# Notas en R

Herman Jaramillo

3 de abril de 2019

## 1. Introducción

El lenguaje R [1] se basa en el lenguaje S [2] que, a su vez, está basado en Scheme [3] El lenguaje S fue creado por John Chambers en 1976 en Laboratorios Bell y el programa R fue creado por Ross Ihaka y Robert Gentleman (ambos nombres comienzan en R) en la universidad de Auckland en Nueva Zelanda.

Para más información acerca de la historia de R el lector puede referirse a este blog [4]. La siguiente lista muestra algunas de las razones por las cuales R es un lenguaje atractivo para desarrolladores, profesores y científicos.

1.1 R no es un paquete de estadística. Es un lenguaje de programación bajo el paradigma de orientación a objetos

1.2 R es de dominio público (de libre distribución)

1.3 R es una herramienta útil, flexible y robusta para el análisis de datos.

1.4 R incluye exelentes herramientas de visualización

1.5 R tiene millones de usuarios, muchos de ellos contribuyendo activamente con nuevas y útiles librerías

En este documento presentamos elementos básicos de R, comenzando con el uso de R como una calculadora (modo comando) y siguiendo con la descripción de tipos simples y compuestos así como la elaboración de programas básicos. Al final mencionamos algunas herramientas útiles para el desarrollo en R a la vez que introducimos un tutorial elaborado en Jupyter.

[1] https://en.wikipedia.org/wiki/R_(programming_language)
[2] https://en.wikipedia.org/wiki/S_(programming_language)
[3] https://en.wikipedia.org/wiki/Scheme_(programming_language)
[4] https://statfr.blogspot.com/2018/08/r-generation-story-of-statistical.html

## 2. R como calculadora

A diferencia de Python, R presenta funciones nativas (sin necesidad de importar paquetes) útiles, tales como exponenciales, trignométricas, logarítmicas, etc.

```
> 2+3 # suma
[1] 5
> 3*3 # multiplicación
[1] 9
> 2^(0.5) # exponenciación decimal (raíz cuadrada en este caso. También se puede sqrt(2))
[1] 1.414214
> log2(2) # logaritmo en base 2
[1] 1
> log10(11) # logaritmo en base 10
[1] 1.041393
> log(11) # logaritmo natural (base e )
[1] 2.397895
> pi # valor de la constante pi
[1] 3.141593
> cos(pi) # función coseno
[1] -1
> sin(pi)  # función seno
[1] 1.224647e-16
> exp(1) # función exponencial
[1] 2.718282
> tan(0.3) # función tangente
[1] 0.3093362
> quit()
Save workspace image? [y/n/c]: y
```

La sesión se puede salvar respondiento “y” . Automáticamente la historia queda registrada en el archivo `.Rhistory` de tipo

```
:R>file .Rhistory
.Rhistory: UTF-8 Unicode text
```

Para evitar la pregunta se puede salir con el comando `quit("yes")` o `quit("no")`. También podemos salirnos con el comando q(). La opción `c` signifca " cancelar "(seguir dentro del ambiente `R` ). Si, luego de salir del ambiente `R` ejecutamos el comando en Linux `ls -la` observamos que existe un archivo con el nombre `.Rhistory` . El archivo `.Rhistory` a se puede recuperar con el comando `loadhistory()`. Podemos revisar la historia como se muestra a continuación

```
> loadhistory(file=".Rhistory")
> history()

2+3 # suma
3*3 # multiplicación
2^(0.5) # exponenciación decimal (raíz cuadrada en este caso. También se puede sqrt(2))
log2(2) # logaritmo en base 2
log10(11) # logaritmo en base 10
log(11) # logaritmo natural (base e )
```

```
pi # valor de la constante pi
cos(pi) # función coseno
sin(pi)  # función seno
exp(1) # función exponencial
tan(0.3) # función tangente
quit()
history()
```

Podemos verificar que, si el archivo de historia se llama `.Rhistory` no es necesario cargarlo. El sistema lo hace por defecto. La historia se lista simplemente con el comando `history()`.

Para entender más sobre la instrucción `history()` podemos usar la ayuda dentro de R . Por ejemplo miremos las siguientes líneas y la salida (se sale del diálogo con "q" )

```
> a=2
> b=3
> c=a+b
> history()


2+3 # suma
3*3 # multiplicación
2^(0.5) # exponenciación decimal (raíz cuadrada en este caso)
log2(2) # logaritmo en base 2
log10(11) # logaritmo en base 10
log(11) # logaritmo natural (base e )
pi # valor de la constante pi
cos(pi) # función coseno
sin(pi)  # función seno
exp(1) # función exponencial
tan(0.3) # función tangente
quit()
a=2
b=3
c=a+b
history()
>?history()
```

Cuando queramos averiguar sobre alguna función de R podemos simplemente escribir una interrogación adelante del numbre de la función (incluidos los paréntesis () ) o usar la sintaxis `help(funcion)` (sin paéntesis, en este caso). Pruebe con

```
> help(history)
```

También podemos escribir el comando `help()` para abrir la forma como funciona el manual dentro de R.

Cuando corrimos el comando `history()` observamos que se almacenan los últimos 25 comandos. Lo mismo que en Linux existe una variable ambiente llamada `R_HISTSIZE` con la cual podemos controlar en número de comandos que recuerde el comando `history()`. Usemos los comandos en Linux

```
LinuxPrompt> rm .RData
LinuxPrompt> rm .Rhistory
```

para limpiar la memoria de R. Ejecutemos ahora los siguentes comandos simples en R.

```
> a=2
> b=3
> c=a+b
> quit("yes")
```

Corramos ahora el comando en Linux `ls -lt` y revisemos los archivos que comienzan con `.R`

```
> a=2
> b=3
> c=a+3
> quit("yes")
:R>ls -la .R*
-rw-r--r-- 1 herman herman 74 Mar 24 15:48 .RData
-rw------- 1 herman herman 26 Mar 24 15:48 .Rhistory
```

Además del archivo `.Rhistory` se creó también el archivo `.RData`. El archivo `.RData` no se creó en la primera lista de comandos por que no había nada que almacenar. Se manejó R como una calculadora y no se guardó nada en memoria. Ahora ya se guardaron algunas variables en memoria y podemos recuperarlas como mostramos enseguida.

```
> load(".RData")
> ls()
[1] "a" "b" "c"
> print(a)
[1] 2
```

Note que el comando `ls` lista las variables en memoria. Todas las variables fueron recuperdas con sus nombres y sus valores. Se podría haber ejecutado el comando `object()` en vez de `ls()` . Ambas funciones son equivalentes. También podríamos salvar el archivo de datos (variables y su contenido) usando el comando `save.image("archivo")`. Por ejemplo

```
> a=2
> b=3
> c=a+b
> save.image("prueba.RData")
> quit("no")
```

Recobramos el archivo con los comandos

```
> load("prueba.RData")
> ls()
[1] "a" "b" "c"
> print(a)
[1] 2
```

Por último vemos que podemos obtener ayuda acerca de instrucciones de programación con el comando `help("instruccion")` . Por ejemplo

```
>help("for")
Control                  package:base                  R Documentation

Control Flow

Description:

These are the basic control-flow constructs of the R language.
They function in much the same way as control statements in any
Algol-like language.  They are all reserved words.

Usage:

if(cond) expr
if(cond) cons.expr  else  alt.expr

for(var in seq) expr
while(cond) expr
repeat expr
break
next

Arguments:
```

En este caso buscamos el manual para la instrucción `for`. Nos salimos del diálogo con la letra "q". Tambíen pudimos usar en vez de `help("for")`el comando `?"for"` y obtener el mismo resultado.

# 3. Programación en R

## 3.1. Programación Sin Interface Gráfica

El programa más simple es el programa `hola_mundo.R` que listamos a continuación

hola_mundo.R

```
print(``Hola Mundo'')
```

El programa se corre con el comando `Rscript`. Por ejemplo

```
:Code> Rscript hola_mundo.R
[1] "Hola Mundo"
```

Un programa elemental pero un poco más complejo es el programa `cuadrados.R` que imprime los 10 primeros cuadrados de enteros positivos.

cuadrados.R

```
print(``Calculo de los primeros 10 cuadrados'')
for(i in 1:10)
  print(i^2)
```

y el resultado es

```
:Code>Rscript cuadrados.R
[1] "Cálculo de los primeros 10 cuadrados"
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
[1] 49
[1] 64
[1] 81
[1] 100
```

A continuación extendemos los elementos de programación para construir programas de mayor complejidad.

## 3.2. Parte Analítica

Estudiaremos la definición de tipos simples y compuestos (nativos e importados). Luego las instrucciones básicas de programación y terminamos con la parte gráfica de R. La mayoría de este contenido se muestra en un cuaderno elaborado con la plataforma Jupyter.

### 3.2.1. Variables

Estudiamos las variables de tipo: Numericas, Alfabeticas, Booleanas.

### 3.2.2. Operadores

Presentamos los operadores aritméticos, relacionales y lógicos en las tres tablas a continuación.

### 3.2.3. Tipos de Datos

El tipo de datos lo mostramos en detalle en la siguiente sección que es un tutorial en R mediante la plataforma Juypter. Solo, a manera de introducción presentamos los vectores y algunas funciones de fácil uso.

| Operadores Aritméticos | |
|---|---|
| Nombre | Símbolo |
| Suma | + |
| Resta | - |
| Multiplicación | $*$ |
| División | / |
| Exponenciación | $\wedge$ o $**$ |
| Módulo | %% |

Operadores Aritméticos

| Operadores Relacionales | |
|---|---|
| Nombre | Símbolo |
| Igual | == |
| Mayor que | $>$ |
| Menor que | $<$ |
| Mayor o igual que | $>=$ |
| Menor o igual que | $<=$ |
| Módulo | %% |

Operadores Relacionales

#### 3.2.3.1. **Vectores** Un ejemplo

```
x <- c(5,2,0,4,-2,0)
```

Funciones en el vector. A continuación las siguientes funciones importantes que se pueden aplicar a un vector

```
> sum(x)
[1] 9
> length(x)
[1] 6
> mean(x)
[1] 1.5
```

| Operadores Lógicos | |
|---|---|
| Nombre | Símbolo |
| y | & |
| o | \| |
| no | ! |

Operadores Lógicos

```
> var(x)
[1] 7.1
> sd(x)
[1] 2.664583
> max(x)
[1] 5
> min(x)
[1] -2
> median(x)
[1] 1
> sort(x)
[1] -2  0  0  2  4  5
> summary(x)
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -2.0     0.0     1.0     1.5     3.5     5.0
```

## 3.3. Interfaces Gráficas

La frase *Ambientes Integrados de Desarrollo* viene del inglés Integrated Development Environment IDE [5] Estos ambientes permiten la verificación de sintaxis con colores, el almacenamiento de programas en repositorios, debugging, búsqueda y programación visual. Los IDE más populares para R en Linux son:

3.6 Rstudio [6]

3.7 Jupyter [7].

3.8 Jupyterlab [8]

3.9 Architect [9]

3.10 Eclipse [10]

entre otros.

A continuación incluimos un cuaderno desarrollado en la interface gráfica Jupyter con un tutorial en R.

[5] https://en.wikipedia.org/wiki/Integrated_development_environment
[6] https://www.rstudio.com/
[7] https://jupyter.org/
[8] https://github.com/jupyterlab/jupyterlab
[9] https://www.getarchitect.io/
[10] https://www.eclipse.org/ide/

# 4. Tutorial en R

## 4.1. Consultar documentación: help(), ls(), example, ?

```
In [1]: # use el comando  " help()
        # para ver como se usa este comando (el manual del manual)
        help()

        # use la instrucción  help(integer) para consultar sobre los enteros
        # useela instrucción help(ls) para mostrar como listar objetos
        # Ejemplo:

        a<-5
        b<-6
        ls()  # el comando "objects()"  produce el mismo resultado
        # asigna 5 a la variable "a" y 6 a la variable "b" lista los objetos definidos
        # hasta este momento.

        # para ver el manual de una función oprima interrogación y el nombre de la función.
        # Por ejemplo ?print()  muestra el manual de la función print()
        # ?print()

        # ejemplos: remueva los dos comentarios siguientes para ver una ilustración
        #require(lattice)
        # remueva comentario para ver el ejemplo de historgrama
        # example(histogram)

        # En general si escribe una función sin paréntesis se presenta
        # el prototipo de la función y algunas veces la definición de
        # misma. Por ejemplo


        print
```

```
function (x, ...)
UseMethod("print")
```

```
function (x, do.NULL = TRUE, prefix = "col")
\{
    if (is.data.frame(x) \&\& do.NULL)
        return(names(x))
    dn <- dimnames(x)
    if (!is.null(dn{[}{[}2L{]}{]}))
        dn{[}{[}2L{]}{]}
    else \{
        nc <- NCOL(x)
        if (do.NULL)
            NULL
        else if (nc > 0L)
            paste0(prefix, seq\_len(nc))
        else character()
    \}
\}
```

## 4.2. Tipos Simples

Estudiamos los tipos : enteros, dobles, reales (floating point), complejos. Realmente en R los elementos se asumen por defecto como vectores. Los escalares enteros, dobles o reales se consideran como vectores de una sola componente.

La asignación en R se escribe con el símbolo <-. Así, mientras tradicionalmente se escribe a=b para indicar que se le asigna "b´´ a "a´´ , en R decimos a <- b

Por defecto los números que conocemos como enteros son dobles en R. Los enteros en R se pueden definir colocando "L´´ al final del número o usando la función "as´´ como mostramos a continuación. La función "as´´ fuerza un número a ser de un tipo determinado. Use "?as´´ para ver el manual de esta función.

Usamos la función "cat()´´ para concatenar (imprimir) cadenas de caracteres con variables. Use ?cat() para mayor información.

### 4.2.1. Tipos enteros y dobles

Por defecto los números en R son dobles

In [2]:
```
a <- 5L
b <- as.integer(-5.25)
c <- 3

# pregunta: ¿a que clase pertence la variable "a"?.
# use ?class para obtner más ayuda.
cat("La variable a es de  clase",class(a), "\n")
cat("La variable c es de la clase",class(c))

# typeof() muestra el tipo de una variable
cat("\nEl tipo de la variable a es ",typeof(a))
# mode indica si la variable es numérica, especial (función), simbólica (nombre)
# o de lenguaje (retorna de un llamado)
cat("\nEl modo de la variable a es", mode(a))
cat("\nel tipo de la variable b es",typeof(b))
cat("\nel tipo de la variable c es", typeof(c))
cat("\nel tipo de la variable d es", typeof(c), "\n\n")
```

```
La variable a es de  clase integer
La variable c es de la clase numeric
El tipo de la variable a es  integer
El modo de la variable a es numeric
el tipo de la variable b es integer
el tipo de la variable c es double
el tipo de la variable d es double
```

In [3]:
```
# otra forma de verificar el tipo de una variable (por ejemplo, "a") es
# usando la función "is.C(a)" que verifica si el objeto "a" pertence a una clase "C"
is.integer(a)
```

TRUE

In [4]:
```
# hay un caso en que los números en R son enteros (integers).
# es decir el tipo de las secuencias definidas como sigue es entero.
a <- 2:5
print(a)
typeof(a)
```

```
[1] 2 3 4 5
```

ínteger'

```
In [5]: # observe los resultados de las siguientes preguntas
        is.integer(1)
        is.integer(1:2)
        is.integer(as.integer(1))
        is.integer(as.double(1:2))
```

FALSE
TRUE
TRUE
FALSE

```
In [6]: # los números de punto flotante se asumen como dobles
        a=0.1
        typeof(a)
```

'double'

### 4.2.2. secuencias de enteros

```
In [7]: a <- 5:16
        print(a)
```

```
 [1]  5  6  7  8  9 10 11 12 13 14 15 16
```

### 4.2.3. Números Complejos

Se les agrega la letra i

```
In [8]: a <- 3i
        b <- 3 + 2i
        typeof(a)
        typeof(b)
```

'complex'
'complex'

### 4.2.4. Tipos Booleanos

Arrojan solo los valores TRUE o FALSE (falso y verdadero)

```
In [9]: a <- 3<5
        print(a)
        typeof(a)
```

```
[1] TRUE
```

'logical'

## 4.3. Tipos Compuestos

### 4.3.1. Listas

Las listas pueden contener elementos de distintos tipos. Números, caracteres, funciones, etc. La función list() se usa para crear listas. Consulte con ?list() el manual

```
In [10]: mi_lista <- list(5,3,TRUE, c(1,3,4),FALSE, "coco")
         paste(mi_lista)  # print arroja etiquetas, cat no funciona con listas
         print(mi_lista)
```

4.1 '5' 4.2 '3' 4.3 'TRUE' 4.4 'c(1, 3, 4)' 4.5 'FALSE' 4.6 'coco'

```
[[1]]
[1] 5

[[2]]
[1] 3

[[3]]
[1] TRUE

[[4]]
[1] 1 3 4

[[5]]
[1] FALSE

[[6]]
[1] "coco"
```

### 4.3.2. Vectores

Por defecto los objetos en R son vectores de tipo doble.

```
In [11]: x <- c(1,5,3,0,5)
         typeof(x)
         length(x)
```

'double'
5

```
In [12]: x <- c(1,6,"hello","TRUE")
         x
         typeof(x)
```

4.7 '1' 4.8 '6' 4.9 'hello' 4.10 'TRUE'
'character'
se pueden usar puntos y comas para separar instrucciones y crear vectores con el opearador ":"

```
In [13]: x <- 1:-6; x; typeof(x)
```

4.11 1 4.12 0 4.13 -1 4.14 -2 4.15 -3 4.16 -4 4.17 -5 4.18 -6
ínteger'
funciones seq() y rep(). Secuencia y repecición de objetos. Use ?seq() y ?rep() para más información

```
In [14]: y = seq(1,5, by=0.2)
         typeof(y)
         y

         nombres=c(rep("José", 5))
         cat("José 5 veces:  ", nombres)
         typeof(nombres)
```

'double'
4.19 1 4.20 1.2 4.21 1.4 4.22 1.6 4.23 1.8 4.24 2 4.25 2.2 4.26 2.4 4.27 2.6 4.28 2.8 4.29 3 4.30 3.2 4.31 3.4 4.32 3.6 4.33 3.8 4.34 4 4.35 4.2 4.36 4.4 4.37 4.6 4.38 4.8 4.39 5

```
José 5 veces:   José José José José José
```

'character'

```
In [15]: # sintaxis inicial, final y número de elementos en la secuencia
         y =seq(1,3, length.out=4)
         y
```

4.40 1 4.41 1.66666666666667 4.42 2.33333333333333 4.43 3
Acceso a los elementos de un vector. En R las componentes comienzan en 1 No en 0 como en C, Python y Java, por ejemplo.

```
In [16]: print(y[0])  # en R los vectores comienzan en índice 1
         print(y[1])
         print(y[-1]) # y[-1] todos menos el primero
         print(y[-2]) # y[-2] todos menos el segundo
         y[c(1,3)] # el elemento 1 y el elemento 3
         y[c(1.2,3.6)] # numeros reales se truncan a enteros
         y[c(1,6)] # no existe el elemento 6
```

```
numeric(0)
[1] 1
[1] 1.666667 2.333333 3.000000
[1] 1.000000 2.333333 3.000000
```

4.44 1 4.45 2.33333333333333
4.46 1 4.47 2.33333333333333
4.48 1 4.49 <NA>
Usamos elementos lógicos como índice

```
In [17]: y[c(TRUE,FALSE,TRUE,FALSE)]  # filtro por verdadero/falso
         y[y<2]  # el índice es una condición lógica
```

4.50 1 4.51 2.33333333333333
4.52 1 4.53 1.66666666666667

#### 4.3.2.1. Vectores con nombre y valor

```
In [18]: x <- c("primero"=3, "segundo"=4, "tercero"=1)
         names(x)
         x["tercero"]
         x[c("tercero","primero")]
```

4.54 'primero' 4.55 'segundo' 4.56 'tercero'
**tercero:** 1
**tercero** 1 **primero** 3

#### 4.3.2.2. Vectores mixtos (de números y caracteres)

```
In [19]: alfanum=c(1,2,3,"a","b","c")  # el tipo se convierte en caracteres
         print(alfanum)
         typeof(alfanum)
         class(alfanum)
         b=as.integer(alfanum)
         print(b)  # note que no convierte los caracteres, solo los números
         typeof(b)
```

```
[1] "1" "2" "3" "a" "b" "c"
```

'character'
'character'

```
Warning message in eval(expr, envir, enclos):
"NAs introduced by coercion"
```

```
[1]  1  2  3 NA NA NA
```

ínteger'

#### 4.3.2.3. Modificación de un vector

```
In [20]: y[1] <- 1000
         print(y)
         y[y<2] <- 0  # los elementos menores que 2 se vuelven 0
         y
         y <- y[1:2]  # trunca los elementos 3 y 4
         y
         y <- NULL  # borrar el contenido del vector
         y
```

```
[1] 1000.000000    1.666667    2.333333    3.000000
```

4.57 1000 4.58 0 4.59 2.33333333333333 4.60 3
4.61 1000 4.62 0

```
NULL
```

#### 4.3.2.4. Cambio de tipo de booleano a entero

```
In [21]: x <- c(FALSE,TRUE,TRUE,FALSE)
         as.numeric(x)
```

4.63 0 4.64 1 4.65 1 4.66 0

### 4.3.3. Matrices

#### 4.3.3.1. Matrices desde secuencias o con componentes fijas

```
In [22]: y <- matrix(1:20, nrow=5, ncol=4)  # la matriz se llena por columnas, por defecto
         y
```

```
    1  6   11  16
    2  7   12  17
    3  8   13  18
    4  9   14  19
    5  10  15  20
```

```
In [23]: # forma más rápida
         y <- matrix(1:20, 5,4)
         print(y)  #note que la salida tiene un formato diferente por el print()
```

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

```
In [24]: # matriz componentes 8 de dos filas y 3 columnas
         y <- matrix(8,2,3)
         y
```

```
    8  8  8
    8  8  8
```

#### 4.3.3.2. Matricies desde vectores

#### 4.3.3.3. rbind() y cbind() ensamble por filas y columnas

```
In [25]: fila1 = c(1,3)
         fila2 = c(-2,5)
         A=rbind(fila1,fila2)
         cat("matriz A construida por filas\n")
         print(A)

         B=cbind(fila1,fila2) # por columnas.
         cat("\nmatriz B construida por columnas\n")
         print(B)
```

```
matriz A construida por filas
      [,1] [,2]
fila1    1    3
fila2   -2    5

matriz B construida por columnas
     fila1 fila2
[1,]     1    -2
[2,]     3     5
```

```
In [26]: valores <- c(1,52,-3,5,3,4)
         mimatriz <- matrix(valores, nrow=2, ncol=2, byrow=TRUE)  # ahora por fila
         # note que trunca los que no se usa

In [27]: mimatriz
```

| | |
|---|---|
| 1 | 52 |
| -3 | 5 |

#### 4.3.3.4. Acceso a elementos de la matriz

```
In [28]: # una matriz de 3x3
         A=matrix(  c(1,3,-1,2,3,5,-2,2,9),    # componentes
                  nrow=3,                     # número de filas
                  ncol=3,                     # número de columnas
                  byrow=TRUE)                 # por filas

         # una componente: la componente de la fila 1 columna 2
         cat("la matrix A es : \n")
         print(A)
         cat("componente 1,2 de la matriz es: ",A[1,2], "\n")
         # la fila 1:
         cat("fila 1 de la matriz es", A[1,])
         # la columna 3:
         cat("columna 3 de la matriz es", A[,3])
```

```
la matrix A es :
     [,1] [,2] [,3]
[1,]    1    3   -1
[2,]    2    3    5
[3,]   -2    2    9
componente 1,2 de la matriz es:  3
fila 1 de la matriz es 1 3 -1columna 3 de la matriz es -1 5 9
```

```
In [29]: y <- matrix(1:20, nrow=5, ncol=4) # la matriz se llena por columnas, por defecto

         cat("Matriz 'y' es: \n")
         print(y)

         # cuarta columna de la matriz 'y'
         cat("\n")
         print('cuarta columna')
         y[,4]
         # tercera fila de la matriz 'y'
         print('tercera fila')
         y[3,]
         # filas 2 a 4, columnas 1 a 3
         print("filas 2 a 4 y columnas 1 a 3")
         print(y[2:4, 1:3])
```

```
Matriz 'y' es:
     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
```

```
[5,]    5   10   15   20
```

```
[1] "cuarta columna"
```

4.67 16 4.68 17 4.69 18 4.70 19 4.71 20

```
[1] "tercera fila"
```

4.72 3 4.73 8 4.74 13 4.75 18

```
[1] "filas 2 a 4 y columnas 1 a 3"
     [,1] [,2] [,3]
[1,]    2    7   12
[2,]    3    8   13
[3,]    4    9   14
```

#### 4.3.3.5. Matriz de caracteres

```
In [30]: M = matrix(c('a','b','a','d','c','b'), nrow=2, ncol=3, byrow=TRUE)
         print(M)
```

```
     [,1] [,2] [,3]
[1,] "a"  "b"  "a"
[2,] "d"  "c"  "b"
```

### 4.3.4. Arreglos: Son matrices pero en general pueden tener más dimensiones

```
In [31]: v1 <- c(3,2,5)
         v2 <- c(13,2,4,4)

         # dos matrices de 3x3.
         # se llenan por columnas en el orden dado
         print("tensor 3x3x2 por columnas")
         r <- array( c(v1,v2), dim=c(3,3,2))
         print(r)
```

```
[1] "tensor 3x3x2 por columnas"
, , 1

     [,1] [,2] [,3]
[1,]    3   13    4
[2,]    2    2    3
[3,]    5    4    2

, , 2

     [,1] [,2] [,3]
[1,]    5    4    2
[2,]   13    4    5
[3,]    2    3   13
```

#### 4.3.4.1. nombrar filas y columnas

```
In [32]: v1 <- c(-1,2,3)
         v2 <- c(23,11,22,54,54,-12,7)
         nombresColumna <- c("COL1","COL2", "COL3")
         nombresFila <- c("F1","F2","F3")
         nombresMatriz <- c("Matriz1","Matriz2")

         r <- array( c(v1,v2),  dim=c(3,3,2),
                    dimnames=list(nombresFila,nombresColumna,nombresMatriz))
         print(r)
```

```
, , Matriz1

   COL1 COL2 COL3
F1   -1   23   54
F2    2   11   54
F3    3   22  -12

, , Matriz2

   COL1 COL2 COL3
F1    7    3   22
F2   -1   23   54
F3    2   11   54
```

#### 4.3.4.2. Acceso de Elementos en un Arreglo

**paste y paste0**

```
In [33]: r211 <- r[2,1,1]

         r11 <- r[1,,1]
         M1 <- r[,,1]
         M2 <- r[,,2]
         cat("componente 211:", r211)
         cat("\nFila 1 de la matriz 1:",r11)
         cat("\n\nMatriz 1:\n")

         # imprima fila por fila indentada con tab
         cat("\t", M1[1,] , "\n")
         cat("\t", M1[2,], "\n")
         cat("\t", M1[3,], "\n")
```

```
componente 211: 2
Fila 1 de la matriz 1: -1 23 54

Matriz 1:
         -1 23 54
         2 11 54
         3 22 -12
```

In [34]:
```r
# otra forma de consultar acerca de una función es usando "args()".
# use ?args  para mayor información en la función args()

print("argumentos de paste")
args(paste)
print("argumentos de paste0")
args(paste0)

print("Ejemplos con paste")
# paste arroja salida a la terminal

paste("one", "dos", "3")
paste("one","dos", ",3", sep="_")


# ejemplo con seq
paste("hola", seq(1,7,2))
paste("hola", seq(1,7,2), sep="-")
# una sola cadena separada por comas
paste("hola", seq(1,7,2), sep="-", collapse=" , ")

# los mismos ejemplos con paste0
print("Ejemplos con paste0")
paste0("one", "dos", "3")
paste0("one","dos", ",3", sep="_")


# ejemplo con seq
paste0("hola", seq(1,7,2))
paste0("hola", seq(1,7,2), sep="-")
# una sola cadena separada por comas
paste0("hola", seq(1,7,2), sep="-", collapse=" , ")

# vemos que paste0 pega los campos (usa el separador sep="")
# cuando se usa separador paste0 lo pone al final.
# mejor dicho el paste0() no usa separador,
# como está indicado en los argumentos
```

```
[1] "argumentos de paste"
```

```
function (..., sep = " ", collapse = NULL)
NULL
```

```
[1] "argumentos de paste0"
```

```
function (..., collapse = NULL)
NULL
```

```
[1] "Ejemplos con paste"
```

óne dos 3′
óne_dos_3′
4.76 ′hola 1′ 4.77 ′hola 3′ 4.78 ′hola 5′ 4.79 ′hola 7′

4.80 'hola-1' 4.81 'hola-3' 4.82 'hola-5' 4.83 'hola-7'
'hola-1 , hola-3 , hola-5 , hola-7'

```
[1] "Ejemplos con paste0"
```

ónedos3'
ónedos,3_'
4.84 'hola1' 4.85 'hola3' 4.86 'hola5' 4.87 'hola7'
4.88 'hola1-' 4.89 'hola3-' 4.90 'hola5-' 4.91 'hola7-'
'hola1- , hola3- , hola5- , hola7-'

#### 4.3.4.3. captura de salida con capture.output()

```
In [35]: r211 <- r[2,1,1]

         r11 <- r[1,,1]
         M1 <- r[,,1]
         M2 <- r[,,2]
         cat("componente 211:", r211)
         cat("\nFila 1 de la matriz 1:",r11)
         cat("\n\nMatriz 1:\n")

         # las instrucciones cat y print no son buenas para indentar
         # usamos catpure.output para imprimir la matriz M1 con indentación
         out <- capture.output(M1)
         out[1] <- paste0("M1", out[1]) # incluye la fila de caracteres M1 ...
         cat(paste("\t" ,out, collapse = "\n"))
```

```
componente 211: 2
Fila 1 de la matriz 1: -1 23 54

Matriz 1:
         M1   COL1 COL2 COL3
         F1   -1   23   54
         F2    2   11   54
         F3    3   22  -12
```

#### más de cat,print,paste,paste0 y capture.output

```
In [36]: # reconoce la instrucción de línea nueva
         a=cat("Desde cat:","perros", "gatos", "\n")
         # no reconoce la instrucción de línea nueva
         b=print("Desde print: perros gatos")
         c=paste("Desde paste:","perros","gatos")
         d=paste0("Desde paste0:","perros","gatos")
         e=capture.output(b)
```

```
Desde cat: perros gatos
[1] "Desde print: perros gatos"
```

```
In [37]: print("TIPOS")
         typeof(a)
         typeof(b)
         typeof(c)
```

```
typeof(d)
typeof(e)

print("retornos")
print(a)
print(b)
print(c)
print(d)
print(e)
```

```
[1] "TIPOS"
```

ÑULL′
′character′
′character′
′character′
′character′

```
[1] "retornos"
NULL
[1] "Desde print: perros gatos"
[1] "Desde paste: perros gatos"
[1] "Desde paste0:perrosgatos"
[1] "[1] \"Desde print: perros gatos\""
```

## 4.4. Marcos de Datos (Data Frames)

Son como matrices pero los datos pueden ser de tipos diferentes. Son como tablas o mejor bases de datos

```
In [38]: # los datos de empleados los podemos enmarcar en emp.datos
         # como sigue
         emp_datos <- data.frame(
             emp_id = c(1:4),
             emp_nombre=c("Julio", "Enrique", "Juan", "Gloria"),
             salario=c(4344,3244,1355,6533),
             fecha_de_inicio=as.Date(c("2012-01-02" , "2010-03-04" ,
                                       "2009-05-01" , "2015-03-02"))
             )

         # imprima
         print(emp_datos)
```

```
  emp_id emp_nombre salario fecha_de_inicio
1      1      Julio    4344      2012-01-02
2      2    Enrique    3244      2010-03-04
3      3       Juan    1355      2009-05-01
4      4     Gloria    6533      2015-03-02
```

```
In [39]: str(emp_datos)  # muestra la estructura
```

```
'data.frame':        4 obs. of  4 variables:
 $ emp_id         : int  1 2 3 4
 $ emp_nombre     : Factor w/ 4 levels "Enrique","Gloria",..: 4 1 3 2
 $ salario        : num  4344 3244 1355 6533
 $ fecha_de_inicio: Date, format: "2012-01-02" "2010-03-04" ...
```

```
In [40]: # resumen de la estructura. Media, mediana, primero y tercer cuarto, mín y máx
         print(summary(emp_datos))
```

```
     emp_id       emp_nombre    salario     fecha_de_inicio
 Min.   :1.00   Enrique:1    Min.   :1355   Min.   :2009-05-01
 1st Qu.:1.75   Gloria :1    1st Qu.:2772   1st Qu.:2009-12-17
 Median :2.50   Juan   :1    Median :3794   Median :2011-02-01
 Mean   :2.50   Julio  :1    Mean   :3869   Mean   :2011-09-01
 3rd Qu.:3.25                3rd Qu.:4891   3rd Qu.:2012-10-16
 Max.   :4.00                Max.   :6533   Max.   :2015-03-02
```

La función summary() trabaja con muchos tipos de datos. Los quartiles (primero y tercero) son extraídos con un algoritmo explicado en el manual de la función quantile(). Se puede consultar con la instrucción ?quantile Hay nueve tipos de algoritmos para calcular los quartiles. El tipo 7 es el que usa la función summary(). Ninguno de ellos produce los quartiles que usa Wolfram Alpha. Usemos un ejemplo más simple

```
In [41]: # extracción de datos del marco de datos
         res <- data.frame(emp_datos$emp_nombre,emp_datos$salario)
         print(res)
```

```
  emp_datos.emp_nombre emp_datos.salario
1                Julio              4344
2              Enrique              3244
3                 Juan              1355
4               Gloria              6533
```

## 4.5. Factores (Factors)

Los factores clasifican los datos por categoría (levels). Es útil cuando hay grupos grandes con pocas categorías. Por ejemplo, una población grande con solo dos géneros (M,F).

```
In [42]: # creamos un vector de colores
         datos <- c("blanco","negro","rosado","negro","amarillo","azul","rosado",
                    "blanco", "negro")
         print(datos)
         typeof(datos)
         # aplique la función factor()
         datos_factor <- factor(datos)
         typeof(datos_factor)  # el tipo factor es integer
         print(is.factor(datos_factor))
         print(datos_factor)

         radio=c(1.2,2.4,5.2,5.2,0.5,0.5,2.1,9.5,0.5)
         peso=c(2.3,1.2,53.0,2.1,5.1,0.2,3.3,5.2,0.1)
         nuevos_datos <- data.frame( datos, radio, peso)
```

```
[1] "blanco"   "negro"    "rosado"   "negro"    "amarillo" "azul"     "rosado"
[8] "blanco"   "negro"
```

'character'
ínteger'

```
[1] TRUE
[1] blanco   negro    rosado   negro    amarillo azul     rosado   blanco
[9] negro
Levels: amarillo azul blanco negro rosado
```

```
In [43]: # buscamos cuales de los campos son factores
         cat("Es el campo 'datos' factor?",is.factor(nuevos_datos$datos), "\n")
         cat("Es el campo 'radio' factor?",is.factor(nuevos_datos$radio), "\n")
         cat("Es el campo 'peso;  factor?",is.factor(nuevos_datos$peso), "\n")
         print("Imprimimos los pesos")
         print(nuevos_datos$peso)
         print(nuevos_datos$datos)
```

```
Es el campo 'datos' factor? TRUE
Es el campo 'radio' factor? FALSE
Es el campo 'peso;  factor? FALSE
[1] "Imprimimos los pesos"
[1]  2.3  1.2 53.0  2.1  5.1  0.2  3.3  5.2  0.1
[1] blanco   negro    rosado   negro    amarillo azul     rosado   blanco
[9] negro
Levels: amarillo azul blanco negro rosado
```

Observamos que el campo "datos"que fue inicialmente declarado como un vector de caracteres se convirtió en "factor"mediante la instrucción "datos_factor <- factor(datos)"sin embargo el tipo sige siendo 'character'y la variable es diferente.

```
In [44]: typeof(datos)
```

'character'

## 4.6. Operaciones Unarias y Binarias y funciones en tipos

### 4.6.1. Entre tipos simples +,-,*,^, %

```
In [45]: # con númereos reales

         a <- 5
         b <- 5.3
         cat("a+b=",a+b, "\n")
         cat("a-b",a-b, "\n")
         cat("a*b=",a*b, "\n")
         cat("a^b=",a^b, "\n")
         c <- 21
         cat("c%a=", c%%a, "\n")  # 21 módulo 5 es 1

         # con númereos complejos
```

```
        c <- a+2i
        d <- a-2i
        cat("c=",c, "\n")
        cat("d=",d, "\n")
        cat("el producto de los complejos c*d es", c*d, "\n")
        cat("la exponenciación de complejos c^d es", c^d, "\n")
        cat("parte real del número c es ", Re(c), "\n")
        cat("parte imaginaria del número c es ", Im(c), "\n")
        cat("el módulo  c es ", Mod(c), "\n")
        cat("el conjuado de   c es ", Conj(c), "\n")
        is.complex(c)
```

```
a+b= 10.3
a-b -0.3
a*b= 26.5
a^b= 5064.552
c%a= 1
c= 5+2i
d= 5-2i
el producto de los complejos c*d es 29+0i
la exponenciación de complejos c^d es 1025.943-9639.46i
parte real del número c es  5
parte imaginaria del número c es  2
el módulo  c es  5.385165
el conjuado de   c es  5-2i
```

TRUE

### 4.6.2. Relacionales

Retornan valores booleanso

```
In [46]: a <- 5
         b <- 3
         cat("a<b es", a<b, "\n")
         cat("a>b es", a>b, "\n" )
         cat("a==b es", a==b, "\n")
```

```
a<b es FALSE
a>b es TRUE
a==b es FALSE
```

### 4.6.3. Lógicos

```
In [47]: a<b & a>b
         a<b | a>b
         ! (a<b)  # este es unario pero lo clasificamos acá
```

FALSE
TRUE
TRUE

### 4.6.4. Entre escalar y vector

```
In [48]: a <- 5:10
         print(a)
         print(5*a)
         print(5+a)
         print(a**2)
```

```
[1]  5  6  7  8  9 10
[1] 25 30 35 40 45 50
[1] 10 11 12 13 14 15
[1]  25  36  49  64  81 100
```

```
In [49]: b <- c(5,5,2,-4)
         print(2*b)  # multiplicación de un escalar por un vector
         print(2^b)  # exponenciación de un número a un vector
```

```
[1] 10 10  4 -8
[1] 32.0000 32.0000  4.0000  0.0625
```

```
In [50]: y <- seq(1,2, by=0.2)
         print(y)
         print(2*y)

         cat("la longitud del vector y es",length(y))
```

```
[1] 1.0 1.2 1.4 1.6 1.8 2.0
[1] 2.0 2.4 2.8 3.2 3.6 4.0
la longitud del vector y es 6
```

### 4.6.5. Entre dos vectores

```
In [51]: a= seq(1:3)
         b=seq(3,5)
         cat("a=",a,"  ")
         cat("b=",b, "\n")
         cat("a+b=",a+b, "\n")
         c=seq(3,10)
         cat("c=",c,"\n")
         # note que a pesar a que los vectores tienen dimensiones distintas
         # la operación se hace solo son correctas las components del hasta
         # la menor de las dos dimensiones
         cat("a+c=",a+c)
```

```
a= 1 2 3   b= 3 4 5
a+b= 4 6 8
c= 3 4 5 6 7 8 9 10
```

```
Warning message in a + c:
"longer object length is not a multiple of shorter object length"
```

```
a+c= 4 6 8 7 9 11 10 12
```

```
In [52]: # dot product %*%
         print(a)
         print(b)
         cat("the product of a and b is:", a %*% b)
```

```
[1] 1 2 3
[1] 3 4 5
the product of a and b is: 26
```

### 4.6.6. Operaciones entre matricies

```
In [53]: fila1 = c(1,3)
         fila2 = c(-2,5)
         A=rbind(fila1,fila2)
         cat("matriz A construida por filas\n")
         print(A)

         B=cbind(fila1,fila2) # por columnas.
         cat("\nmatriz B construida por columnas\n")
         print(B)
```

```
matriz A construida por filas
      [,1] [,2]
fila1    1    3
fila2   -2    5

matriz B construida por columnas
     fila1 fila2
[1,]     1    -2
[2,]     3     5
```

```
In [54]: # producto the matrices
         cat("El producto de las matrices A y B es \n")
         print( A%*%B)
         cat("Suma de las matrices A y B es \n\n")
         print( A+B)
```

```
El producto de las matrices A y B es
      fila1 fila2
fila1    10    13
fila2    13    29
Suma de las matrices A y B es

      [,1] [,2]
fila1    2    1
fila2    1   10
```

```
In [55]: 3*A  # producto de escalar por matriz
         3^A  # número elevado a la matriz
```

| | | |
|---|---|---|
| fila1 | 3 | 9 |
| fila2 | -6 | 15 |
| fila1 | 3.0000000 | 27 |
| fila2 | 0.1111111 | 243 |

```
In [56]: # dimensiones de la matriz.
         length(A)
         dim(A)
```

4
4.92 2 4.93 2

### 4.6.7. Operaciones con arreglos

```
In [57]: v1 <- c(3,2,5)
         v2 <- c(13,2,4,4)

         # dos matrices de 3x3.
         # se llenan por columnas en el orden dado
         r <- array( c(v1,v2), dim=c(3,3,2))
         cat("Arreglo r es:\n")
         print(r)


         A=r[,,1]
         B=r[,,2]
         C=A+B
         cat("La suma de las dos matrices A y B es\n")
         print(C)

         w1 <- c(2,1,-1)
         w2 <- c(-13,2,-1,-4)
         s <- array( c(v1,v2), dim=c(3,3,2))
         cat("El arreglo s es:\n")
         print(s)
         # sumemos ahora dos arreglos
         t= r+s
         cat("La suma de los arreglos r y s es:\n")
         print(t)
```

```
Arreglo r es:
, , 1

     [,1] [,2] [,3]
[1,]    3   13    4
[2,]    2    2    3
[3,]    5    4    2

, , 2

     [,1] [,2] [,3]
[1,]    5    4    2
[2,]   13    4    5
[3,]    2    3   13

La suma de las dos matrices A y B es
     [,1] [,2] [,3]
[1,]    8   17    6
[2,]   15    6    8
```

```
[3,]    7    7   15
El arreglo s es:
, , 1

     [,1] [,2] [,3]
[1,]    3   13    4
[2,]    2    2    3
[3,]    5    4    2

, , 2

     [,1] [,2] [,3]
[1,]    5    4    2
[2,]   13    4    5
[3,]    2    3   13

La suma de los arreglos r y s es:
, , 1

     [,1] [,2] [,3]
[1,]    6   26    8
[2,]    4    4    6
[3,]   10    8    4

, , 2

     [,1] [,2] [,3]
[1,]   10    8    4
[2,]   26    8   10
[3,]    4    6   26
```

```
In [58]: length(r)
         dim(r)
```

18
4.94 3 4.95 3 4.96 2

### 4.6.8. Operaciones Estadísticas Simples en Datos

```
In [59]: # un vector
         x=c(2, 3 ,5, 7 , 11 ,13,17, 19 ,23 ,29, 31)
         summary(x)
         quantile(x, probs=0.25, type=9)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2.00    6.00   13.00   14.55   21.00   31.00
```

**25\ %:** 5.375

## 4.7. Instalación de paquetes

Mostramos como se instala el paquete "pryr". https://github.com/hadley/pryr Este paquete permite, entre muchas otras cosas, revisar la memory que se usa en el sistema y revisar la memoria de las variables usadas. Tambíen se puede consultar la página: https://cran.r-project.org/web/packages/pryr/index.html

In [60]:
```
# Si usted está corriendo este tutorial en jupyter es por que ya
# tiene el kernel "R" instalado,sino, lo puede instalar con las
# siguientes instrucciones (desde una sessión de R por fuera de
# Jupyter)

# instalar el paquete "IRkernel" (remueva comentario)
#install.packages('IRkernel')

# con esto puede cargar el kenel  "R" para el usuario con la instrucción
# remueva comentario
#IRkernel::installspec()
# para todo el sistema con el comando
# remueva comentario
#IRkernel::installspec(user = FALSE)

#Veamos ejemplos con el paquete "pryr"
# descomente la línea si el paquete no está instalado
#install.packages("pryr")
```

In [61]:
```
# creamos un vector de colores
datos <- c("blanco","negro","rosado","negro","amarillo",
           "azul","rosado","blanco", "negro")
print(datos)
typeof(datos)
# aplique la función factor()
datos_factor <- factor(datos)
typeof(datos)
pryr::address(datos)
pryr::address(datos_factor)
pryr::address(datos) == pryr::address(datos_factor)
pryr::mem_used()

d <- c
# note que con la asignación "<- " d y c comparten la misma memoria
print("dirección de memoria de c")

pryr::address(c)
print("dirección de memoria de d")
pryr::address(d)
```

```
[1] "blanco"   "negro"    "rosado"   "negro"    "amarillo" "azul"     "rosado"
[8] "blanco"   "negro"
```

'character'
'character'
'0x556b105fab48'
'0x556b106d3e00'
FALSE

```
39.8 MB
```

```
[1] "dirección de memoria de c"
```

'0x556b0fa6e2f8'

```
[1] "dirección de memoria de d"
```

'0x556b0fa6e2f8'

### 4.7.1. Cargar paquetes: Consulta sobre paquetes en la sesión, objetos y funciones dentro de los paquetes

```
In [62]: # para cargar un paquete en la sesión se puede hacer con la
         # instrucción "library(paquete)"
         # por ejemplo
         library(pryr)

         # otra forma de cargar un paquete es usando "require(nombre_paquete)".
         # require() arroja advertencia si el paquete no existe. library()
         # para el proceso


         # para ver que paquetes están cargados en el sistema se usa el
         # comando "search() "
         print("lista de librerías cargadas en la sesión: ")
         search()

         # luego podemos usar , para listar todos los obejtos de "pryr"
         print("lista de todos los objetos en el paquete pryr")
         ls("package:pryr")

         # o listar las funciones de pryr

         print("lista de todas las funciones en el paquete pryr")
         # remueva el comentario de la siguiente línea para ver la lista de funciones
         #lsf.str("package:pryr")
```

```
[1] "lista de librerías cargadas en la sesión: "
```

4.97 '.GlobalEnv' 4.98 'package:pryr' 4.99 'jupyter:irkernel' 4.100 'package:stats' 4.101 'package:graphics' 4.102 'package:grDevices' 4.103 'package:utils' 4.104 'package:datasets' 4.105 'package:methods' 4.106 Áutoloads' 4.107 'package:base'

```
[1] "lista de todos los objetos en el paquete pryr"
```

4.108 '%.%' 4.109 '%<a-%' 4.110 '%<c-%' 4.111 '%<d-%' 4.112 áddress' 4.113 ás.envlist' 4.114 ást' 4.115 'bits' 4.116 'bytes' 4.117 'call_tree' 4.118 'compare_size' 4.119 'compose' 4.120 'do_call' 4.121 'dots' 4.122 énclosing_env' 4.123 éval2' 4.124 éxplicit' 4.125 'f' 4.126 'fget' 4.127 'find_funs' 4.128 'find_uses' 4.129 'ftype' 4.130 'fun_args' 4.131 'fun_body' 4.132 'fun_calls' 4.133 ínspect' 4.134 ís_active_binding' 4.135 ís_promise' 4.136 ís_s3_generic' 4.137 ís_s3_method' 4.138 'make_call' 4.139 'make_function'

4.140 'mem_change' 4.141 'mem_used' 4.142 'method_from_call' 4.143 'modify_call' 4.144 'modify_lang' 4.145 ñamed_dots' 4.146 ñames_c' 4.147 óbject_size' 4.148 ótype' 4.149 'parent_promise' 4.150 'parenv' 4.151 'parenvs' 4.152 'partial' 4.153 'promise_info' 4.154 'rebind' 4.155 'refs' 4.156 'rls' 4.157 'sexp_type' 4.158 'show_c_source' 4.159 'standardise_call' 4.160 'subs' 4.161 'substitute_q' 4.162 'track_copy' 4.163 'typename' 4.164 únenclose' 4.165 úneval' 4.166 'where'

```
[1] "lista de todas las funciones en el paquete pryr"
```

```
In [63]: search()
```

4.167 '.GlobalEnv' 4.168 'package:pryr' 4.169 'jupyter:irkernel' 4.170 'package:stats' 4.171 'package:graphics' 4.172 'package:grDevices' 4.173 'package:utils' 4.174 'package:datasets' 4.175 'package:methods' 4.176 Áutoloads' 4.177 'package:base'

## 4.8. Programación en R

### 4.8.1. Instrucciones condicionales "if´´

#### 4.8.1.1. Condicional if

```
In [64]: # un ejemplo simple
         x <- 4
         y <- 3
         if (x != y )
         {
           print("x!=y")
         }
```

```
[1] "x!=y"
```

#### 4.8.1.2. Concicional "if/else´´

```
In [65]: # un ejemplo simple
         if (x<y) {
             print("x<y")
         } else {
             print("x>= y")
         }
```

```
[1] "x>= y"
```

#### 4.8.1.3. Condicional " if / else if/ else if/ .... ´´ en escalera

```
In [66]: if (x < y ) {
             print("x < y")
         } else if (x==y)  {
             print("x=y")
         } else {
             print("x > y")
         }
```

```
[1] "x > y"
```

#### 4.8.1.4. Condicional "switch

```
In [67]: switch(2,"rojo","verde","amarillo")

         switch("color", "color" = "rojo", "forma" = "círculo", "área" = 5)
```

'verde'
'rojo'

### 4.8.2. Ciclos

#### 4.8.2.1. Ciclo "for´´

```
In [68]: # ejemplos simples
         for (x in c(1,3,5,6))
             print(x)

         cat("\n")
         for(x in 2:4 ) {
             print(x)
             print(x+1)
         }
```

```
[1] 1
[1] 3
[1] 5
[1] 6

[1] 2
[1] 3
[1] 3
[1] 4
[1] 4
[1] 5
```

#### 4.8.2.2. Ciclo "while´´

```
In [69]: # ciclo simple
         x <- 1
         while(x<5)
         {
             print(x)
             print(x^2)
             cat("\n")
             x <- x+1
         }
```

```
[1] 1
[1] 1

[1] 2
[1] 4

[1] 3
[1] 9
```

```
[1] 4
[1] 16
```

## 4.9. Lectura y trabajo con datos

### 4.9.1. Datos que vienen con R. Función datos()

```
In [70]: # la instrucción data() muestra una colección de datos que pueden ser usados
         # para hacer pruebas con R
         # remueva el siguiente comentario para correr la instrucción
         # data()

         # uno de los conjuntos de datos es datasets CO2 Carbon Dioxide
         data(CO2)

         # número de registros (filas) y columnas (campos)
         dim(CO2)

         # priMeras y últimas líneas
         head(CO2)
         # use también head(1:10, CO2) para las 10 primeras lineas o
         # head(1:10,CO2)
         tail(CO2)


         # use head(CO2,2) o tail(CO2,2) para ver las primeras 2 o últimas 2 filas

         # Estadísticas
         summary(CO2)

         # halle la media de la columba "conc"
         mean(CO2$conc)
```

4.178 84 4.179 5

| Plant | Type | Treatment | conc | uptake |
|---|---|---|---|---|
| Qn1 | Quebec | nonchilled | 95 | 16.0 |
| Qn1 | Quebec | nonchilled | 175 | 30.4 |
| Qn1 | Quebec | nonchilled | 250 | 34.8 |
| Qn1 | Quebec | nonchilled | 350 | 37.2 |
| Qn1 | Quebec | nonchilled | 500 | 35.3 |
| Qn1 | Quebec | nonchilled | 675 | 39.2 |

| | Plant | Type | Treatment | conc | uptake |
|---|---|---|---|---|---|
| 79 | Mc3 | Mississippi | chilled | 175 | 18.0 |
| 80 | Mc3 | Mississippi | chilled | 250 | 17.9 |
| 81 | Mc3 | Mississippi | chilled | 350 | 17.9 |
| 82 | Mc3 | Mississippi | chilled | 500 | 17.9 |
| 83 | Mc3 | Mississippi | chilled | 675 | 18.9 |
| 84 | Mc3 | Mississippi | chilled | 1000 | 19.9 |

```
    Plant             Type         Treatment       conc          uptake
 Qn1    : 7   Quebec     :42   nonchilled:42   Min.   :  95   Min.   : 7.70
```

```
Qn2    : 7   Mississippi:42   chilled   :42   1st Qu.: 175   1st Qu.:17.90
Qn3    : 7                                    Median : 350   Median :28.30
Qc1    : 7                                    Mean   : 435   Mean   :27.21
Qc3    : 7                                    3rd Qu.: 675   3rd Qu.:37.12
Qc2    : 7                                    Max.   :1000   Max.   :45.50
(Other):42
```

435

#### 4.9.1.1. Lectura de datos de la web. Operaciones con datos

use > read.csv() # para archivos separados on coma > read.table() # para una lectura de tablas más generales

consulte el manual ?read.table donde encuentra otras opciones de lectura

```
In [71]: condados=read.csv("http://www.calvin.edu/~stob/data/counties.csv")
         dim(condados)
         names(condados)
         head(condados)
         tail(condados)
         summary(condados)
         x=condados$WaterArea
         summary(x)
         mean(x)
         sum(x)
```

4.180 3141 4.181 9

4.182 'County' 4.183 'State' 4.184 'Population' 4.185 'HousingUnits' 4.186 'TotalArea' 4.187 'WaterArea' 4.188 'LandArea' 4.189 'DensityPop' 4.190 'DensityHousing'

| | County | State | Population | HousingUnits | TotalArea | WaterArea | LandArea | DensityPop | DensityHousing |
|---|---|---|---|---|---|---|---|---|---|
| | Autauga County | Alabama | 43671 | 17662 | 604.45 | 8.48 | 595.97 | 73.3 | 29.6 |
| | Baldwin County | Alabama | 140415 | 74285 | 2026.93 | 430.58 | 1596.35 | 88.0 | 46.5 |
| | Barbour County | Alabama | 29038 | 12461 | 904.52 | 19.61 | 884.90 | 32.8 | 14.1 |
| | Bibb County | Alabama | 20826 | 8345 | 626.16 | 3.14 | 623.03 | 33.4 | 13.4 |
| | Blount County | Alabama | 51024 | 21158 | 650.60 | 5.02 | 645.59 | 79.0 | 32.8 |
| | Bullock County | Alabama | 11714 | 4727 | 626.06 | 1.04 | 625.01 | 18.7 | 7.6 |
| | County | State | Population | HousingUnits | TotalArea | WaterArea | LandArea | DensityPop | DensityHousing |
| 3136 | Sublette County | Wyoming | 5920 | 3552 | 4935.66 | 53.08 | 4882.57 | 1.2 | 0.7 |
| 3137 | Sweetwater County | Wyoming | 37613 | 15921 | 10491.17 | 65.87 | 10425.30 | 3.6 | 1.5 |
| 3138 | Teton County | Wyoming | 18251 | 10267 | 4221.80 | 214.04 | 4007.76 | 4.6 | 2.6 |
| 3139 | Uinta County | Wyoming | 19742 | 8011 | 2087.56 | 5.90 | 2081.66 | 9.5 | 3.8 |
| 3140 | Washakie County | Wyoming | 8289 | 3654 | 2242.75 | 2.69 | 2240.06 | 3.7 | 1.6 |
| 3141 | Weston County | Wyoming | 6644 | 3231 | 2400.07 | 2.21 | 2397.86 | 2.8 | 1.3 |

```
             County          State         Population       HousingUnits
Washington County:  30   Texas   : 254   Min.   :     67   Min.   :     70
Jefferson County :  25   Georgia : 159   1st Qu.:  11206   1st Qu.:   5117
Franklin County  :  24   Virginia: 135   Median :  24595   Median :  11076
Jackson County   :  23   Kentucky: 120   Mean   :  89596   Mean   :  36901
Lincoln County   :  23   Missouri: 115   3rd Qu.:  61758   3rd Qu.:  26844
Madison County   :  19   Kansas  : 105   Max.   :9519338   Max.   :3270909
(Other)          :2997   (Other) :2253
  TotalArea           WaterArea          LandArea          DensityPop
Min.   :     1.99   Min.   :   0.00   Min.   :     1.99   Min.   :    0.0
1st Qu.:   445.74   1st Qu.:   1.94   1st Qu.:   431.71   1st Qu.:   17.0
Median :   651.39   Median :   6.43   Median :   616.48   Median :   42.6
Mean   :  1207.92   Mean   :  81.71   Mean   :  1126.21   Mean   :  243.0
3rd Qu.:   985.50   3rd Qu.:  21.58   3rd Qu.:   923.19   3rd Qu.:  104.6
Max.   :147842.51   Max.   :9719.74   Max.   :145899.69   Max.   :66940.1

DensityHousing
```

```
 Min.   :    0.0
 1st Qu.:    7.9
 Median :   18.9
 Mean   :  102.7
 3rd Qu.:   45.3
 Max.   :34756.7
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    1.94    6.43   81.71   21.58 9719.74
```

81.7079783508437
256644.76

#### 4.9.1.2. Más operaciones con datos

In [72]:
```
class(condados)
#address funciona por que pryr ya está cargado en memoria
address(condados)
# descomente la próxima línea para ver los reulstados de condados[1,1]
# condados[1,1]
# descomente para ver la  segunda columna (estados)
# condados[,2]
# descomente para ver condados
# condados[1,]
# combine algunas filas. Columnas de 1 a 4
condados[c(1,5,7), 1:4]

# nombres de columna. También "names(condados)" funciona.
# liste la función colnames() escribiendo simplemente
# colnames. Se ve que colnames llama a names() .
head(colnames(condados))

# se guardan los estados en la variable "estados"
estados=condados$State

# filtro de filas con head y columnas con combinación c()
head(condados[c('State', 'LandArea')])

# índices lógicos
indice <- condados$State=='Colorado'
areaColorado=condados[indice,]$TotalArea
typeof(areaColorado)
print(areaColorado)
summary(areaColorado)

# View no funciona en Jupyter
# View muestra los datos en una hoja electrónica, en ventana
# aparte
# View(condados)

# Estructura de los Datos
```

```
print("Estructura de los Datos 'condados'")
str(condados)
```

```
# lista de valores con frecuencias
print("lista de valores con frecuencias")
table(condados$State)
```

'data.frame'
'0x556b10567090'

| | County | State | Population | HousingUnits |
|---|---|---|---|---|
| 1 | Autauga County | Alabama | 43671 | 17662 |
| 5 | Blount County | Alabama | 51024 | 21158 |
| 7 | Butler County | Alabama | 21399 | 9957 |

4.191 'County' 4.192 'State' 4.193 'Population' 4.194 'HousingUnits' 4.195 'TotalArea' 4.196 'WaterArea'

| State | LandArea |
|---|---|
| Alabama | 595.97 |
| Alabama | 1596.35 |
| Alabama | 884.90 |
| Alabama | 623.03 |
| Alabama | 645.59 |
| Alabama | 625.01 |

'double'

```
 [1] 1197.71  723.53  805.43 1355.48 2557.09 1541.12  751.37 1015.01 1781.37
[10]  396.46 1290.87 1230.44  800.33  739.90 1148.52  154.94 1068.09  842.75
[19] 1691.80 1850.89 2129.56 1533.95 2955.76  150.25 1869.60 3259.75 1123.14
[28] 1593.24 1620.95  778.06 1785.76 2161.56  383.90 1699.93 2633.86 4775.42
[37] 2586.39 1844.86 3341.11  877.70 4750.94 2039.97 2242.57 1293.87 1269.74
[46]  542.21 2210.69  687.74  973.23 1644.35 2397.73 3222.91  912.14 2368.02
[55] 3170.25  388.29 1288.49  549.60  619.25  558.96 2524.12 4021.56 2369.10
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  150.2   824.1  1534.0  1652.3  2226.6  4775.4
```

```
[1] "Estructura de los Datos 'condados'"
'data.frame':       3141 obs. of  9 variables:
 $ County        : Factor w/ 1874 levels "Abbeville County",..: 83 90 101 151 166 226 236 249 297 319 ...
 $ State         : Factor w/ 51 levels "Alabama","Alaska",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ Population    : num  43671 140415 29038 20826 51024 ...
 $ HousingUnits  : num  17662 74285 12461 8345 21158 ...
 $ TotalArea     : num  604 2027 905 626 651 ...
 $ WaterArea     : num  8.48 430.58 19.61 3.14 5.02 ...
 $ LandArea      : num  596 1596 885 623 646 ...
 $ DensityPop    : num  73.3 88 32.8 33.4 79 ...
 $ DensityHousing: num  29.6 46.5 14.1 13.4 32.8 7.6 12.8 84.3 27.2 25.4 ...
[1] "lista de valores con frecuencias"
```

```
             Alabama               Alaska              Arizona
                  67                   27                   15
            Arkansas           California             Colorado
                  75                   58                   63
         Connecticut             Deleware District of Columbia
                   8                    3                    1
```

| Florida | Georgia | Hawaii |
|---|---|---|
| 67 | 159 | 5 |
| Idaho | Illinois | Indiana |
| 44 | 102 | 92 |
| Iowa | Kansas | Kentucky |
| 99 | 105 | 120 |
| Louisiana | Maine | Maryland |
| 64 | 16 | 24 |
| Massachusetts | Michigan | Minnesota |
| 14 | 83 | 87 |
| Mississippi | Missouri | Montana |
| 82 | 115 | 56 |
| Nebraska | Nevada | New Hampshire |
| 93 | 17 | 10 |
| New Jersey | New Mexico | New York |
| 21 | 33 | 62 |
| North Carolina | North Dakota | Ohio |
| 100 | 53 | 88 |
| Oklahoma | Oregon | Pennsylvania |
| 77 | 36 | 67 |
| Rhode Island | South Carolina | South Dakota |
| 5 | 46 | 66 |
| Tennessee | Texas | Utah |
| 95 | 254 | 29 |
| Vermont | Virginia | Washington |
| 14 | 135 | 39 |
| West Virginia | Wisconsin | Wyoming |
| 55 | 72 | 23 |

#### 4.9.1.3. Datos de formato txt. Más ejemplos en "table()´´, "xtabs()´´

```
In [73]: file <-
"http://www.sr.bham.ac.uk/~ajrs/papers/        sanderson09/sanderson09_table2.txt"
         a <- read.table(file, header=TRUE, sep="|")
         typeof(a)
         length(a)
         dim(a)
         head(a,3)
         # cuente cuantos por categoría
         table(a$cctype)
         # table (matriz) de número de tipos cctype versus tipos det
         table(a$cctype, a$det)
         # una forma altenativa con xtabs()  (agrega las etiqueta "ctype"y "det")
         xtabs(~cctype + det, data=a)
         prop.table(xtabs(~ cctype + det, data=a))  # muestra proporciones
```

'list'20

4.197 20 4.198 20

| nedname | obsid | det | z | kT | kT.lo | kT.up | r500.kpc | r500.kpc.lo | r500.kpc.up | Z | Z.lo | Z.up | cctype | S01 | S01.lo | S01.up | index |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NGC 5044 | 798 | S | 0.0082 | 1.17 | 1.12 | 1.21 | 513 | 502 | 523 | 0.367 | 0.3062 | 0.454 | CC | 50 | 33.3 | 77.1 | 0.714 |
| Abell 262 | 2215 | S | 0.0163 | 2.08 | 2.00 | 2.19 | 693 | 678 | 712 | 0.379 | 0.3043 | 0.479 | CC | 126 | 105.7 | 173.9 | 0.912 |
| Abell 1060 | 2220 | I | 0.0124 | 2.92 | 2.81 | 3.03 | 829 | 812 | 845 | 0.497 | 0.4296 | 0.571 | non-CC | 191 | 183.5 | 199.6 | 0.435 |

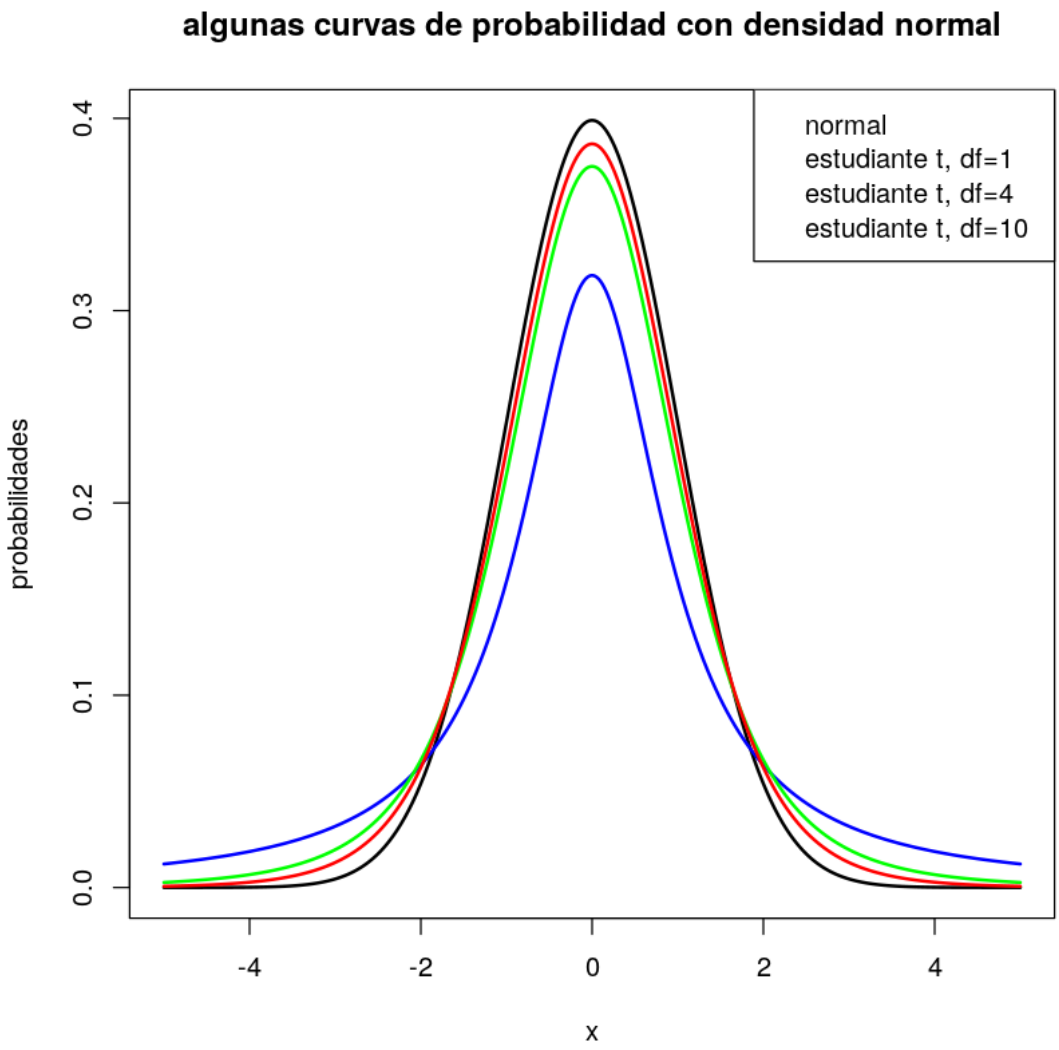
```
CC non-CC
 9      11
```

```
         I S
  CC     2 7
  non-CC 8 3


        det
cctype   I S
  CC     2 7
  non-CC 8 3


        det
cctype      I    S
  CC     0.10 0.35
  non-CC 0.40 0.15
```

## 4.10. Gráficas en R

### 4.10.1. Gráficas x-y

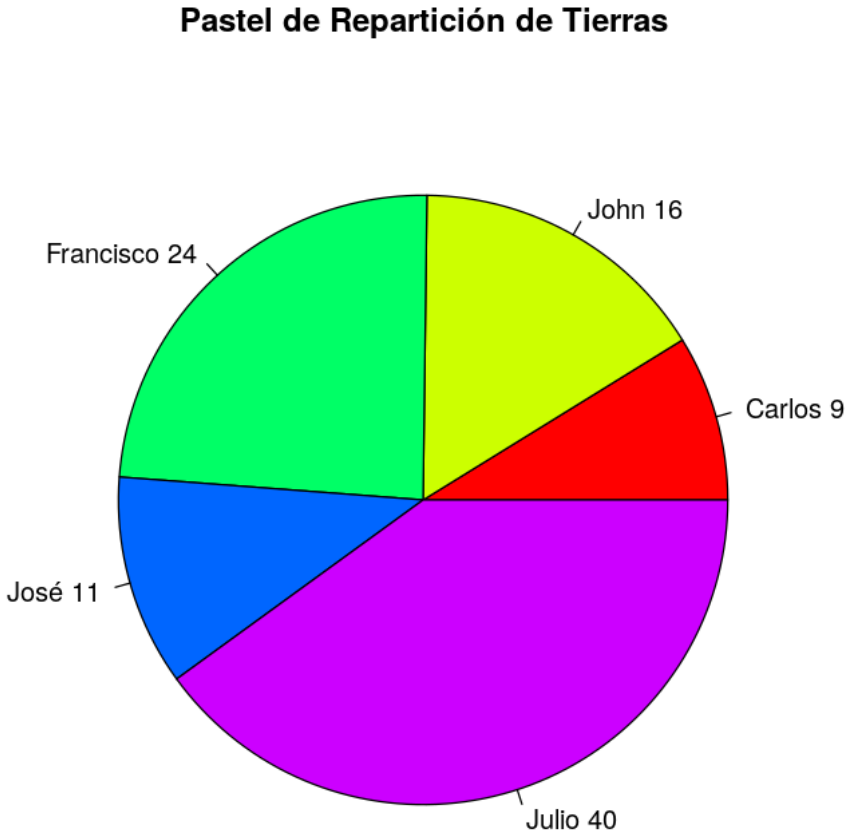
In [75]:
```r
# Student distributions qt(), pt(), rt(), dt()
# use ?qt() para consultar detalles.

# dominio
x = seq(-5,5,0.01)
plot(x, dnorm(x), type='l', lwd=2, col="black", ylab="probabilidades",
    main="algunas curvas de probabilidad con densidad normal")
lines(x, dt(x,1), lwd=2, col='blue')
lines(x, dt(x,4), lwd=2, col='green')
lines(x, dt(x,8), lwd=2, col='red')
legend("topright", col=c("black", "blue", "green", "red"),
      legend=c("normal", "estudiante t, df=1", "estudiante t, df=4",
      "estudiante t, df=10"))
```




### 4.10.2. Diagramas de Pastel (Pie-Charts)

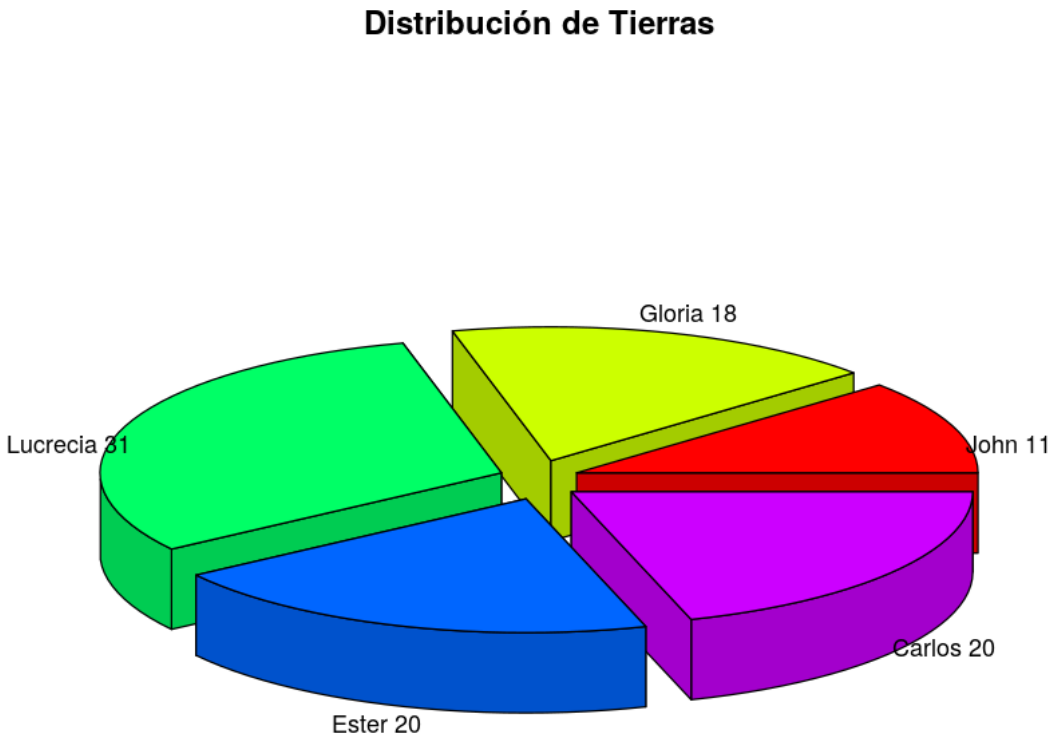
#### 4.10.2.1. 2D

```
In [76]: trozos <- c(11,20, 30, 14,50)
         etiquetas <- c("Carlos", "John", "Francisco", "José", "Julio")
         pct <- round(trozos/sum(trozos)*100)  # porcentajes
         etiquetas <- paste(etiquetas, pct)  # agregue los porcentajes
         pie( trozos, labels=etiquetas, col=rainbow(length(etiquetas)),

         main="Pastel de Repartición de Tierras"  )
```




#### 4.10.2.2. 3D

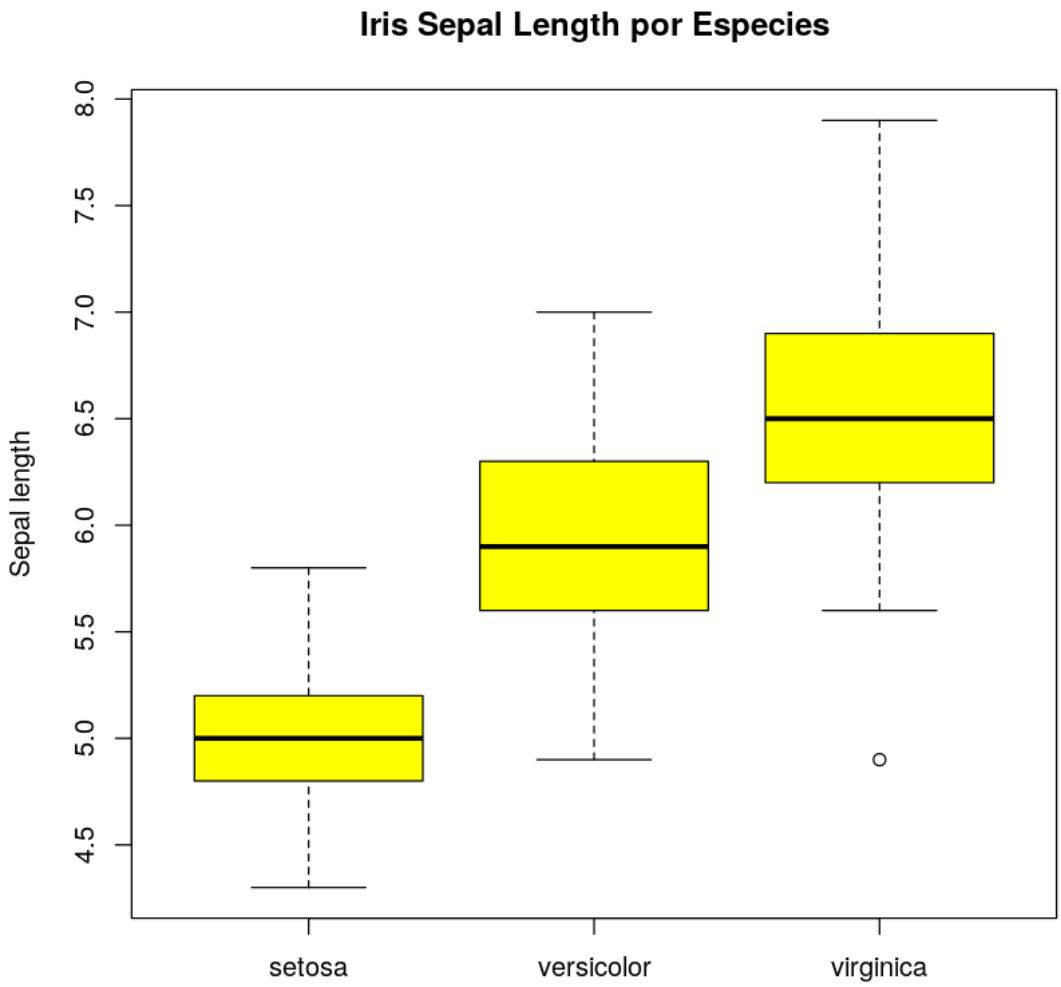
```
In [77]: # importamos la librería "plotrix"
         # remueva comentario si no está instalado el paquete "plotrix"
         # install.packages("plotrix")
         library(plotrix)

         trozos <- c(13,20,35,23,23)
         etiquetas <- c("John", "Gloria", "Lucrecia", "Ester", "Carlos")
         pct <- round(trozos/sum(trozos)*100)  # porcentajes
         etiquetas <- paste(etiquetas, pct)  # agregue los porcentajes


         pie3D(trozos, labels=etiquetas, explode=0.1, labelcex=.9, main="Distribución de Tierras")
```




### 4.10.3. Diagramas de Caja y Bigote (box and whiskers plots)

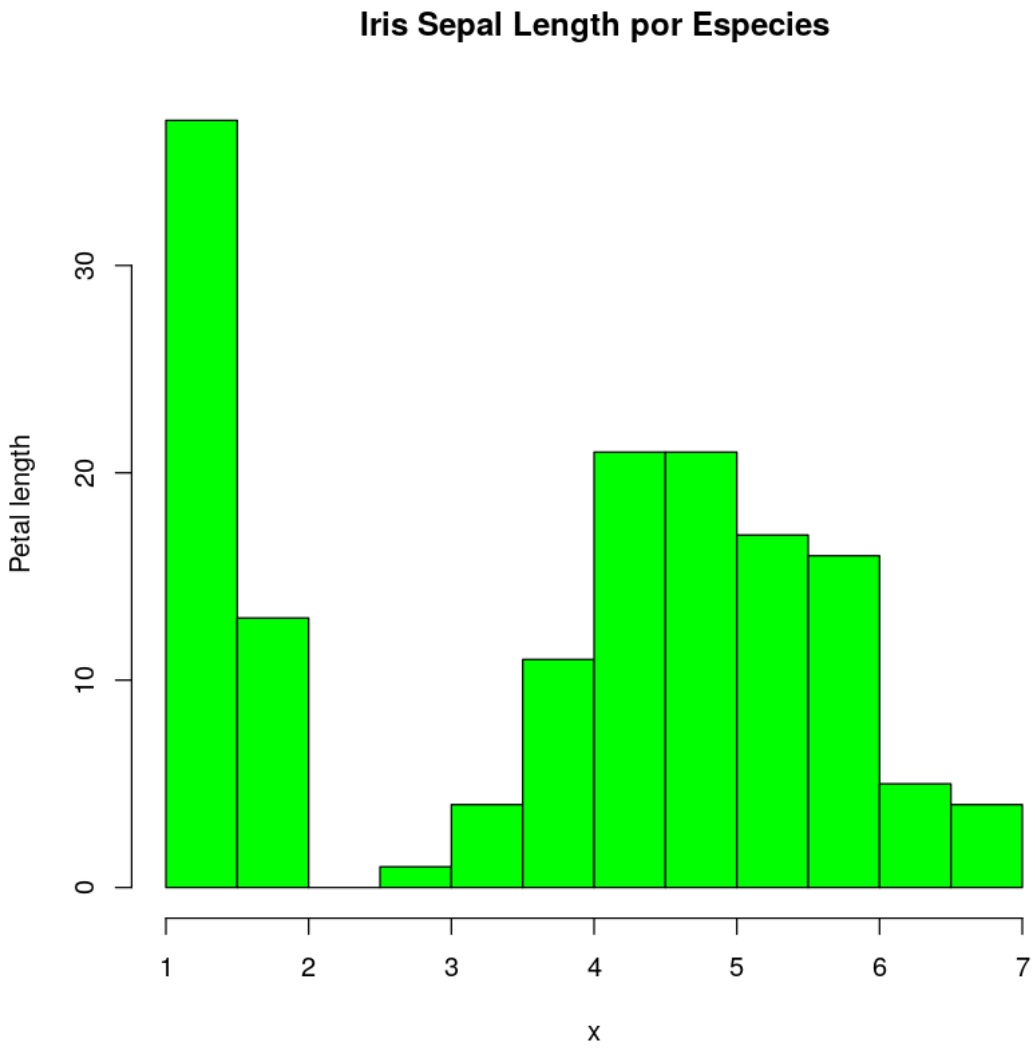
```
In [78]: data(iris)
         head(iris)
         # summary(iris)
         boxplot(Sepal.Length ~ Species, data=iris, col="yellow", ylab="Sepal length",
                 main="Iris Sepal Length por Especies")
```

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|---|---|---|---|
| 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5.0 | 3.6 | 1.4 | 0.2 | setosa |
| 5.4 | 3.9 | 1.7 | 0.4 | setosa |



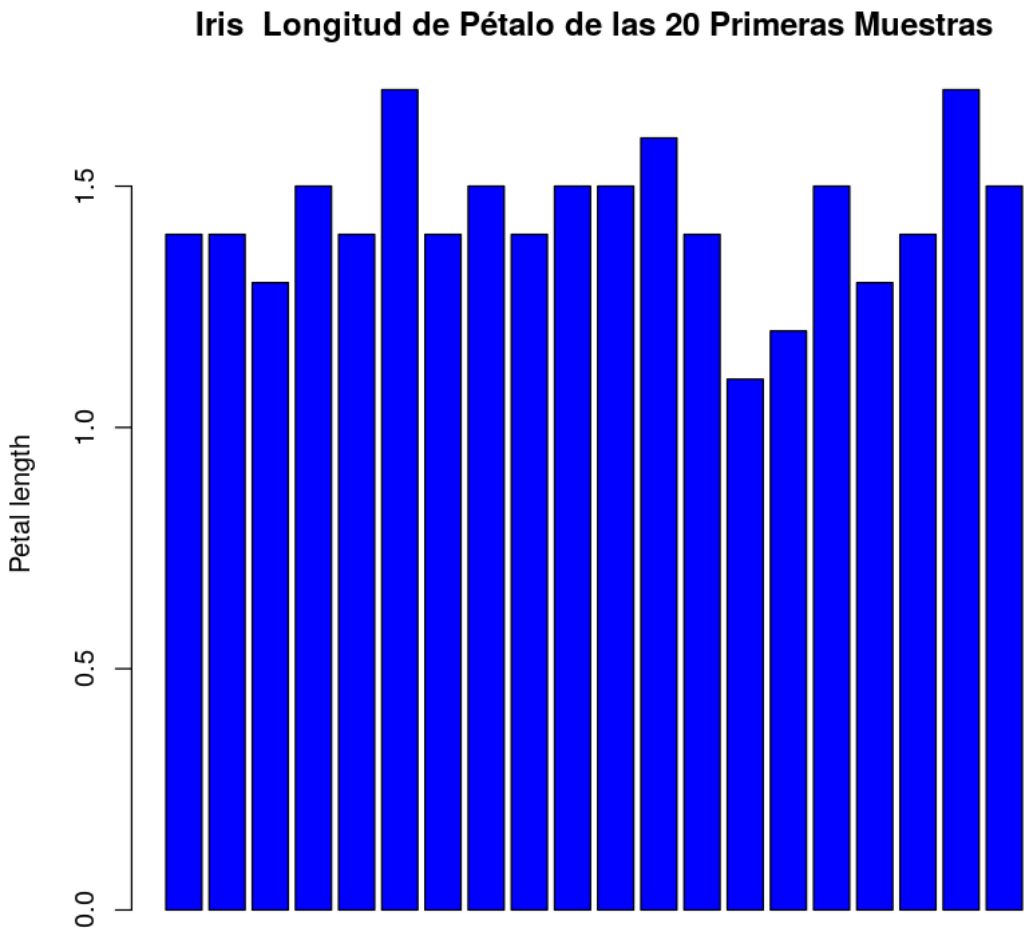

### 4.10.4. Histogramas

```
In [79]: data(iris)

         # convierta tipos para poder usar hist()

         x=as.numeric(iris$Petal.Length)

         hist(x , col="green", ylab="Petal length",
                main="Iris Sepal Length por Especies" )
```



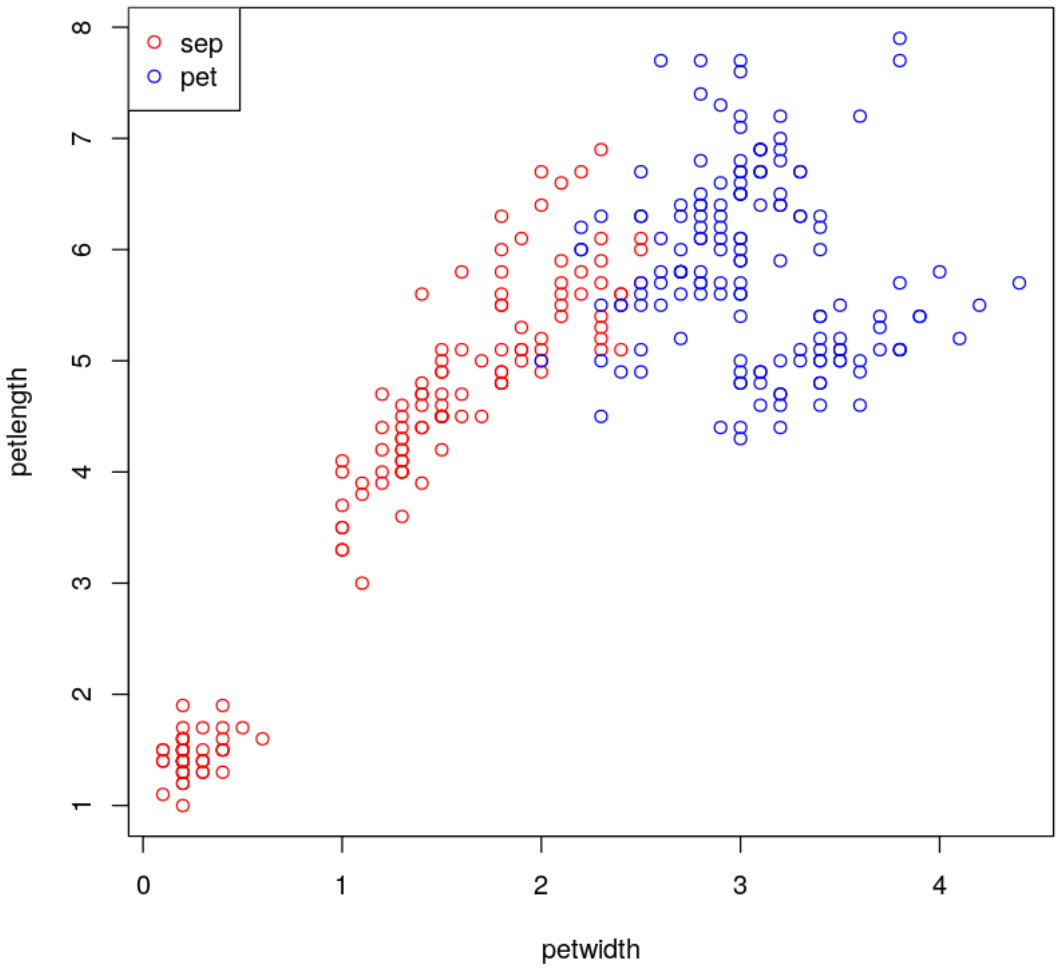

### 4.10.5. Diagramas de barras (bar charts)

```
In [81]: petalosnum <- iris$Petal.Length
         petalos2 <- petalosnum[ c(1:20)]
         barplot(petalos2,  col="blue", ylab="Petal length",
                 main="Iris  Longitud de Pétalo de las 20 Primeras Muestras" )
```




### 4.10.6. Diagramas de Dispersión (Scatter Plots)

```
In [82]: # tomamos varias grupos. Sepal.Width vs Petal.Width y
         # Petal.Width vs Petal.Length
         sepwidth <- iris$Sepal.Width
         seplength <- iris$Sepal.Length
         petwidth <- iris$Petal.Width
         petlength <- iris$Petal.Length

         plot(petwidth,petlength,xlim=range(c(petwidth,sepwidth))
              ,ylim=range(c(petlength,seplength)),col="red")
         points(sepwidth,seplength, col="blue")

         legend("topleft", col=c("red", "blue"),
                legend=c("sep", "pet"), pch=1)
```


sep
pet
petlength
petwidth
1
2
3
4
5
6
7
8
0
1
2
3
4