R: Un ambiente y lenguaje para el Cálculo y la Graficación Estadística

Gabriel Nuñez Antonio
ITAM

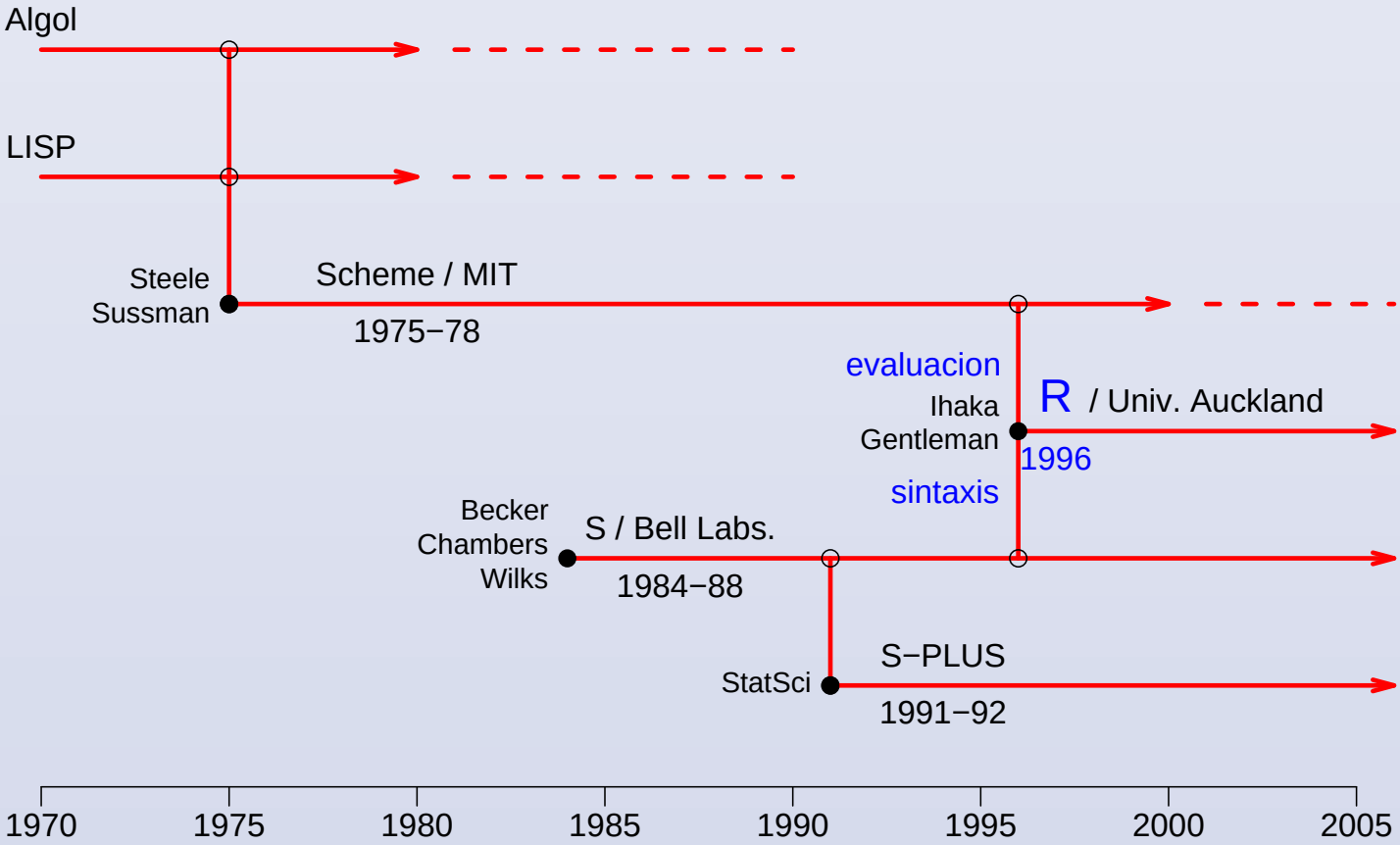
Ernesto Barrios Zamudio
ITAM

XXI Foro Nacional de Estadística
Acapulco, 2006

Contenido

- Historia.
- Introducción.
- Manipulación de Datos.
- Gráficos.
- Análisis Estadístico.

Genealogia de R



Artículo Original

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.

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 desarollo de R y proyectos de código abierto relacionados.

Introducción

- R ofrece una gran cantidad de funciones para realizar análisis gráfico y estadístico.
- Inicialmente se podría pensar que R es demasiado complejo para los no especialistas. Esto está muy lejano de la realidad, ya que una característica importante de R es su flexibilidad
- 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 letras, digitos 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, esta 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.)

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
```

- Si un objeto ya existe, su valor anterior se elimina.

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"
>
```

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

```
>help(rm)
>?rm
```

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"
```

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
>
```

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)
```

Generación de Datos

Secuencias Regulares

- Algunas susesiones 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
```

- De manera directa con ayuda de la función `c()`.

```
> z<-c(5,2,3.5,26,3,5,-56,8)
> z
[1]   5.0   2.0   3.5  26.0   3.0   5.0 -56.0   8.0
```

Generación de Datos

- 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)
```

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
[1]  11  12  13 101  -4  -5
```

$\star$ Hay que tener cuidado con las logitudes de los vectores en las operaciones.

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(1:6,2,3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
> matrix(0,ncol=3,nrow=2)
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
```

```
> 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

* 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
```

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
```

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), wich, wich.max`, etc.

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,]
> Mrow1
[1]  1  5  9 13 17
>
> M[,2]
[1] 5 6 7 8
```

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   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

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

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

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

```
> x<-Msub[1,2]
> x
[1] 5
```

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
                  # obtener un vector de ceros (de longitud 10)
> Y
 [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?
[1]  2 20
```

Accesando los valores de un objeto

- 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
```

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){ asq<-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
```

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 ejecuta el siguiente comando:

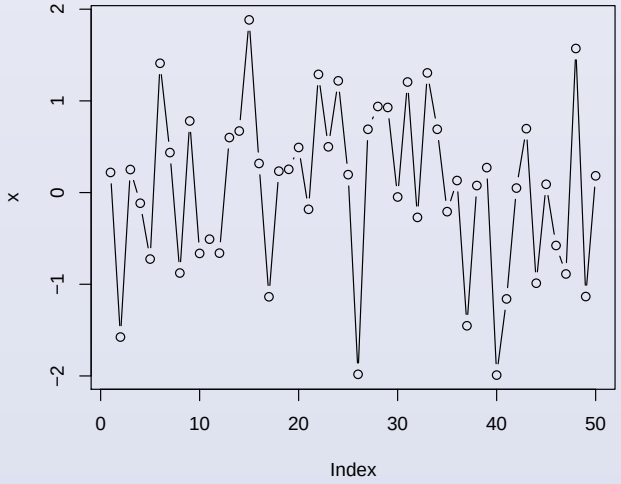
```
> demo(graphics)
```

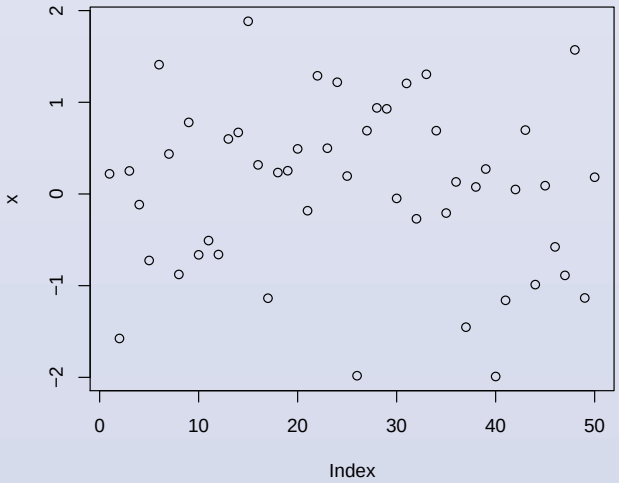
- Sería difícil exponer en esta presentación todas las opciones y posibilidades que ofrece R en terminos gráficos. De manera particular, cada función gráfica tiene un gran número de opciones (argumentos), lo que resulta en una amplia flexibilidad en la construcción de gráficos.

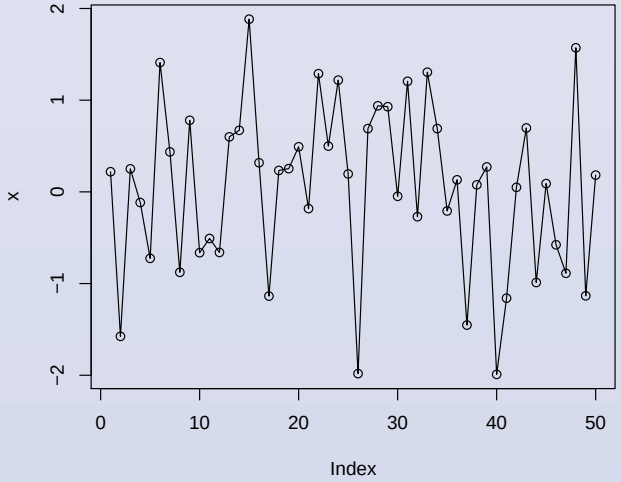
Gráficos en R

- Notemos la flexibilidad de las funciones gráficas.

```
> x=rnorm(50)
> plot(x)
> plot(x,type="b")
> plot(x,type="o")
```



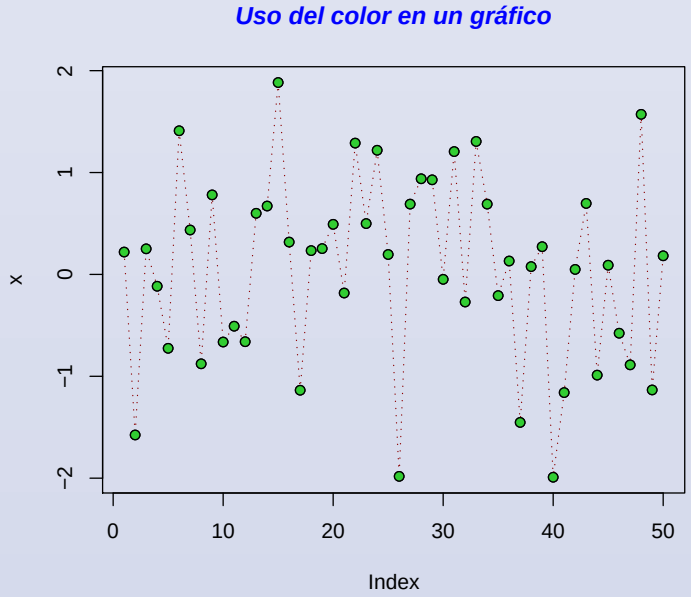




Gráficos en R

- Uso del color en los gáficos.

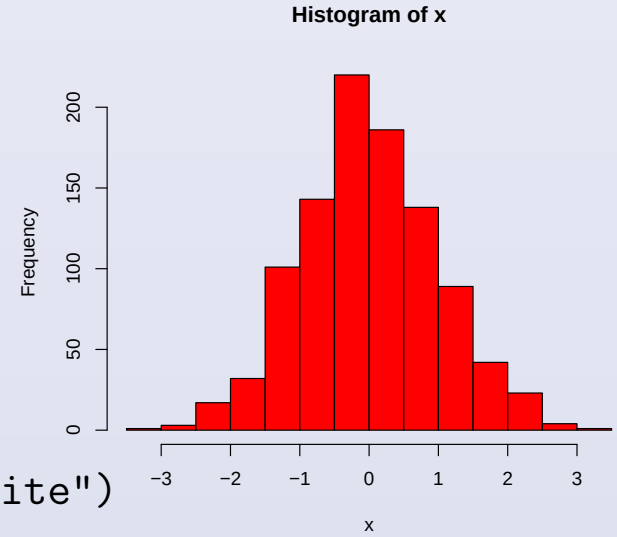
```
>plot(x)
>lines(x, col = "red4", lty = "dotted")
>points(x, bg="limegreen", pch = 21)
>title(main = "Uso del color en un  grafico",cex.main =
+ 1.2,font.main = 4,col.main = "blue")
```

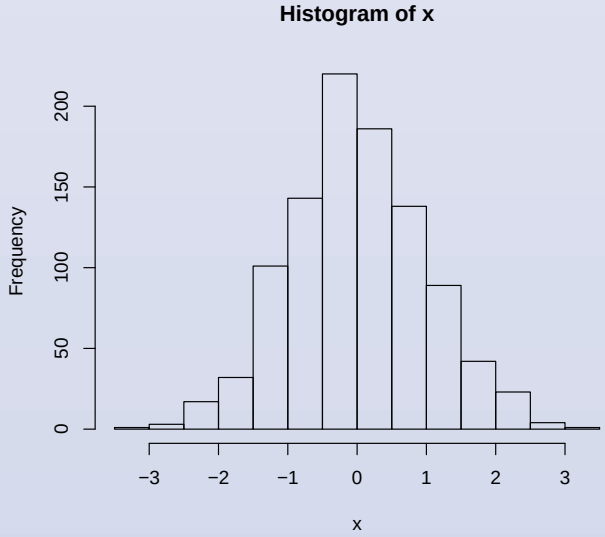


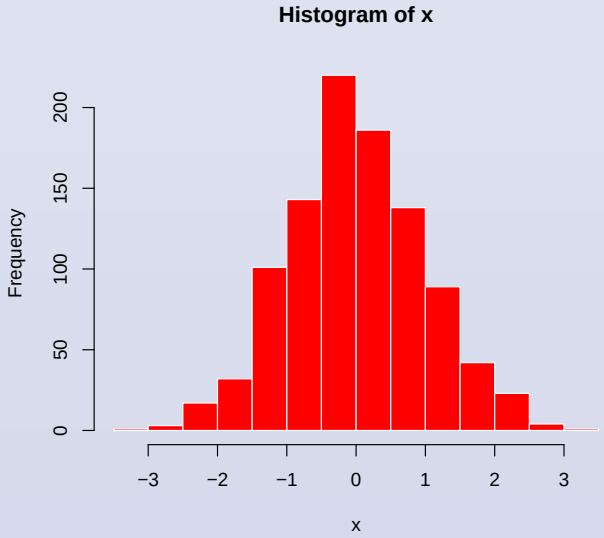
Gráficos en R

- Continuemos con la flexibilidad de las funciones gráficas.

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



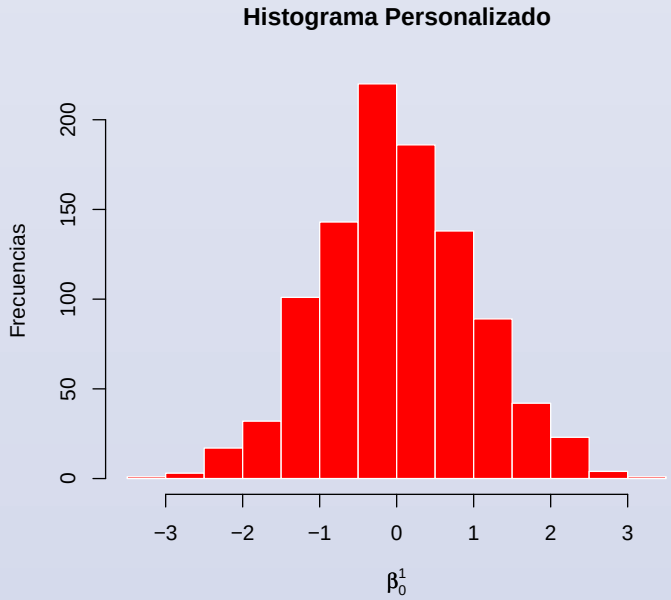




Gráficos en R

- Con los parámetros gáficos `xlab`, `ylab` y `main` uno puede agregar estiquetas a los ejes X, Y, y darle un título al gráfico, respectivamente.

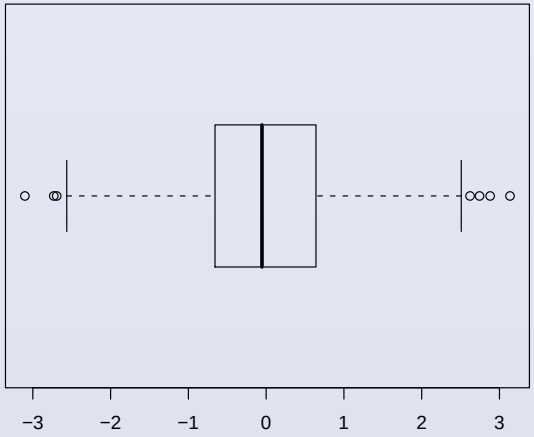
```
> hist(x,col="red",border="white",xlab=expression(beta[0]^1),
+ ylab="Frecuencias",main="Histograma Personalizado")
```



Más Gráficos

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

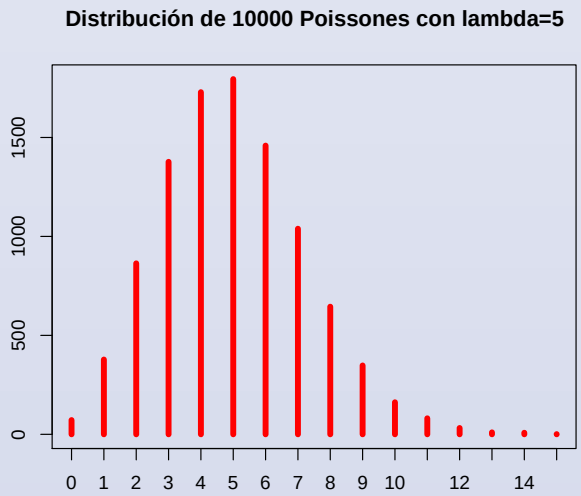
```
> boxplot(x,horizontal=T)
> stem(x[1:50])
```



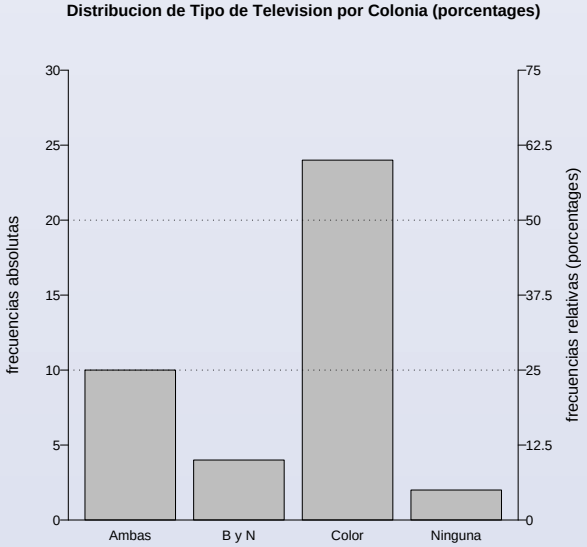

```
  The decimal point is at the |
-3 | 1
-2 |
-1 | 72110
-0 | 99998544333332222110
 0 | 22222234456678889
 1 | 011356
 2 | 2
```

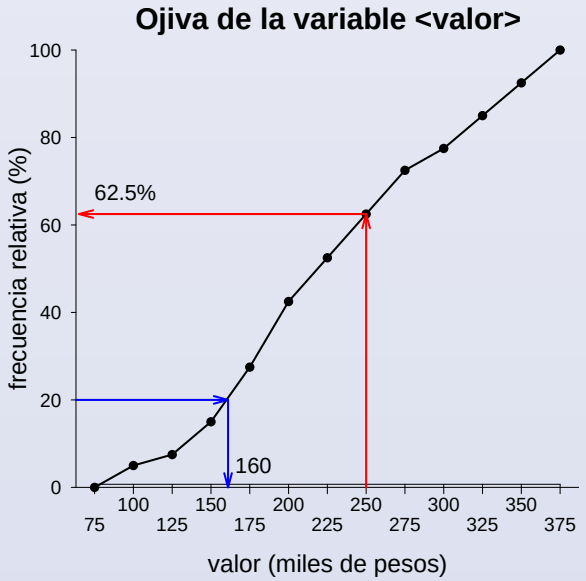
Más Gráficos

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

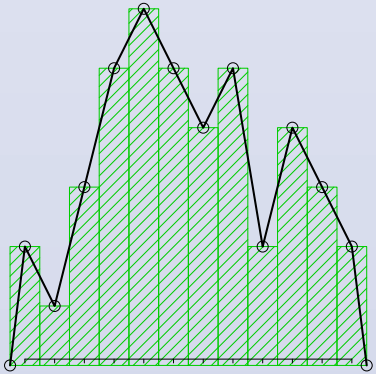


Gráficos Personalizados

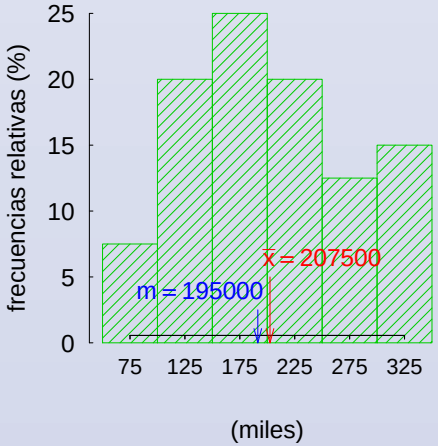




Histograma y poligono de frecuencias



Medidas de tendencia central



Análisis Estadísticos usando R

- R no sólo ofrece una gran flexibilidad en la construcción de gráficos, también ofrece una amplia gama de posibilidades para realizar análisis estadísticos (tanto descriptivos como inferenciales). A continuación se muestran sólo algunos ejemplos.

Ejemplo 1

```
>datos<-rnorm(100, 2, 4)

#Muestra de 100 observaciones normales con media 2 y desv. est.
```

- La función `summary()` calcula algunas estadísticas descriptivas.

```
>summary(datos)

   Min.   1st Qu.  Median    Mean   3rd Qu.    Max.
-9.5900  -0.7551   1.4080   1.5000  3.8160    10.7
```

Ejemplo 2

- A continuación se muestra el ajuste de un modelo de regresión lineal múltiple. Los datos asociados a las variable independientes (x) y a la variable dependiente (y), se pueden leer de una base de datos con ayuda de las funciones `scan()` o `read.tabla()`. Para este ejemplo, se generaron de la siguiente manera.

```
> y=rnorm(10)
> x1=c(1:10)
> x2=c(rep(1,5),rep(0,5))
> x3=rnorm(10)
```

- La función `lm()` es usada para ajustar modelos lineales.

```
>modelo1<-lm(y~x1+x2+x3)   # Se define y ajusta el modelo.
> modelo1
Call: lm(formula = y ~ x1 + x2 + x3)
Coefficients: (Intercept)        x1          x2          x3
                   2.8520     -0.3599     -1.8483      0.3328
```

⋆ Observe de qué tipo es el objeto `modelo1`.

Ejemplo 2 (Continuación ...)

- Con la ayuda de la función genérica `summary()` se pueden obtener algunos resultados resumen del ajuste.

```
>summary(modelo1)
Call: lm(formula = y ~ x1 + x2 + x3)

Residuals:
    Min      1Q  Median      3Q     Max
-0.6709 -0.3335 -0.0218  0.2858  0.7165

Coefficients:
            Estimate    Std. Error  t value  Pr(>|t|)
(Intercept)      2.8520     1.0690   2.668   0.0371 *
x1              -0.3599     0.1305  -2.759   0.0329 *
x2              -1.8483     0.7467  -2.475   0.0481 *
x3               0.3328     0.3616   0.920   0.3930
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 0.5758 on 6 degrees of freedom Multiple
R-Squared: 0.5797,     Adjusted R-squared: 0.3695 F-statistic: 2.758
on 3 and 6 DF,  p-value: 0.1342
```

Ejemplo 2 (Continuación ...)

- Se puede notar que la función `summary()` se puede aplicar tanto a un vector de datos como a un modelo. He aquí la flexibilidad de R.
- La tlabla de ANOVA para un análisis de regresión múltiple se obtiene usando la función `anova()`

```
> anova(modelo1)
```

```
Analysis of Variance Table

Response: y
          Df  Sum Sq   Mean Sq   F value  Pr(>F)
x1         1  0.17703  0.17703   0.5339  0.49250
x2         1  2.28595  2.28595   6.8942  0.03929 *
x3         1  0.28074  0.28074   0.8467
0.39299 Residuals  6 1.98945 0.33158
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Ejemplo 2 (Continuación ...)

- En realidad `modelo1` es un objeto tipo lista, todos sus componentes se pueden obtener con ayuda de la función `names()`.

```
> names(modelo1)
 [1] "coefficients"  "residuals"   "effects"     "rank"
 [5] "fitted.values" "assign"      "qr"          "df.residual"
 [9] "xlevels"       "call"        "terms"       "model"
```

- Así , sus componentes se pueden accesar, por ejemplo, con:

```
> modelo1$residuals
          1           2           3           4           5
-0.23464471  1.18946063 -0.05046907  0.17749660 -1.08184345
          6           7           8           9          10
-0.49368395 -1.58228384  2.06572986 -0.11623393  0.12647186
```

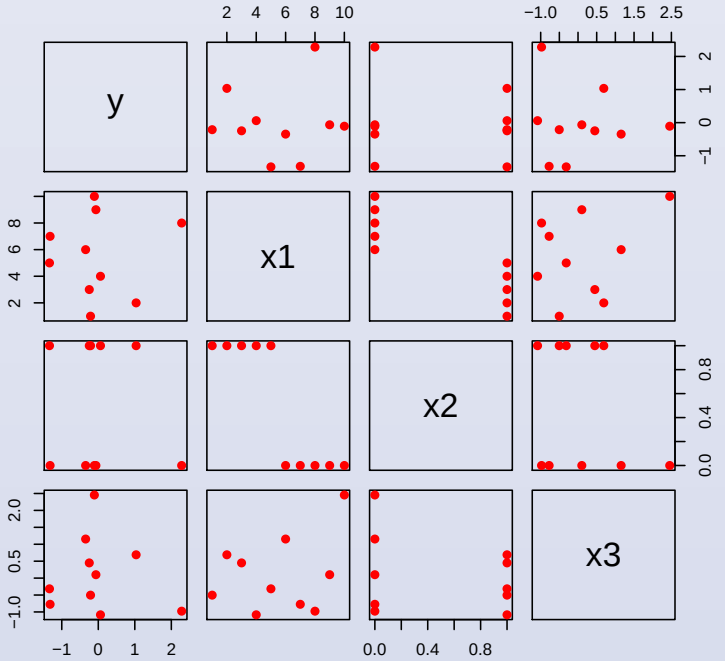
Ejemplo 2 (Continuación ...)

- Se puede notar que la función `plot()` es una función genérica, es decir:

```
>datos=data.frame(y,x1,x2,x3)
> datos
```

	y	x1	x2	x3
1	-0.21193520	1	1	-0.5032437
2	1.03567146	2	1	0.6893553
3	-0.24656601	3	1	0.4493651
4	0.06023359	4	1	-1.0839041
5	-1.33581988	5	1	-0.3160428
6	-0.34656175	6	0	1.1543616
7	-1.31937170	7	0	-0.7734411
8	2.28289129	8	0	-0.9766752
9	-0.06481415	9	0	0.1010843
10	-0.10739500	10	0	2.4550769

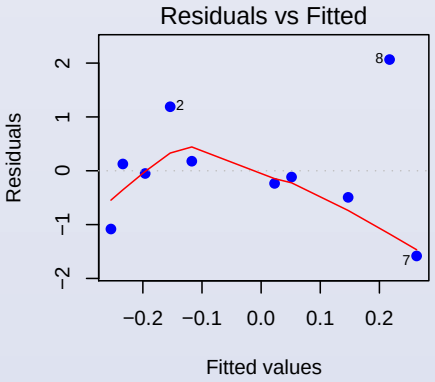
```
>plot(datos)
```

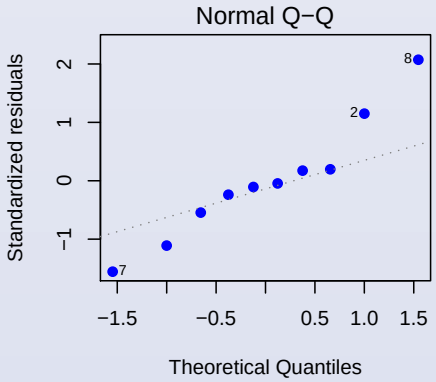


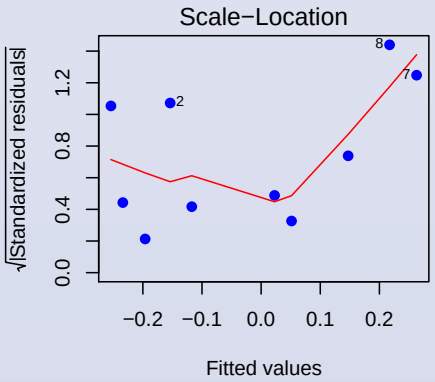
Ejemplo 2 (Continuación ...)

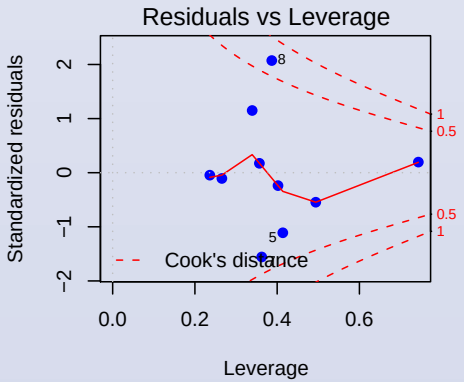
- Ahora, recordando que modelo1 es un objeto de tipo lista, al que se le asigno un ajuste de regresión lineal, ...

```
>plot(modelo1)
```









Análisis Estadísticos usando R

- Pruebas de hipótesis, paramétricas y no paramétricas:

```
t.test(), var.test(), cor()
wilcox.test(), cor.test(datos,method="spearman")
binom.test(), kruskal.test()
⋮
```

- Análisis y ajuste de modelos:

`lm()`	Modelos de regresión
`glm()`	Modelos lineales generalizados
`survfit(), coxph()`	Análisis de supervivencia
`prcomp()`	Análisis multivariado
`hclust(), plclust`	
`density()`	Estimadores de kernel
`ts.plot(), acf()`	Series de tiempo
⋮	

Consideraciones Finales

- Existen más de 865 paquetes, en el sitio web, que ajustan modelos específicos, grafican datos muy particulares (datos direccionales), realizan simulaciones de variables específicas, etc, etc, ...
- R resulta no sólo en una opción sino una opción atractiva para la graficación y el análisis estadístico.
- Aunque en está plática no se muestra, en términos prácticos el manejo de memoria en R es más eficiente que en S-plus y esto es redituable al momento de programar.

Referencias

1. Ihaka, R. y Gentleman, R. (1996). R: A Language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics*, **5**, 3, 299-314.

2. `http://cran.r-project.org`

3. Dalgaard, P. (2002). *Introductory Statistics with R*. Springer-Verlag. New York.

4. Everrit, S. B. (1996). *A handbook of Statistical Analyses using S-PLUS*. Chapman & Hall. New York.