank Redemption, The (1994)"
[3] "    0000000015  316612  9.1 Godfather, The (1972)"
[4] "    0000000016  264079  9.0 Dark Knight, The (2008)"
```

more . . .

- ```
>download.file("http://www.stat.columbia.edu/~gelman/arm/
examples/death.polls/polls.dat", "polls.dat")

>polls <- matrix (scan("polls.dat"), ncol=5, byrow=TRUE)
```

# Respaldo de Datos

- Para salvar o respaldar datos en un archivo uno de los comandos que se puede utilizar es la función `write()`.

```
> write(datos,file="./salida.R", ncol=2)
```

## Respaldo de Datos

- Tablas de datos en la memoria de R pueden guardarse como archivos delimitados:

```
> lineas<-read.table(file="Sesiones/Datasets/mis_datos2.txt", sep="
> write.table(lineas, file = "nuevo2.csv", sep = ",",
  row.names = FALSE)
```

- Podemos verificar leyendo de nuevo el archivo:

```
> nuevas_lineas <- read.table("nuevo2.csv", sep = ",",
  header = TRUE)
> print(nuevas_lineas)
```

|   | ind | var1  | var2   |
|---|-----|-------|--------|
| 1 | 1   | rojo  | 0.095  |
| 2 | 2   | rojo  | 3.221  |
| 3 | 3   | verde | -2.342 |

## Respaldo de datos

- Mientras trabajemos con R, sin embargo, conviene más usar el formato interno de R:

```
> save(lineas,file="mis_datosR.rdata")  
> rm(list=ls()) #Borrar objetos de \textsf{R} en sesión  
> load("mis_datosR.rdata")  
> ls()  
> head(lineas)  
> head(lineas,2)
```

more . . .



# Listas

- Los datos en R se almacenan en objetos. Los objetos con los que más usualmente trabajamos son *listas*. Las listas son estructuras de datos recursivas: son colecciones ordenadas de listas. Podemos crear una lista con la función *list*:

```
> manzana <- list(nombre = "manzana", peso = 36.2, tipo = "fruta")  
> pera <- list(nombre = "pera", tipo = "fruta", peso = 25)  
> print(manzana)
```

```
$nombre
```

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

```
$peso
```

```
[1] 36.2
```

```
$tipo
```

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

```
> frutas <- list(unos = manzana, dos = pera)
```

```
> names(manzana)
```

```
[1] "nombre" "peso"    "tipo"
```

```
> names(frutas)
```

## Listas

- donde la función `names` nos da los nombres de las componentes de la lista. Usando los nombres podemos extraer componentes de la lista de la siguiente forma:

```
> manzana$tipo
```

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

```
> manzana$peso
```

```
[1] 36.2
```

```
> pera$peso
```

```
[1] 25
```

```
> manzana$peso + pera$peso
```

```
[1] 61.2
```



## Listas

- Aunque también podemos extraer las componentes con la notación de doble corchete:

```
> pera[[1]]
```

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

```
> manzana[[2]] + pera[[2]]
```

```
Error in manzana[[2]] + pera[[2]] :  
non-numeric argument to binary operator
```

## Listas

- La última línea nos da un error: no podemos sumar la componente 2 de **manzana** con la componente 2 de **pera**. Este tipo de errores son comunes, y pueden ser muy frustrantes para los principiantes de R. Cuando aparecen estos errores, conviene checar los tipos de cada uno de los objetos sobre los que estamos operando con la función `mode`, que nos dice el tipo de las componentes de cada objeto.

```
> mode(pera$tipo)
```

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

```
> mode(manzana$peso)
```

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

```
> mode(manzana)
```

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

el problema es que la suma no está definida para un objeto de tipo *character* y uno de tipo *numeric*.

more . . .



# Data frame

- Construyamos dos vectores: uno numérico y uno de caracteres.

```
> x <- c(1.5, 2.2, -5)
> a <- c("manzana", "pera", "plátano")
> mode(x); mode(a)
[1] "numeric"
[1] "character"
```

- Ahora, podemos juntar estos dos vectores en una lista

```
> mi_lista <- list(medida = x, fruta = a)
> mode(mi_lista)
[1] "list"
> print(mi_lista$medida)
[1] 1.5 2.2 -5.0
> print(mi_lista)
$medida
```

## Data Frame

- Ya casi tenemos la estructura más usual en R. Para crearla usamos la función `data.frame`:

```
> mis_datos <- data.frame(mi_lista)
> mode(mis_datos)
```

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

```
> names(mis_datos)
```

```
[1] "medida" "fruta"
```

```
> mode(mi_lista)
```

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

```
> names(mi_lista)
```

```
[1] "medida" "fruta"
```

## Data Frame

- Nótese que cuando aunque el tipo de `mis_datos` sigue siendo lista, cuando hicimos `print(mis_datos)` no obtuvimos el mismo resultado que cuando hicimos `print(mi_lista)`. La razón es que al usar la función `data.frame` convertimos a `mi_lista` en un objeto de una *clase* distinta:

```
> class(mi_lista)
```

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

```
> class(mis_datos)
```

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

- La función `print` funciona de distinta manera según la clase de su argumento.
- En general, todos los objetos además de tener un tipo o modo tienen una clase. La clase es estructura adicional que determina, entre otras cosas, cómo aplican las funciones a ese objeto.

more . . .



# Accesando los valores de un objeto

- El sistema de índices: La nomenclatura `[i,]` y `[,j]` es usada en **R** para hacer referencia a un renglón completo o a una columna completa de una matrix.

```
> M<-matrix(1:20,4,5)
```

```
> M
```

|      | [,1] | [,2] | [,3] | [,4] | [,5] |
|------|------|------|------|------|------|
| [1,] | 1    | 5    | 9    | 13   | 17   |
| [2,] | 2    | 6    | 10   | 14   | 18   |
| [3,] | 3    | 7    | 11   | 15   | 19   |
| [4,] | 4    | 8    | 12   | 16   | 20   |

```
> Mrow1<-M[1,]
```

```
> M[,2]
```

```
> Mrow1
```

```
[1] 5 6 7 8
```

```
[1] 1 5 9 13 17
```



## Accesando los valores de un objeto

- Se pueden seleccionar submatrices, en forma similar, con la ayuda de la función `c()`. Por ejemplo, para seleccionar las columnas 1 y 3 de la matriz `M`.

```
> M
      [,1] [,2] [,3] [,4] [,5]
[1,]     1     5     9    13    17
[2,]     2     6    10    14     1
[3,]     3     7    11    15    19
[4,]     4     8    12    16    20
```

- Elementos particulares de una matriz se accesan usando la nomenclatura `[i,j]`.

```
> Msub<-M[,c(1,3)]
```

```
> Msub
      [,1] [,2]
[1,]     1     9
[2,]     2    10
[3,]     3    11
[4,]     4    12
```

```
> x<-Msub[1,2]
```

```
> x
[1] 9
```

## Accesando los valores de un objeto

- Elementos y subconjuntos de elementos de vectores y matrices se pueden excluir especificando un entero negativo o un conjunto de enteros negativos.

```
> M
```

|      | [,1] | [,2] | [,3] | [,4] | [,5] |
|------|------|------|------|------|------|
| [1,] | 1    | 5    | 9    | 13   | 17   |
| [2,] | 2    | 6    | 10   | 14   | 18   |
| [3,] | 3    | 7    | 11   | 15   | 19   |
| [4,] | 4    | 8    | 12   | 16   | 20   |

```
> Msub2<-M[,-c(1,5)]
```

```
> Msub2
```

|      | [,1] | [,2] | [,3] |
|------|------|------|------|
| [1,] | 5    | 9    | 13   |
| [2,] | 6    | 10   | 14   |
| [3,] | 7    | 11   | 15   |
| [4,] | 8    | 12   | 16   |

## Accesando los valores de un objeto

- Otros subconjuntos de elementos de vectores y matrices se pueden obtener usando operadores logicos, tales como:  $<$ ,  $>$ ,  $<=$  (menor o igual a),  $>=$  (mayor igual a),  $==$  (igual a).

```
> X<-2*(1:10)
```

```
> X
```

```
[1]  2  4  6  8 10 12 14 16 18 20
```

```
> X[X<=10]
```

```
[1]  2  4  6  8 10
```

```
> Y<-rep(0,10)      # la funcion rep se usa para
```

```
> Y                  # obtener un vector de ceros (de longitud 10).
```

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

```
> Y[c(1,10)]<-1     # le asigna 1 a los elementos 1 y 10.
```

```
> X[Y>0]             # ¿Sabes cual es el resultado de esta  
# operacion?
```

- La función **which()** también es muy útil para seleccionar subconjuntos de vectores y matrices que cumplen con cierta condición. Por ejemplo, los siguientes procedimientos producen el mismo resultado.

```
> y<-rnorm(1000); x<-rnorm(1000); X<-cbind(x,y)
```

## Procedimiento 1

```
> z1.resp<-0
> for(j in 1:1000){
+   if(X[j,1]<X[j,2]) z1.resp<-z1.resp+exp(-(X[j,1]+X[j,2])) }
> z1.resp
[1] 1321.078
```

## Procedimiento 2

```
> indice<-which(X[,1]<X[,2])
> zz<-exp(-(X[,1]+X[,2]))
> z2.resp<-sum(zz[indice])
> z2.resp
[1] 1321.078
```

## more . . .New slide

- Elementos particulares.
- Subvectores: La función `c()`.
- Subvectores: La función `-c()`.
- Selección condicional

```
> mydata<-scan(file="Datos\\NMV.dat2")
> dat<-matrix(mydata,ncol=3,byrow=T)
> names(dat)<-c("auto", "vida","otro")
> dat
##
##
> v1<-dat1$auto
> v1[5]
> v1[c(3,5,7)]
> v1[c(5:10)]
> a1<-v1[-c(1:5)]
#
```

## more . . .New slide

```
> v1[v1>20]  
> which(v1>20) # Qué hace esta instrucción  
#  
> dat1$auto[dat1$vida<18]
```

- Se pueden usar los operadores lógicos: & (and), | (or), != (diferente)
- ¿Qué hacen las siguientes instrucciones?

```
> dat1$vida[dat1$auto>19 & dat1$vida<=22]  
> dat1$vida[dat1$auto>19 | dat1$vida<=22]  
> which(dat1$auto != 18.61664)  
> which(dat1$auto != 18.61664)
```

## more . . .New slide

- Existen algunas bases de datos disponibles en R.

```
> data()  
> head(swiss)  
> help(swiss)
```

- Aunque el sistema de índices para extraer partes de vectores, matrices o data frame son lógicos, se pueden emplear funciones adicionales que hacen las cosas un poco más fácil: `subset()` y `transform()`.

```
> subbase<-subset(swiss,Fertility<60)  
> subbase  
> subbase2<-transform(swiss,log.Education=log(Education))  
> subbase2
```

- Para aprender mas, **MIRE** el *help* de `subset`.

```
> head(airquality)  
> subbase3<-subset(airquality, Temp <60, select = c(Ozone, Temp))
```

## Accesando los valores de un objeto: Listas

- Una gran parte de los objetos usuales de R son listas. Esto incluye los *marcos de datos* o *data frames*, y los objetos que dan como resultado distintos análisis. Veamos unos ejemplos:

```
> setwd("K:/Datos")      # Observe: setwd("K:\\Datos")
> so2_df <- read.table("2008S02P2.csv", sep = ",", header = TRUE)
> mode(so2_df)
```

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

```
> class(so2_df)
```

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

La función *read.table* produce un marco de datos. Las componentes son

```
> names(so2_df)
```

```
[1] "FECHA.D"      "HORA.N.2.0"    "MSO2VAL.N.7.3" "MSO2SUR.N.7.3"
[5] "MSO2TAC.N.7.3" "MSO2EAC.N.7.3" "MSO2LLA.N.7.3"
```



## Accesando los valores de un objeto: Listas

- Podemos extraer las componentes como explicamos arriba, quizá creando nuevos objetos, y luego aplicarles funciones:

```
> sta_ursula <- so2_df$MSO2SUR.N.7.3
```

```
> head(sta_ursula, 20)
```

```
[1] 0.013 0.023 0.015 0.010 0.007 0.006 0.005 0.004
```

```
[9] 0.008 0.008 0.004 0.003 0.003 0.002 0.002 0.002
```

```
[17] 0.002 0.002 0.002 0.002
```

```
> summary(sta_ursula)
```

| Min.      | 1st Qu.   | Median    | Mean      | 3rd Qu.   | Max.      | NAs      |
|-----------|-----------|-----------|-----------|-----------|-----------|----------|
| 0.000e+00 | 2.000e-03 | 5.000e-03 | 5.977e-03 | 7.000e-03 | 1.100e-01 | 1.920e+0 |

```
> length(sta_ursula)
```

```
[1] 5112
```

```
> mode(sta_ursula)
```

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

more . . .

- Este conjunto de datos se basa en información de pólizas anuales de vehículos contratadas en 2004 o 2005. Hay 67856 pólizas, de las cuales que 4624 (6.8%) tuvieron al menos una reclamación.

EJERCICIO: Leer el archivo `car.csv`, cambiar de nombre a las variables y extraer subconjuntos de información de la variable *valor del vehiculo*.

## Accesando los valores de un objeto

En resumen:

- La estructura de datos más común en R es la lista.
- Los conjuntos de datos son listas de clase *dataframe*, y es común que los resultado de los análisis sean listas con distintas clases
- Las funciones de R muchas veces son polimórficas: dan resultados distintos según la clase de los objetos en los que se evalúan.
- Cuando queremos extraer de objetos que tienen modo de lista, se usa la notación de \$ mostrada arriba (¡aunque no es la única manera, y no siempre es la mejor!)

# Funciones

- R tiene funciones especiales para matrices, por ejemplo `solve()` para invertir, `qr()` para la descomposición QR, `eigen()` para obtener los eigenvalores y eigenvectores, `svd()` para obtener la descomposición de valor singular, etc.
- En R uno puede encontrar:
  - Funciones matemáticas básicas: `log`, `exp`, `log10`, `log2`, `sin`, `cos`, `tan`, `asin`, `acos`, `abs`, `sqrt`, etc.
  - Funciones especiales: `gamma`, `digamma`, `beta`, `besselI`, ...
  - Funciones estadísticas: `mean`, `median`, `lm`,...
  - Algunas otras funciones como: `sum(x)`=suma de los elementos de `x`, `max(x)`, `min(x)`, `which`, `which.max`, etc.

more . . .



# Creando tus propias funciones

- Una función en **R** puede tener cualquier número de argumentos y las operaciones que esta realice pueden ser producto de expresiones en **R**.
- La sintaxis general para la definición de una función es:

```
function(arguments){expression}
```

donde **arguments** son los argumentos de la función separados por comas y **expression** es cualquier estructura permitida en **R**. El valor de la última línea dentro de la estructura **expression** será el valor que retorne la función.

## Creando tus propias funciones

### Ejemplo 1.

- La siguiente función retorna la suma de los cuadrados de los elementos del vector **x**.

```
> myfunction<-function(x){ sum(x*x) }
```

- La función **myfunction** ahora puede ser usada de la siguiente manera:

```
> z<-1:50  
> y<-myfunction(z)  
> y  
[1] 42925
```

## Creando tus propias funciones

### Ejemplo 2.

- La siguiente función realiza cierta operación en cada punto de una malla (“grid”) de valores.

```
> grid.calc<-function(x,y){  
+       # Esta funcion calcula sqrt(x*x+y*y)  
+       # en cada punto del grid definido por x y y.  
grid<-matrix(0,length(x),length(y))  
       # Define la matriz para almacenar los resultados.  
+   for(i in 1:length(x)){  
       for(j in 1:length(y))  
           grid[i,j]<-sqrt(x[i]*x[i]+y[j]*y[j])  
+   }  
grid  
}
```



## Creando tus propias funciones

### Ejemplo 2 (Continuación ...)

- La función “`grid.calc`” ahora puede ser usada de la siguiente manera:

```
> superficie<-grid.calc(1:3,1:4)
```

```
> superficie
```

|      | [,1]     | [,2]     | [,3]     | [,4]     |
|------|----------|----------|----------|----------|
| [1,] | 1.414214 | 2.236068 | 3.162278 | 4.123106 |
| [2,] | 2.236068 | 2.828427 | 3.605551 | 4.472136 |
| [3,] | 3.162278 | 3.605551 | 4.242641 | 5.000000 |

more . . .





# Contenido

- Loops.
- Funciones.
- Graficando con R.



# loops

- Hasta ahora hemos visto componentes de **R** que producen evaluaciones de expresiones simples. Sin embargo, **R** como es un legítimo lenguaje de programación permite ejecuciones condicionales y la construcción de *loops*.
- Veamos el siguiente ejemplo (Método de Newton para el cálculo de la raíz cuadrada):

```
> y<- 144
> x<-y/2
> while(abs(x*x-y)>1e-10) x<-(x + y/x)/2
> x;x^2
```

Observemos la construcción **while** :

```
while(condition){expression}
```

Lo anterior indica que **expression** debe ser evaluada hasta que **condition** sea **TRUE**.



## loops

- Hay que notar que en el loop anterior la condición aparece al inicio del ciclo así que la expresión puede nunca ocurrir.
- Una variación del mismo algoritmo, donde la condición a cumplir aparezca al final del loop, es a través de la estructura **repeat**:

```
> x<-y/2
> repeat{
+ x<-(x + y/x)/2
+ if(abs(x*x-y) < 1e-10) break
+ }
> x
```

- Lo anterior ilustra tres estructuras de control: i) una **expresión compuesta**, ii) otra estructura condicional **if** y iii) **break**.



## loops

- La estructura de ciclo más frecuente es **for**, este *loop* se mantiene para un conjunto fijo de valores.

```
> x
[1] 12
> for(k in 1:10){
+ x<-x+2
+ }
> x
```

```
w<-c(1:10)*0
> for(k in 1:10){
+ w[k]<-w[k]+k }
> w
```



# Funciones

- R tiene funciones especiales para matrices, por ejemplo `solve()` para invertir, `qr()` para la descomposición QR, `eigen()` para obtener los eigenvalores y eigenvectores, `svd()` para obtener la descomposición de valor singular, etc.
- En R uno puede encontrar:
  - Funciones matemáticas básicas: `log`, `exp`, `log10`, `log2`, `sin`, `cos`, `tan`, `asin`, `acos`, `abs`, `sqrt`, etc.
  - Funciones especiales: `gamma`, `digamma`, `beta`, `besselI`, ...
  - Funciones estadísticas: `mean`, `median`, `lm`, ...
  - Algunas otras funciones como: `sum(x)`=suma de los elementos de `x`, `max(x)`, `min(x)`, `which`, `which.max`, etc.



# Creando tus propias funciones

- Es posible escribir nuestras propias funciones en R. De hecho, esta es una de las características más atractivas ("a la larga") de trabajar con R.
- Una función en R puede tener cualquier número de argumentos y las operaciones que esta realice pueden ser producto de varias expresiones en R.
- La sintaxis general para la definición de una función es:

```
function(arguments){expression}
```

donde **arguments** son los argumentos de la función separados por comas y **expression** es cualquier estructura permitida en R. El valor de la última línea dentro de la estructura **expression** será el valor que retorne la función.





## Creando tus propias funciones

### Ejemplo 1.

- La siguiente función retorna la suma de los cuadrados de los elementos del vector **x**.

```
> myfunction<-function(x){ sum(x*x) }
```

- La función **myfunction** ahora puede ser usada de la siguiente manera:

```
> z<-1:50
```

```
> y<-myfunction(z)
```

```
> y
```

```
[1] 42925
```



## Editando una función

- Una ventaja de usar funciones para realizar nuestro trabajo es que estas se pueden editar. No se necesita volver a escribir nuevamente la función.
- Las funciones `fix()` y `edit()` se utilizan para editar una función.

### Ejemplo 2.

```
> my.f<-function(){32}
> my.f()
> fix(my.f)          # Cambiar 40 por 32
> my.f()
> edit(my.f)         # Cambiar 60 por 40
> my.f()             # Qué observas?
>                   # Qué propones?
```

- Se puede notar que la función `edit()` trabaja de manera similar a `fix()`, sólo que la primera requiere de la asignación de su valor.

## El cuerpo de una función



- El cuerpo de una función es un bloque de comandos encerrados entre ‘‘llaves’’. El valor de la función será el último valor ejecutado.
- **Cuidado:**
  - La asignación dentro de una función es diferente.
  - Cuando se declara o crea una función, está posee su propio “ambiente” de trabajo el cual está subordinado al “ambiente” de trabajo global o general.

### Ejemplos 3 y 4

```
> x<-5
> mf<-function(){
+ x<-6
+ x }
> mf()
[1] 6
> x
[1] 5
```

```
> x<-5
> mf<-function(){print(x)}
> mf()
[1] 5
> rm(x)
> mf()
Error en print(x) :
objeto x no encontrado
```



## Archivos script y `source()`

- R puede leer el contenido de un archivo y ejecutar los comandos como si estos fueran escritos en la línea de comandos.
- El comando correspondiente es `source()`:

```
source(file="k:\\funciones.R")
```

## Ejemplo

Construya un archivo `programa1.R` que incluya con los siguientes comandos:



```
# Programa: programa1.R
#
f.sd <- function(n){
w1<-rnorm(n)
sd(w1)
}
f.iqr <- function(n){
w2<-rnorm(n)
IQR(w2)
}
dat1<-c(1:500)*0.0
dat2<-c(1:500)*0.0

for(i in 1:500){
dat1[i]<-f.sd(100)
dat2[i]<-f.iqr(100)
boxplot(dat1,dat2)
}
```



## Creando tus propias funciones

### Ejemplo 2.

- La siguiente función realiza cierta operación en cada punto de una malla (“grid”) de valores.

```
> grid.calc<-function(x,y){  
+       # Esta funcion calcula sqrt(x*x+y*y)  
+       # en cada punto del grid definido por x y y.  
grid<-matrix(0,length(x),length(y))  
       # Define la matriz para almacenar los resultados.  
+   for(i in 1:length(x)){  
       for(j in 1:length(y))  
           grid[i,j]<-sqrt(x[i]*x[i]+y[j]*y[j])  
+   }  
grid  
}
```



## Creando tus propias funciones

### Ejemplo 2 (Continuación ...)

- La función “`grid.calc`” ahora puede ser usada de la siguiente manera:

```
> superficie<-grid.calc(1:3,1:4)
```

```
> superficie
```

|      | [,1]     | [,2]     | [,3]     | [,4]     |
|------|----------|----------|----------|----------|
| [1,] | 1.414214 | 2.236068 | 3.162278 | 4.123106 |
| [2,] | 2.236068 | 2.828427 | 3.605551 | 4.472136 |
| [3,] | 3.162278 | 3.605551 | 4.242641 | 5.000000 |



more . . .

• .





# Graficando con R

- R ofrece una gran variedad de gráficos, aunado a la posibilidad y flexibilidad de crearlos y personalizarlos.
- Para tener una idea de las gráficas que ofrece R se puede ejecutar el siguiente comando:

```
> demo(graphics)
```

- Sería difícil exponer en esta presentación todas las opciones y posibilidades que ofrece R en terminos gráficos. De manera particular, cada función gráfica tiene un gran número de opciones (argumentos), lo que resulta en una amplia flexibilidad en la construcción de gráficos.

# Gráficos en R



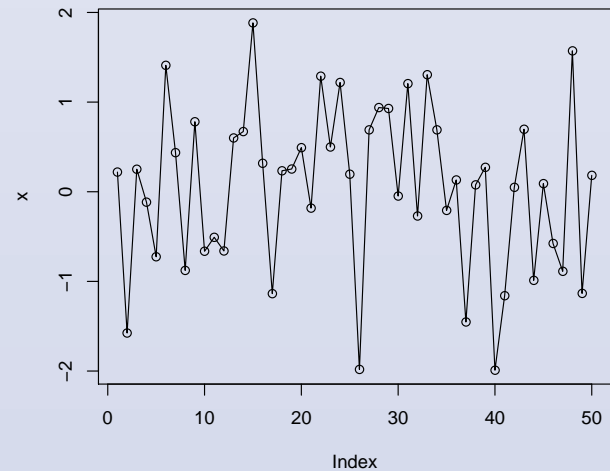
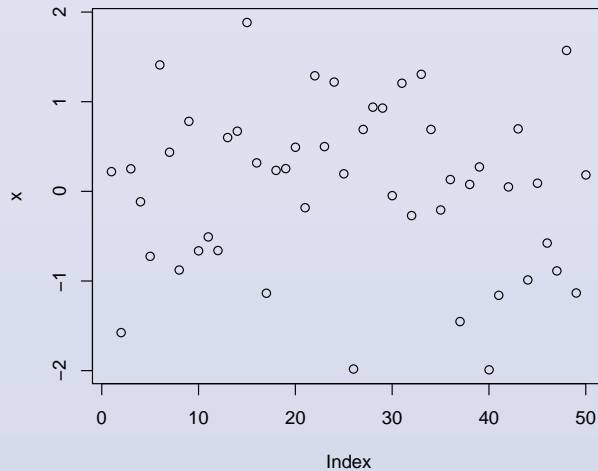
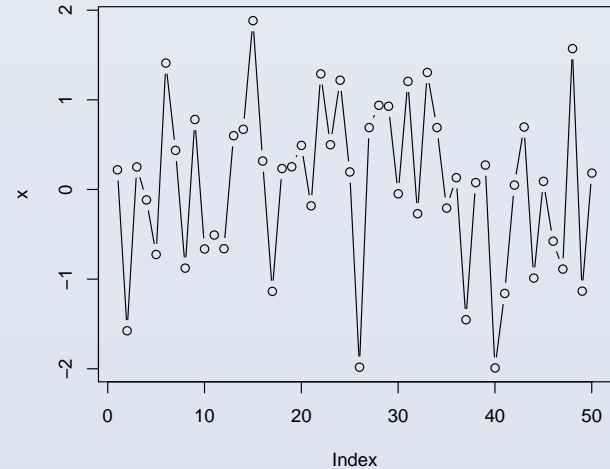
- Notemos la flexibilidad de las funciones gráficas.

```
> x<-rnorm(50)
```

```
> plot(x)
```

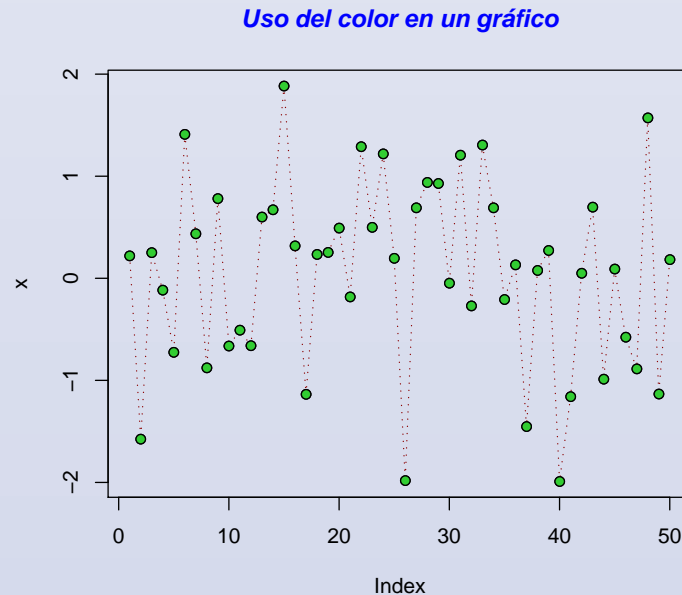
```
> plot(x,type="b")
```

```
> plot(x,type="o")
```



- Uso del color en los gráficos.

```
>plot(x)
>lines(x, col = "red4", lty = "dotted")
>points(x, bg="limegreen", pch = 21)
>title(main = "Uso del color en un grafico",cex.main =
+ 1.2,font.main = 4,col.main = "blue")
```

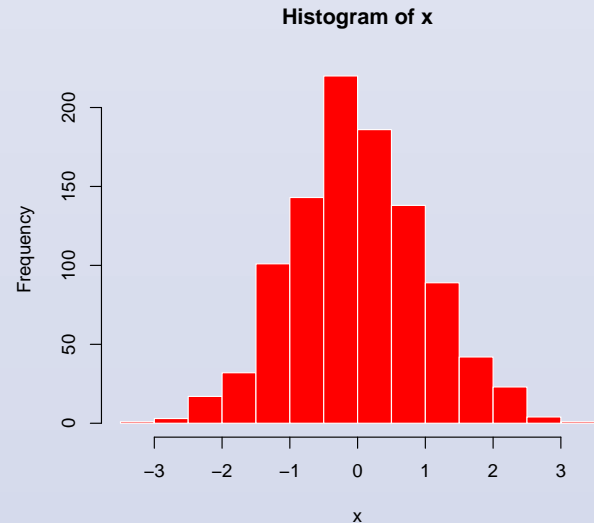
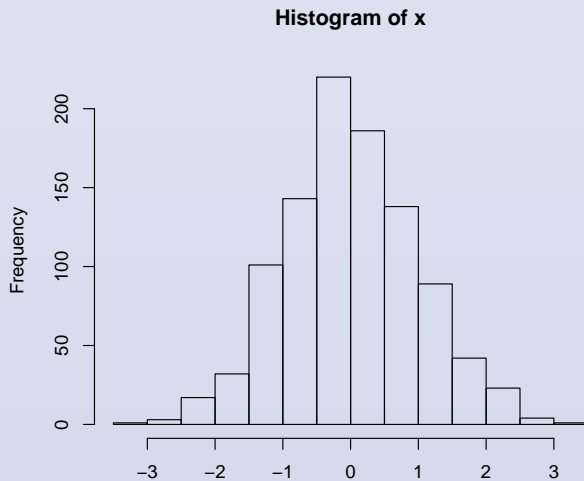
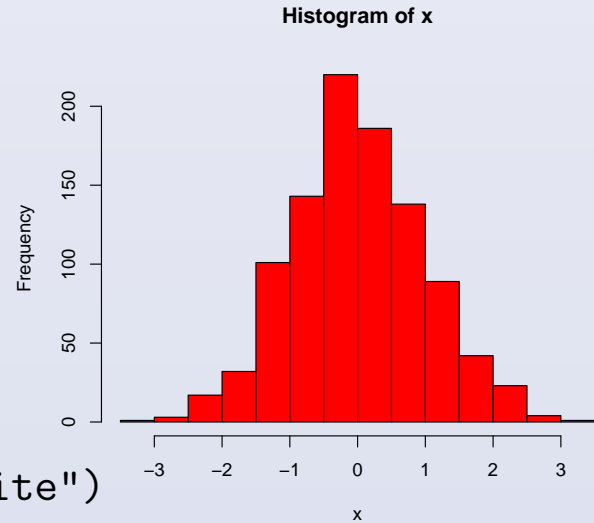


# Gráficos en R



- Continuemos con la flexibilidad de las funciones gráficas.

```
> x<-rnorm(1000)
>
>
> hist(x)
> hist(x,col="red")
> hist(x,col="red",border="white")
```

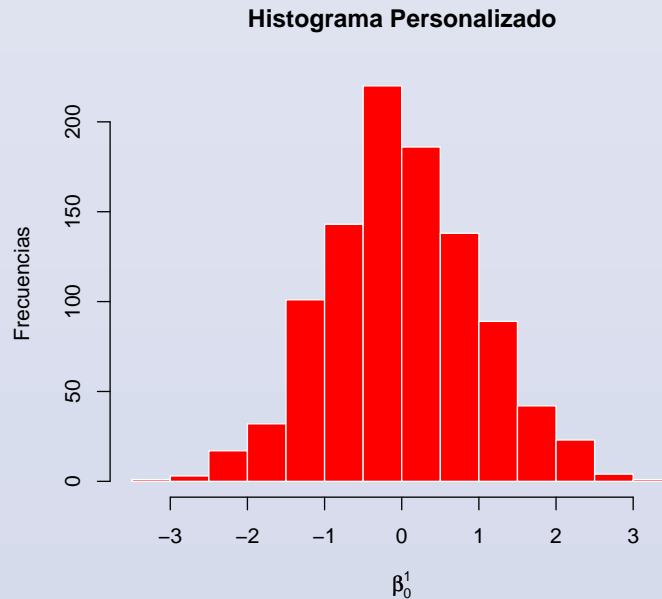


## Gráficos en R



- Con los parámetros gráficos **xlab**, **ylab** y **main** uno puede agregar etiquetas a los ejes  $X$ ,  $Y$ , y darle un título al gráfico, respectivamente.

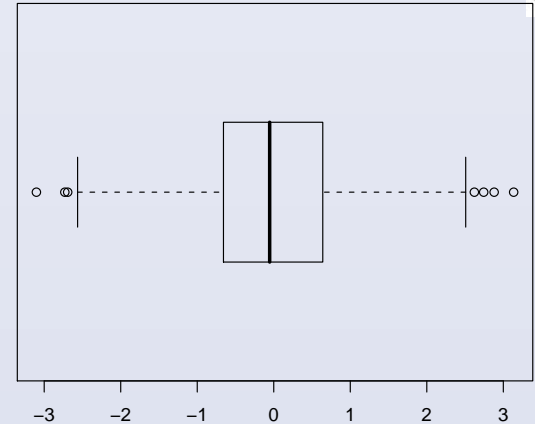
```
> hist(x,col="red",border="white",xlab=expression(beta[0]^1),  
+ ylab="Frecuencias",main="Histograma Personalizado")
```



## Más Gráficos

- En R se pueden graficar diagramas de caja y brazo, de tallo y hoja, distribuciones discretas de probabilidad, etc.

```
> boxplot(x, horizontal=T)
> stem(x[1:50])
```



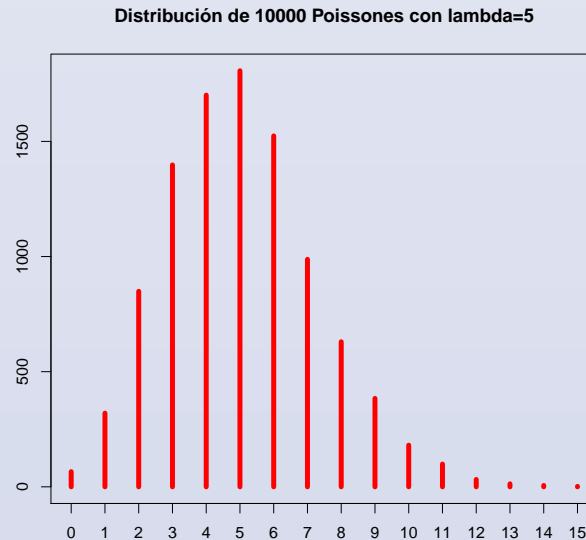
The decimal point is at the |

```
-3 | 1
-2 |
-1 | 72110
-0 | 99998544333332222110
 0 | 2222234456678889
 1 | 011356
 2 | 2
```



## Más Gráficos

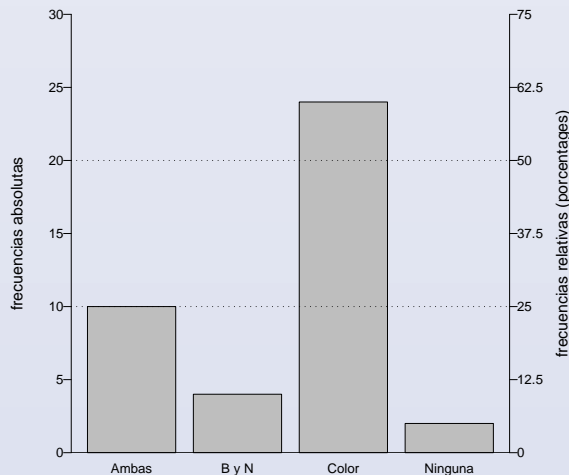
```
> plot(table(rpois(10000,5)), type = "h", col = "red", lwd=5,  
+ main="Distribucion de 10000 Poissones con lambda=5",ylab="")
```



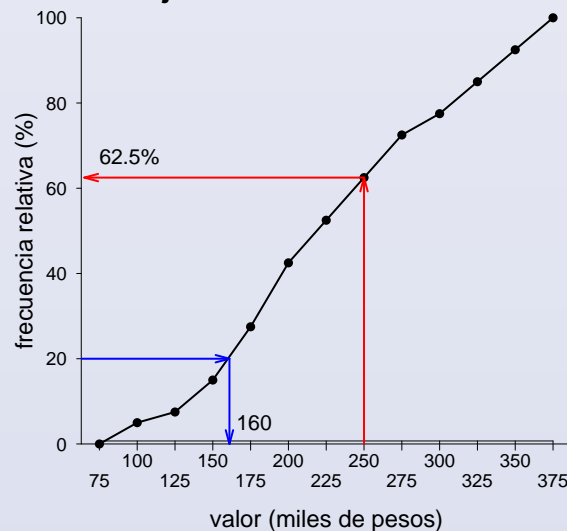
# Gráficos Personalizados



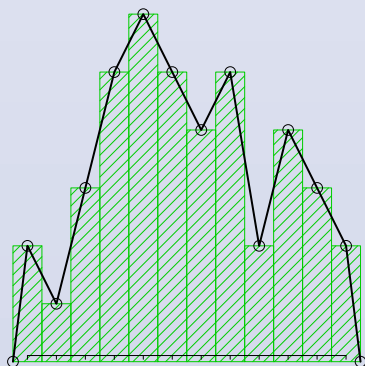
Distribucion de Tipo de Television por Colonia (porcentajes)



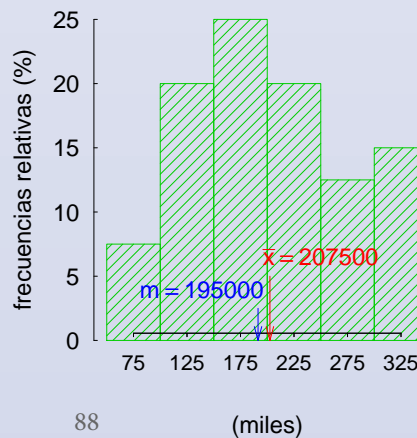
Ojiva de la variable <valor>



Histograma y poligono de frecuencias



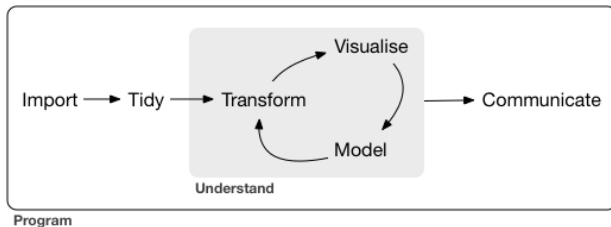
Medidas de tendencia central





# Ciencia de Datos

- ▶ LA CIENCIA DE DATOS es una **disciplina** emocionante que nos permite trabajar con datos para su entendimiento y la generación de **conocimiento**.
- ▶ Un modelo de las herramientas necesarias en un proyecto de Ciencia de Datos:



# Ciencia de Datos

- ▶ Big Data
- ▶ Julia, Python y amigos
- ▶ Datos rectangulares

# Ciencia de Datos

- ▶ Big Data
- ▶ Julia, Python y amigos
- ▶ Datos rectangulares

**Programación en R:** *Hands-on Programming with R*  
(Garrett Grolemund)

# Ciencia de Datos

- ▶ Big Data
- ▶ Julia, Python y amigos
- ▶ Datos rectangulares

## **Programación en R:** *Hands-on Programming with R* (Garrett Grolemund)

### *Prerequisitos:*

- ▶ R ( <https://cloud.r-project.org> ),
- ▶ RStudio ( <http://www.rstudio.com/download> )
- ▶ R paquetes: tidyverse, ggplot2, etc,...

# Ciencias de Datos: Consiguiendo ayuda y reproduciendo los análisis

- ▶ Internet:

<https://blog.rstudio.com>

<https://www.r-bloggers.com>,

etc, . . .

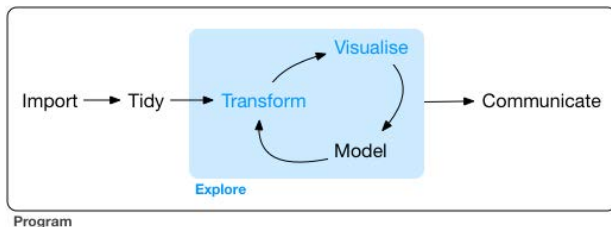
- ▶ stackoverflow ( <https://stackoverflow.com> )

*Se requieren 3 cosas para reproducir los análisis:*

- ▶ Los R-paquetes
- ▶ Los Datos
- ▶ Código (script)

# Exploración de Datos

**Exploración de Datos:** Mirar los datos, generar hipótesis rápidamente, probarlas y repetir una y otra vez el ciclo.



- ▶ Visualización de Datos
- ▶ Transformación de Datos
- ▶ Análisis Exploratorio de Datos

# Exploración de Datos: Requisitos

- ▶ ggplot2: “The Layered Grammar of Graphics”,  
<http://vita.had.co.nz/papers/layered-grammar.pdf>
- ▶ tidyverse

```
install.packages("tidyverse")  
library(tidyverse)
```

```
library(ggplot2)
```

# Exploración de Datos

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))
```

